ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 43, No. 10, pp. 1131-1143, 2016. 10
http://dx.doi.org/10.5626/JOK.2016.43.10.1131

대용량 그래프 압축과 마이닝을 위한 그래프 정점 재배치 분산 알고리즘

(A Distributed Vertex Rearrangement Algorithm for Compressing and Mining Big Graphs)

박 남 용 † 박 치 완 † 강 유 ††
(Namyong Park) (Chiwan Park) (U Kang)

요 약 수십억 개 간선들로 구성된 대용량 그래프를 어떻게 효과적으로 압축할 수 있을까? 정점 재배치를 통해 인접 행렬의 0이 아닌 값들을 집중시키면 그래프를 효율적으로 압축할 수 있을 뿐 아니라 페이지랭크 등 여러 그래프 마이닝 알고리즘의 수행 속도를 개선할 수 있다. 최신 정점 재배치 기법인 SlashBurn은 실세계 네트워크의 멱법칙 특성을 활용하는 실세계 그래프에 효과적인 방법이다. 하지만 단일 머신 기반으로 설계되어 대용량 그래프에 대해 처리 속도가 현저히 느려지거나 적용이 불가능한 한계가 있다. 본 논문에서는 이러한 한계를 극복하기 위한 분산 SlashBurn을 제안한다. 분산 SlashBurn은 대규모의 정점 재배치 프로세스를 분산 처리하여 대용량 그래프를 기존 방법보다 훨씬 빠르고 확장성 있게 처리한다. 대용량 실세계 그래프들에 대한 실험 결과, 분산 SlashBurn은 단일 머신 SlashBurn보다 45배 이상 빠르게 동작하였고, 16배 이상 큰 그래프를 처리할 수 있었다.

키워드: 그래프 압축, 그래프 마이닝, 분산 알고리즘, SlashBurn

Abstract How can we effectively compress big graphs composed of billions of edges? By concentrating non-zeros in the adjacency matrix through vertex rearrangement, we can compress big graphs more efficiently. Also, we can boost the performance of several graph mining algorithms such as PageRank. SlashBurn is a state-of-the-art vertex rearrangement method. It processes real-world graphs effectively by utilizing the power-law characteristic of the real-world networks. However, the original SlashBurn algorithm displays a noticeable slowdown for large-scale graphs, and cannot be used at all when graphs are too large to fit in a single machine since it is designed to run on a single machine. In this paper, we propose a distributed SlashBurn algorithm to overcome these limitations. Distributed SlashBurn processes big graphs much faster than the original SlashBurn algorithm does. In addition, it scales up well by performing the large-scale vertex rearrangement process in a distributed fashion. In our experiments using real-world big graphs, the proposed distributed SlashBurn algorithm was found to run more than 45 times faster than the single machine counterpart, and process graphs that are 16 times bigger compared to the original method.

Keywords: graph compression, graph mining, distributed algorithm, SlashBurn

· 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.R0190-15-2012, 빅데이터 처리 고도화 핵심 기술개발 사업 총괄 및 고성능 컴퓨팅 기술을 활용한 성능 가속화 기술 개발).

† 비 회 원 : 서울대학교 컴퓨터공학부
namyong.park@snu.ac.kr
chiwanpark@snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
(Seoul National Univ.)
ukang@snu.ac.kr
(Corresponding author임)

논문접수 : 2016년 5월 9일
(Received 9 May 2016)
논문수정 : 2016년 7월 11일
(Revised 11 July 2016)
심사완료 : 2016년 7월 12일
(Accepted 12 July 2016)

정보과학회논문지 제43권 제10호(2016. 10)

1. 서 론

그래프의 인접 행렬 상에서 간선들이 어떻게 분포되어 있는지는 그래프 마이닝 알고리즘의 성능에 큰 영향을 미칠 수 있는 중요한 요소이다. 그래프의 간선 수가 동일하더라도, 즉 인접 행렬에서의 0이 아닌 값(non-zero)의 개수가 같더라도 0이 아닌 값들이 고르게 퍼져 있는 경우보다 집중되어 있는 경우에 훨씬 효율적으로 그래프를 압축할 수 있다[1]. 또한 인접 행렬에서 0이 아닌 값들이 집중되도록 정점들을 재배치 하는 것을 통해 많은 그래프 마이닝 알고리즘의 수행 속도를 개선할 수 있다[2]. 예를 들어, PageRank, Random Walk with Restart 등 널리 사용되는 그래프 마이닝 알고리즘들은 행렬-벡터 곱 연산을 반복적으로 수행하는 경우가 많은데 이의 최신 방법인 블록 단위 곱 연산 기법[3] 적용 시 인접 행렬이 적은 수의 밀집 블록으로 구성되어 있는 경우가 많은 수의 희소 블록으로 구성되어 있는 경우보다 속도가 빠르다. 이것은 간선들이 적은 수의 블록들에 밀집되어 있는 경우 상대적으로 적은 수의 블록만 전송하면 되고, 각각의 밀집 블록 또한 보다 효율적으로 압축할 수 있기 때문이다.

SlashBurn 알고리즘[2]은 정점 재배치를 위한 최신 기법으로 실세계 그래프들의 멱법칙(power-law) 특성을 활용한다. 멱법칙 분포를 따르는 대부분의 실세계 그래프는 높은 차수(degree)를 가지는 소수의 hub 정점들과 낮은 차수를 가지는 나머지 대부분의 정점들로 구성되고, 그래프로부터 hub 정점과 hub 정점에 연결된 간선들을 제거하면 그래프가 여러 개의 작은 연결 요소(connected component)들로 분리된다는 특징을 가진다. 이 연결 요소들은 가장 큰 크기를 가지는 거대 연결 요소(giant connected component)와 spoke라고 불리는 나머지 비 거대 연결 요소들로 나눌 수 있는데, 거대 연결 요소는 원래 그래프에 비해 훨씬 작은 크기를 가진다는 특징이 있다[2]. SlashBurn은 이러한 실세계 그래프의 특징을 이용해 hub 정점과 spoke 정점들을 적절히 재배치하고, 거대 연결 요소에 대해 동일한 과정을 재귀적으로 수행하여 인접 행렬의 0이 아닌 값들이 집중되도록 한다.

한편 기존의 SlashBurn 알고리즘은 단일 머신 기반으로 설계되었기 때문에 단일 머신의 메모리와 처리 속도의 한계로 인하여 많은 실세계 그래프들에 대해서 처리 속도가 현저하게 느려지거나 혹은 메모리 부족 문제 등으로 인해 적용이 불가능해지는 한계를 갖고 있다. 실제로 여러 대규모 실세계 그래프들에 단일 머신 SlashBurn을 적용했을 때, 정점 개수가 수백만 개, 간선 개수가 수천만 개에 이르는 그래프부터는 메모리와 처리 속도의 제약으로 인해 기존의 SlashBurn 알고리즘을 사용하기가 힘들거나 불가능하였다.

본 논문에서는 이러한 한계를 극복하기 위한 분산 SlashBurn 알고리즘을 제안한다. 분산 SlashBurn은 대규모 정점 재배치 프로세스의 분산 처리를 통해 대용량 그래프를 빠르고 확장성 있게 처리한다. 첫 번째로, 분산 SlashBurn에서는 수행 시간 중 큰 비중을 차지하는 연결 요소 알고리즘의 수행 속도와 확장성을 개선하기 위해 최신 분산 연결 요소 알고리즘을 활용하였다. 두 번째로, 로드 밸런싱을 고려한 분산 처리를 통해 정점 재배치 결과를 출력함으로써 대규모 데이터에 대해 확장성 있게 동작할 수 있도록 하였다. 다양한 종류의 실세계 그래프와 합성 그래프에 대한 실험 결과는 기존의 단일 머신 SlashBurn으로는 수행할 수 없었던 대용량 그래프에 대한 정점 재배치를 분산 SlashBurn을 통해 효과적으로 수행할 수 있음을 보여준다.

본 논문의 주요 기여 사항은 다음과 같다:

- **알고리즘.** 본 논문이 제안하는 분산 SlashBurn 알고리즘은 대규모 그래프 데이터에 대해 기존의 단일 머신 기반 SlashBurn 알고리즘보다 효율적이며 확장성 있게 동작한다. 또한 분산으로 hub와 spoke 정점을 재배치하는 여러 경우들을 각각 고려하여 재배치 결과로부터 누락되거나 중복되는 정점들이 없도록 설계되었다.
- **확장성.** 분산 SlashBurn은 단일 머신 SlashBurn보다 16배 이상 큰 실세계 그래프 데이터를 처리할 수 있으며 클러스터를 확장함에 따라 보다 큰 그래프의 처리도 가능할 것으로 기대된다.
- **수행 속도.** 간선 개수가 수억 개 혹은 그 이상 되는 그래프들에 대해서 분산 SlashBurn은 단일 머신 SlashBurn보다 45배 이상 빠르게 동작한다.

논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장에서 SlashBurn 알고리즘과 최신 분산 연결 요소 알고리즘인 Alternating 알고리즘에 대해 살펴본다. 3장에서 본 논문에서 제안하는 분산 SlashBurn 알고리즘을 설명한 후, 4장에서 실험 결과를 제시한다. 5장에서 본 논문과 관련된 연구들에 대해 살펴보고 6장에서 마무리한다.

2. 배경지식

이 장에서는 정점 재배치를 위한 최신 기법인 SlashBurn 알고리즘과 그래프에서 연결 요소를 찾기 위한 최신 분산 기법인 Alternating 알고리즘에 대해 살펴본다.

2.1 SlashBurn 알고리즘

SlashBurn 알고리즘[2]은 효과적인 정점 재배치를 위한 최신 기법으로 그래프 G를 입력으로 받아 G의 인접 행렬의 0이 아닌 값들이 집중되도록 정점을 재배치한다.

Input: 1) Edge set E of a graph $G=(V, E)$
2) a constant k (number of hub vertices, default=1).
Output: Array Γ containing the ordering $V \rightarrow [n]$.

Step 1: Remove k-hubset from G to create the new graph G'. Prepend the removed k-hubset to Γ.
Step 2: Find connected components in G'. Append the vertices in non-giant connected components to the back of Γ such that vertices belonging to a larger connected component are placed before those belonging to a smaller one.
Step 3: Set G to be the giant connected component (GCC) of G'. Return to step 1 and repeat until the size of GCC is smaller than k.

그림 1 SlashBurn 알고리즘[2]
Fig. 1 SlashBurn algorithm[2]

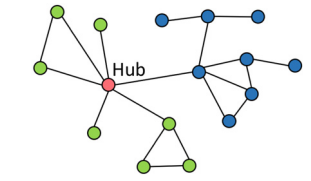
SlashBurn은 실세계 그래프의 대부분이 멱법칙 분포를 따른다는 특성을 이용한다. 멱법칙 특성을 지닌 그래프는 높은 차수를 지닌 소수의 hub 정점들과 낮은 차수를 가진 나머지 대부분의 정점들로 구성되어, 그래프에서 hub 정점과 hub 정점에 인접한 간선들을 제거하면 작은 크기의 여러 연결 요소들로 그래프가 분리된다는 특징이 있다.

SlashBurn 알고리즘은 세 단계로 구성된다(그림 1). 알고리즘의 단계 1과 단계 2에서는 주어진 그래프 G로부터 입력으로 주어진 k개 만큼의 hub 정점을 찾아 제거하고 나머지 그래프인 G'에 대해 연결 요소 알고리즘을 수행하여 G에 속한 정점들을 다음의 세 그룹으로 나눈다:

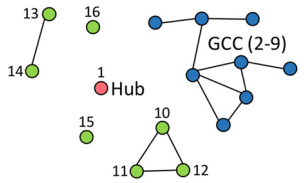
- k-hub 집합: 차수 기준 상위 k개의 정점들의 집합
- GCC: 단계 1에서 발견한 G'의 거대 연결 요소(Giant Connected Component)에 속하는 정점들
- Spokes: G'에서 거대 연결 요소에 속하지 않는 다른 모든 정점들

k-hub 집합과 spokes에 속한 정점들의 재배치 정보가 확정된 후, 그래프 G'의 GCC를 대상으로 단계 1부터 다시 수행된다.

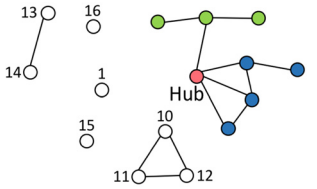
그림 2는 SlashBurn 알고리즘이 주어진 그래프에 대해서 두 차례 반복되는 과정을 보여준다. (a), (b)는 각각 첫 번째 라운드에서 SlashBurn 적용 전후의 그래프를 나타내고, (c), (d)는 각각 두 번째 라운드 중 SlashBurn 적용 전후의 그래프를 나타낸다. SlashBurn 수행 시 분홍색으로 표시되어 있는 hub 정점이 제거된 후 그래프는 하나의 GCC(파란색)와 여러 개의 작은 spoke들(연두색, 연결 요소들 중 GCC를 제외한 나머지)로 나뉜다. 이 중 hub 정점은 가장 작은 식별자를 부여받고((b)에서 1), GCC 정점들은 그 다음 식별자부터 GCC 크기만한 식별자 구간을 할당받는다((b)에서 2-9). Spoke 정점들은 큰 spoke에 속해있는 정점들부터 남아 있는 가장 작은 식별자들을 순서대로 부여받는다((b)에서 10-16). 다음 번 라운드에서는 GCC에 대해서 다시 단계 1부터 같은 과정을 수행한다.



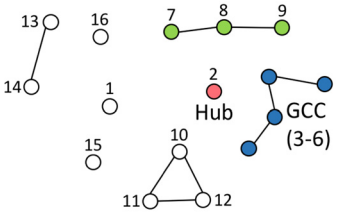
(a) Iteration 1: before SlashBurn



(b) Iteration 1: after SlashBurn



(c) Iteration 2: before SlashBurn



(d) Iteration 2: after SlashBurn

그림 2 SlashBurn 알고리즘의 작동 예시. 숫자는 SlashBurn 알고리즘을 통해 재배치된 인접행렬에서의 각 정점 순서이다.
Fig. 2 Running example of SlashBurn algorithm. Numbers represent the order of each vertex in the adjacency matrix rearranged by SlashBurn algorithm

2.2 Alternating 알고리즘

Alternating 알고리즘[4]은 맵리듀스 기반의 최신 분산 연결 요소 알고리즘이다. 아래에서는 Alternating 알고리즘의 동작 방식에 대해 설명한다.

LargeStar와 SmallStar 연산. Alternating 알고리즘은 그림 3과 4에 설명된 LargeStar와 SmallStar라는

Map (u, v):
  Emit (u, v), (v, u).

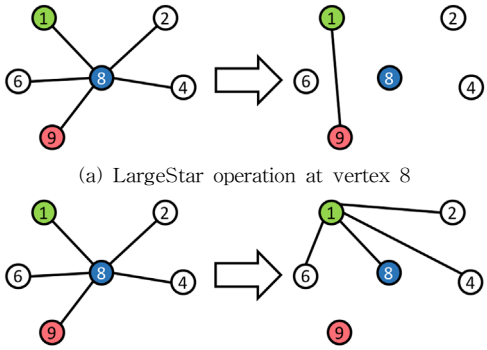
Reduce (u, $\Gamma(u)$):
  Let m = argmin $_{v \in \Gamma+(u)}$ l_V.
  Emit (v, m) for all $v \in \Gamma(u)$ where $l_v > l_u$.

그림 3 LargeStar 맵-리듀스 연산[4]
Fig. 3 LargeStar map-reduce operation[4]

Map (u, v):
  if $l_v \leq l_u$
    Emit (u, v).
  else
    Emit (v, u).

Reduce (u, $N \subseteq \Gamma(u)$):
  Let m = argmin $_{v \in N \cup \{u\}}$ l_V.
  Emit (v, m) for all $v \in N$.

그림 4 SmallStar 맵-리듀스 연산[4]
Fig. 4 SmallStar map-reduce operation[4]



(a) LargeStar operation at vertex 8

(b) SmallStar operation at vertex 8

그림 5 LargeStar와 SmallStar 연산의 수행 예시. 파란색은 연산이 수행되는 정점을 나타낸다. 빨간색은 가장 큰 식별자를 가진 정점을 가리키고, 초록색은 가장 작은 식별자를 가진 정점을 가리킨다.

Fig. 5 Examples of running LargeStar and SmallStar operations. Blue color represents the vertex where the operation is performed. Red and green color indicates the vertex with the largest and the smallest label, respectively

두 가지 간단한 맵리듀스 연산의 반복으로 구성된다. 그림 3과 4에서 사용된 기호들의 의미는 다음과 같다:

- (u, v): 그래프의 정점 u와 v를 잇는 간선
- $\Gamma(u)$: 정점 u의 이웃 정점들의 집합
- $\Gamma+(u)$: 정점 u와 u의 이웃 정점들의 집합. 즉, $\Gamma(u) \cup \{u\}$
- l_u : 정점 u의 숫자 식별자

LargeStar와 SmallStar 연산은 연산이 수행되는 정점 u와 직접적으로 연결된 이웃 정점들 $N \subseteq \Gamma(u)$ 만을 대상으로 하는 분산 알고리즘으로 LargeStar에서 N은 주어진 정점 u 보다 큰 식별자를 가진 이웃 정점들이고, SmallStar에서 N은 주어진 정점보다 식별자가 작거나 같은 이웃 정점들이라는 차이가 있다. LargeStar와 SmallStar 연산이 정점 u에서 수행될 때 $N \cup \{u\}$에 속한 정점 중 가장 작은 식별자를 가진 정점을 m이라 하면, 두 연산은 N에 속한 모든 정점 v에 대하여 간선 (v, u)를 (v, m)으로 교체한다. LargeStar와 SmallStar 연산은 그래프의 간선 구조를 수정할 수 있지만, 모든 정점에 대해 병렬적으로 수행하면 그래프의 전체적인 간선 수가 증가하지 않는다는 것과 정점 간의 연결성이 유지된다는 것이 증명되어 있다[4]. 그림 5에서 두 연산의 수행 예시를 볼 수 있다.

그래프의 간선에 정점 식별자가 큰 쪽에서 작은 쪽으로 방향성을 부여하면 각 연산이 갖는 의미를 좀 더 직관적으로 볼 수 있다. 식별자가 큰 정점이 작은 식별자를 갖는 정점을 가리키는 방향성 그래프에서 LargeStar는 연산이 적용된 정점의 자식 정점들을 정점의 부모 정점들 중 가장 작은 식별자를 가진 부모 정점으로 연결한다. 즉 자식 정점들은 연결되어 있던 부모 정점을 건너 뛰고 부모 정점의 부모 정점에 직접 연결되게 된다. 이에 따라 LargeStar 연산을 그래프의 모든 정점들에 적용하면 그래프의 높이가 줄어든다. 한편 SmallStar는 연산이 적용된 정점의 부모 정점들을 가장 작은 식별자를 가진 부모 정점에게로 연결한다. 결과적으로 SmallStar는 루트 레벨에 있는 정점들의 개수를 일정 비율로 줄여나간다.

Alternating 알고리즘. Alternating(그림 6)은 앞서 살펴본 LargeStar와 SmallStar 연산을 반복적으로 호출

Input: 1) Edges given as a list of key-value pairs (u, v), 2) a unique label l_V for every vertex $v \in V$.
Output: Edges given as a list of key-value pairs (u, c) where u is a vertex label, and c is the label of connected component vertex u belongs to.

Do
  LargeStar
  SmallStar
Until the graph gets converged

그림 6 Alternating 알고리즘[4]
Fig. 6 Alternating algorithm[4]

하는 알고리즘으로 각 연산에 의한 그래프 상의 변화가 더 이상 없을 때까지 반복된다. 앞에서 살펴본 두 연산이 갖는 의미로부터 알 수 있듯이, 알고리즘 수행 결과 입력 그래프는 서로 분리된 트리들(가장 작은 식별자를 가진 정점에 다른 모든 정점들이 연결된 트리)의 집합으로 변화되며, 여기서 하나의 트리가 하나의 연결 요소에 해당된다.

3. 분산 SlashBurn

이 장에서는 SlashBurn 알고리즘의 기존 단일 머신 기반 구현이 갖는 한계에 대해 설명하고 이를 극복하기 위한 분산 SlashBurn 알고리즘을 제안한다.

3.1 단일 머신 기반 구현의 한계

단일 머신 상에서 수행되도록 구현된 SlashBurn 알고리즘의 경우 단일 머신의 메모리 용량과 CPU 처리 속도의 물리적 한계로 인한 다음 두 가지 문제를 가진다. 첫 번째로, 대용량 그래프에 대해 수행되는 경우 알고리즘에서 사용되는 여러 데이터들을 단일 머신의 메모리에 모두 적재하기가 어렵다. 예를 들어, MATLAB으로 구현된 기존 단일 머신 기반 SlashBurn 알고리즘 구현체[5]는 그래프의 인접 행렬을 MATLAB의 희소 행렬(sparse matrix)로 읽어 들이는데, 이 경우 그래프의 정점과 간선 수에 비례하는 만큼의 메모리가 인접 행렬에 필요하다. 두 번째로, 설령 단일 머신의 메모리 용량이 대규모 그래프 데이터를 처리할 수 있을 정도 크기가 된다고 하더라도 머신 한대에서 모든 연산을 수행해야 하므로 대규모 데이터를 처리하는데 굉장히 오랜 시간이 걸릴 수 있다. 실제로 다양한 크기의 실세계 그래프 데이터들에 대해서 실험한 4.3장의 결과를 보면 간선 개수가 수억 개 이상인 그래프들에 대해서는 제한된 시간 안에 단일 머신 기반 알고리즘을 통해 결과를 얻을 수 없었고, 결과를 얻을 수 있었던 경우에도 분산 SlashBurn에 비해 훨씬 많은 시간을 필요로 하였다. 또한 간선 개수가 십 억개 이상인 그래프들에 대해서는 메모리 부족 문제로 인하여 단일 머신 기반 알고리즘의 적용이 불가능했다.

3.2 분산 SlashBurn 알고리즘 개요

본 논문에서 제안하는 분산 SlashBurn 알고리즘(그림 7)은 단일 머신 SlashBurn 알고리즘의 한계를 넘어 대용량 그래프 데이터에 대해 효과적이며 확장성 있게 동작하는 것을 목표로 한다. 효율적인 분산 SlashBurn 알고리즘을 설계하기 위해서는 다음 두 가지 과제를 해결해야 한다.

- 대용량 그래프에 대한 정점 재배치를 어떻게 정확하게 수행할 것인가? 여러 대의 머신에서 분산으로 처리되는 정점 재배치 프로세스는 누락되거나 중복되는 경우없이 모든 정점을 포함하도록 설계되어야 한다.
- 대용량 그래프에 대한 정점 재배치를 어떻게 효율적으로 수행할 것인가? SlashBurn은 반복적인 알고리즘으로 대용량 데이터의 처리를 위해서는 알고리즘의 각 과정이 확장성 있고 효율적으로 동작하도록 설계되어야 한다.

위의 문제들을 해결하기 위하여 분산 SlashBurn에서 사용한 방법은 다음과 같다:

- 여러 hub와 spoke 정점 생성 케이스를 고려한 분산 알고리즘 설계(3.4장). 분산으로 hub와 spoke 정점을 재배치하는 과정에서 발생 가능한 이슈들을 고려하여 분산 알고리즘을 설계한다.

Input: 1) Edge list E of a graph $G=(V, E)$
2) A constant k (number of hub vertices, default=1).
3) A constant s (maximum size of a split)
Output: Set of files created on HDFS containing the ordering $V \rightarrow [n]$.

Step 1.

1. Compute the degree of each vertex in G and identify k-hubset. Output k-hubset.
2. Get incident edges of k-hubset and identify single-vertex spokes. Remove incident edges from E to create the new edge list E'.

Step 2.

1. Using distributed connected component (CC) algorithm, find CCs in E'.
2. Divide E' into the giant connected component (GCC) edges and the non-GCC edges.
3. From the size of each CC, compute the rearrangement range for each non-GCC such that vertices belonging to a larger CC are placed before those belonging to a smaller CC.
4. Compute the aggregation key or split id of each non-GCC depending on the size of each non-GCC and the given split size s.
5. Output vertices in the non-GCC edges in a load-balanced manner and single-vertex spokes if any.

Step 3.

1. Set E to be the GCC edges. Return to step 1 and repeat until the number of vertices in GCC is smaller than k.

그림 7 분산 SlashBurn 알고리즘
Fig. 7 Distributed SlashBurn algorithm

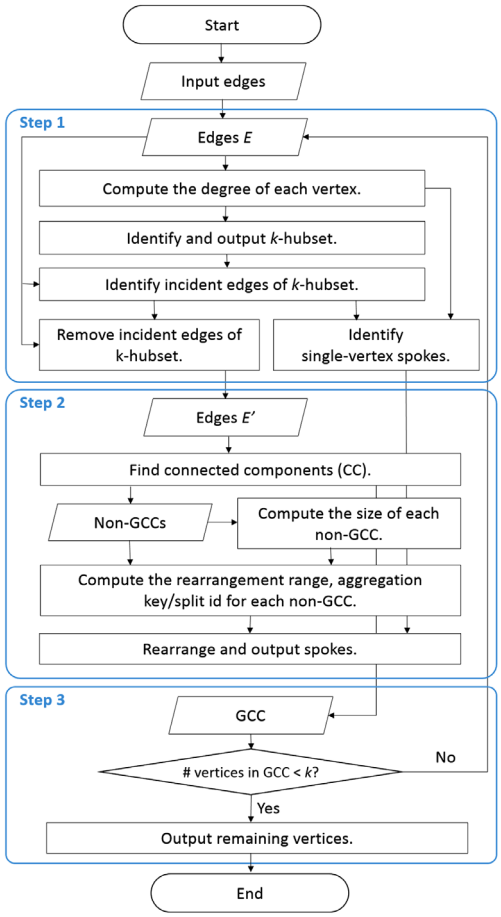


그림 8 분산 SlashBurn 알고리즘의 순서도. 알고리즘의 세부 단계 간의 관계와 각 세부 단계의 입출력을 보여준다.

Fig. 8 Flowchart of distributed SlashBurn algorithm showing the relationship between substeps of the algorithm and the input and output of each substep

- 분산 연결 요소 알고리즘의 사용(3.5장). 대용량 그래프에서 효율적이고 확장성 있게 연결 요소를 찾기 위한 방법으로 맵리듀스(MapReduce) 기반의 최신 분산 연결 요소 알고리즘을 사용한다.
- 정점 재배치 결과의 분산 출력(3.6장). 연결 요소 단위로 spoke 정점들을 출력할 때 특정 머신에 출력 작업이 몰리지 않도록 여러 대의 클러스터 머신에 출력 작업이 적절히 분배될 수 있도록 한다.

기본적인 분산 SlashBurn 알고리즘의 목적은 단일 머신 SlashBurn 알고리즘과 동일하다. 따라서 알고리즘 수행에 따른 그래프 정점 재배치 과정에 대한 예시는 그림 2를 참조하면 된다. 그림 7에 소개된 분산 SlashBurn 알고리즘의 세부 단계의 입출력과 각 세부 단계 사이의 관계는 그림 8의 순서도에서 볼 수 있다. 분산 SlashBurn 알고리즘은 인메모리(In-Memory) 분산 처리 프레임워크인 Apache Spark[6] 상에 구현되었으며 아래에서는 알고리즘의 각 단계별로 자세히 설명한다.

3.3 분산 SlashBurn 알고리즘 입출력

입력. 그래프의 간선 리스트를 입력으로 받는다. 입력 파일의 한 라인이 하나의 간선에 해당되며 한 간선은 두 개의 정점 식별자로 표현된다. 실험에서는 그래프 파일을 하둡(Hadoop) 분산 파일 시스템(HDFS)상에 올려두고 사용하였다.

출력. 단일 머신 기반 알고리즘은 정점 재배치 정보를 담은 배열을 반환하지만, 대용량 그래프 파일을 대상으로 하는 분산 SlashBurn 알고리즘은 배열을 직접 반환하지 않고, HDFS 상에 재배치 정보를 담은 파일을 출력한다. 출력 파일은 '시작인덱스_끝인덱스' 형태의 파일명을 사용하고 정점 재배치 후 시작인덱스부터 끝인덱스까지 매핑될 정점 식별자를 파일 내용으로 가지도록 하였다. 예를 들어, 출력 파일 '0_2'의 내용이 '6 5 9'라면 재배치 시 정점 6이 가장 앞에 오고 그 다음 정점 5와 9가 위치한다.

3.4 Hub와 Spoke 정점 분산 재배치

모든 정점들의 재배치를 하나의 머신에서 처리하는 기존 SlashBurn 알고리즘과 달리 분산 SlashBurn 알고리즘에서는 클러스터의 각 머신이 모든 정점에 대한 정보를 가지고 있지 않다. 따라서 알고리즘의 효율성과 확장성에 대한 고려와 함께 알고리즘의 정점 재배치 결과의 정확성이 고려되어야 한다.

분산 SlashBurn의 라운드가 진행되면서 출력되는 정점들은 다음의 세 종류로 나뉜다. 아래에서는 그래프의 가장 작은 정점 식별자가 0이라고 가정하고 각 정점 종류 별로 재배치 구간을 정리한다.

Hub 정점 (단계 1-1). K-hub 집합은 가장 큰 차수를 가진 k개의 정점들이다. 두 개의 정점(정점 1, 정점 2)으로 표시되는 각 간선을 (정점 1, 1), (정점 2, 1)로 매핑한 후, 정점 식별자를 기준으로 리듀스 연산을 수행하면서 해당 식별자에 매핑된 모든 1을 더해주면 각 정점의 차수를 구할 수 있다. k-hub 집합은 비어 있는 재배치 구간의 가장 앞부분을 차지하므로 k-hub 집합에 속한 정점들은 재배치 범위는 k와 현재 반복 횟수인 i ($i \geq 1$)에 의해 결정되어 다음과 같다: $[k \times (i-1), k \times i - 1]$.

단일 정점 Spoke (단계 1-2). k-hub 집합을 파악한 후 간선 리스트인 E로부터 k-hub 집합과 인접한 간선들을 제거하여 새로운 간선 리스트인 E'를 생성한다. 각 간선의 양 끝 정점이 k-hub 집합에 속하는 간선이 인접 간선에 해당된다. 이 때 특정 정점들의 인접 간선들이 모

두 k-hub 집합과 인접한 경우에 인접 간선 제거와 함께 해당 정점들이 삭제된다. 달리 말하면 이 정점들은 k-hub 집합과 인접한 간선들을 제거한 후에 다른 어떤 정점들과도 연결되어 있지 않게 되는 단일 정점으로 이루어진 spoke들이다. 단일 정점 spoke는 k-hub 집합의 인접 간선 리스트에 속한 정점들 중 인접 간선 리스트만을 고려했을 때의 정점 차수와 전체 그래프에서의 정점 차수가 동일한 정점들에 해당된다. 예를 들어, 그림 2(b)의 16번 정점은 그림 2(a)에서 hub 정점의 인접 간선 중 하나에 연결되어 있는데, 이 때 그래프 전체에서의 정점 차수가 1이고, hub 정점의 인접 간선 만을 고려했을 때의 정점 차수 또한 1로 동일하므로 16번 정점은 단일 정점 spoke에 해당된다. 반면 13번 정점의 경우에는 두 가지 정점 차수가 각각 2와 1로 서로 다르다. 이러한 단일 정점 spoke들은 따로 기록해두어 정점 재배치 결과로부터 누락되지 않도록 한다. 정점 재배치 순서상 단일 정점으로 구성된 spoke들은 단계 2에서 파악하는 크기가 2 이상인 복수 정점 spoke들보다 뒤에 위치한다. 단일 정점 spoke의 재배치 범위는 다음과 같다:

2. 크기가 2 이상인 복수 정점 spoke들이 존재하는 경우: 복수 정점 spoke들의 재배치 범위 끝을 *sEnd*, 단일 정점 spoke들의 개수를 c라 할 때, $[sEnd + 1,\ sEnd + c]$.
3. 크기가 2 이상인 복수 정점 spoke들이 존재하지 않는 경우: 현재 라운드까지 출력된 k-hub 집합의 크기를 h, gcc에 속한 정점의 개수를 g, 단일 정점 spoke들의 개수를 c라 할 때, $[h + g,\ h + g + c - 1]$.

복수 정점 Spoke (단계 2). SlashBurn 알고리즘의 단계 2에서는 k-hub 집합의 인접 간선들이 제거된 새로운 간선 리스트 E'로부터 연결 요소들을 찾는다. 거대 연결 요소를 제외한 나머지 연결 요소에 속한 정점들은 연결 요소 단위로 재배치되며 큰 연결 요소에 속한 정점들이 작은 연결 요소에 속한 정점들보다 재배치 범위에서 앞쪽 구간에 위치한다. m번째 연결 요소의 재배치 범위는, 현재 라운드까지 출력된 k-hub 집합의 크기를 h, gcc에 속한 정점의 개수를 g, 앞서 $m-1$번째까지 출력된 복수 정점 spoke들의 크기의 합을 s, m번째 연결 요소의 크기를 c라고 할 때 다음과 같다: $[h + g + s + 1,\ h + g + s + c]$.

3.5 분산 연결 요소 알고리즘

SlashBurn의 단계 2는 hub 정점들과 그에 인접한 간선들을 제거한 후 남은 그래프에서 (1) 연결 요소들을 찾고(단계 2-1), 이 중 가장 큰 연결 요소(거대 연결 요소)를 나머지 비 거대 연결 요소들로부터 분리하는 작업(단계 2-2)과 (2) 비 거대 연결 요소들에 속한 spoke 정점들을 연결 요소 단위로 재배치하는 작업(단계 2-3~2-5)으로 구성된다. 단계 2의 수행 시간 중 많은 부분은 연결 요소를 찾는 작업에 사용되는데, SlashBurn은 여러 라운드에 걸쳐 반복 수행되므로 단일 머신 기반의 연결 요소 알고리즘은 전체 알고리즘에서 병목이 되어 확장성을 떨어뜨린다. 따라서 단계 2의 효율적인 분산 처리를 위해서는 대용량 그래프의 연결 요소를 효율적으로 찾기 위한 분산 연결 요소 알고리즘이 필수적이다. 이를 해결하기 위해서 본 연구에서는 맵리듀스 기반의 최신 분산 연결 요소 알고리즘인 Alternating 알고리즘[4]을 사용하였다. Alternating 알고리즘의 동작 방식은 앞서 2.2장에서 설명하였다. 4.3장과 4.4장의 대용량 그래프들에 대한 실험 결과는 Alternating 알고리즘을 활용하여 연결 요소를 찾는 과정이 확장성 있게 동작함을 보여준다.

3.6 정점 재배치 결과의 분산 출력

SlashBurn의 단계 2를 효율적으로 분산 처리하기 위해 두 번째로 필요한 것은 spoke 정점들을 출력하는 과정을 분산으로 처리하는 것이다. Spoke 정점들을 출력하는 작업을 Spark의 드라이버 머신 한 대에서 담당하도록 하면 비 거대 연결 요소들의 크기가 큰 대용량 그래프의 경우, 실행 머신들에서 spoke 정점 정보를 드라이버 머신으로 보내기 위한 네트워크 통신 I/O와 정점 정보를 기록하기 위한 디스크 I/O 모두 머신 한대를 통해 이루어지게 되어 병목현상이 발생할 수 있다. 또한 spoke 정점들은 소속된 연결 요소 단위로 출력되어야 하는데, 하나의 연결 요소를 하나의 머신이 출력하도록 고정하면 연결 요소들의 크기에 불균형이 있는 경우 가장 큰 연결 요소를 처리하는 머신이 병목이 된다. 분산 SlashBurn 알고리즘에서는 클러스터 각 실행 머신 간의 로드 밸런싱을 고려한 분산 spoke 출력 프로세스를 통해 정점 재배치 결과를 보다 효율적으로 출력할 수 있게 하였다.

복수 정점 Spoke 분산 출력 프로세스. 여러 대의 실행 머신에서 spoke 출력을 병렬적으로 나누어하기 위한 프로세스는 다음의 과정으로 이루어진다.

(1) 연결 요소별 크기 계산

Alternating 알고리즘은 정점과 정점이 속한 연결 요소 식별자로 구성된 키-값 쌍의 리스트를 반환한다. 이 리스트에 다음의 맵리듀스 연산을 적용하여 연결 요소별 크기를 구한다. (가) 리스트의 각 쌍의 키와 값의 위치를 바꾼 후 모든 쌍의 값을 1로 매핑한다. (나) 키(=연결 요소 식별자)를 기준으로 리듀스 연산을 수행하면서 해당 키에 매핑된 모든 1들을 더하는데, 합계가 해당 연결 요소의 크기이다. 예를 들어, 입력 [(30, 20), (20, 20), (10, 10)]에 위의 (가)를 적용하면 [(20, 1), (20, 1), (10, 1)]이 되고 (나)를 적용하면 [(20, 1+1), (10, 1)]이 되어 연결 요소 10과 20의 크기를 구할 수 있다.

(2) 연결 요소 분할 개수/병합 키, 정점 재배치 구간 계산

전체 spoke 정점들을 클러스터에 적절한 크기로 분산하여 각 실행 머신에 걸리는 로드를 고르게 배분하기 위한 기본 아이디어는 큰 연결 요소에 속하는 spoke 정점들은 여러 대의 실행 머신으로 나누어 출력하고, 크기가 작은 연결 요소들에 속한 spoke 정점들의 경우에는 한 대의 실행 머신으로 모아 출력하는 것이다.

SlashBurn 알고리즘에서 가장 크기가 큰 연결 요소를 제외한 나머지 연결 요소들은 정점 재배치 구간에서 연결 요소의 크기만큼의 구간을 할당받고, 크기가 큰 연결 요소들은 크기가 작은 연결 요소보다 앞쪽 구간에 위치한다. 따라서 분할/병합 과정을 통해 정점들을 실행 머신으로 분배하는 과정에서 연결 요소들 간의 재배치 구간이 뒤섞이지 않도록 주의해야 한다.

연결 요소를 분할할지, 다른 연결 요소와 병합할지는 기준이 되는 값인 분할 사이즈에 의해서 결정한다. 연결 요소가 분할 사이즈 보다 큰 경우에는 해당 연결 요소를 ⌈연결 요소 크기 / 분할 사이즈⌉개로 분할해준다. 만약 분할 사이즈가 연결 요소보다 더 큰 경우에는 연결 요소들의 크기의 합이 분할 사이즈보다 크지 않을 때까지 연결 요소들을 병합한다.

표 1은 아홉 개의 연결 요소들이 존재하는 상황에서 분할 사이즈가 100일 때의 각 연결 요소별 분할 개수/병합키와 정점 재배치 구간을 보여준다. 표 1의 연결 요소 1은 거대 연결 요소 이므로 spoke 출력과는 무관하다. 연결 요소 2는 분할 사이즈보다 크기가 크므로 10조각으로 분할된다. 연결 요소 4, 5, 6의 경우. 연결 요소 4, 5의 크기의 합이 100 미만인데 비해 여기에 연결 요소 6을 병합할 경우 분할 사이즈보다 더 커지므로 연결 요소 4, 5만을 병합한다. 나머지 연결 요소 6, 7, 8 또한 동일한 과정을 통해 병합된다. 병합한 경우에도 정점 재배치 구간은 연결 요소 별로 따로 유지되어야 한다.

그래프의 전체 간선 수와 비교할 때, hub 정점들과 그에 인접한 간선들을 제거한 후의 그래프를 구성하는 연결 요소의 개수는 상대적으로 훨씬 적다. 따라서 대용량 그래프인 경우에도 과정 (2)의 연산은 단일 머신에서 무리 없이 효율적으로 처리 가능하다. 드라이버 머신은 과정 (1)에서 계산한 연결 요소별 크기 정보를 취합한 후 앞서 설명한 방식대로 연결 요소의 분할 개수/병합 키와 식별자 구간을 계산하고 spoke 정점을 재분배하는 데 이 정보를 사용한다.

(3) Spoke 정점 재분배

각 spoke 정점이 속한 연결 요소의 분할 개수/병합키 정보로부터 다음과 같이 키를 다르게 매핑하여 정점을 재분배한다.

가. 정점이 속한 연결 요소가 분할되는 경우:

표 1 예시 연결 요소(CC)들 별 분할 개수/병합키와 정점 재배치 구간 예제. 분할 사이즈는 100이고, 출력된 hub 정점 개수는 3이다.

Table 1 Example of split size/aggregation key and rearranged vertex range for sample connected components (Split size is set at 100. The number of outputted hub vertices is 3)

CC Identifier	CC Size	NumSplits/ Aggregation Key	Rearranged Vertex Range
1 (GCC)	10,000	-	3 ~ 10,002
2	1,000	10	10,003 ~ 11,002
3	100	1	11,003 ~ 11,102
4	50	-1	11,103 ~ 11,152
5	40	-1	11,153 ~ 11,192
6	30	-2	11,193 ~ 11,222
7	10	-2	11,223 ~ 11,232
8	2	-2	11,233 ~ 11,234
9	2	-2	11,235 ~ 11,236

- 분할 식별자 = 정점 식별자 % 분할 개수
- 키 = (연결 요소 식별자, 분할 식별자)

나. 정점이 속한 연결 요소가 병합되는 경우:

- 키 = (병합키, 0)

키에 대한 값으로 (정점 식별자, 해당 연결 요소의 재배치 범위)를 넘겨준다.

Spark 프레임워크 상에서 동일한 키를 가진 데이터는 동일한 실행 머신에서 처리된다. 지정된 분할 사이즈보다 큰 연결 요소에 속하는 정점들의 경우, 연결 요소 식별자는 동일하지만 정점 마다 부여되는 분할 식별자에 의해 서로 다른 실행 머신으로 분산되어 처리된다. 예를 들어 표 1에서 정점 30, 55, 70이 연결 요소 2에 속한다고 할 때 이들에 대한 키는 각각 (2, 30%10), (2, 55%10), (2, 70%10) = (2, 0), (2, 5), (2, 0)이 되어 정점 55는 나머지 두 정점과 다른 실행 머신에서 처리된다. 반대로 크기가 작은 연결 요소에 속한 정점들의 경우 분할 식별자 없이 병합키만을 사용하므로 같은 병합키를 가진 정점들은 하나의 실행 머신에서 처리된다. 예를 들어 표 1에서 정점 80, 85가 각각 연결 요소 4와 5에 속한다고 할 때, 정점 80과 85에 대한 키는 모두 (-1, 0)이므로 두 정점은 동일한 실행 머신에서 처리된다.

(4) Spoke 정점 출력

앞의 재분배 과정에 따라 실행 머신에서 처리되는 spoke 정점들은 분할된 경우와 병합된 경우 두 가지로 나누어진다.

가. 분할된 경우:

각 머신의 모든 정점들이 하나의 연결 요소에 속하기 때문에 하나의 파일에 출력해주면 된다. 다른 실행 머신에서 동일한 연결 요소에 속한 정점들을 출력해줄 수

있기 때문에, 출력 파일은 '시작인덱스_끝인덱스_분할식별자' 형태의 파일명을 사용하여 여러 실행 머신에서 한 파일에 덮어쓰지 않도록 한다.

나. 병합된 경우:

한 머신에 전송된 정점들이 여러 개의 연결 요소에 속할 수 있기 때문에 연결 요소 단위로 구분해서 파일을 출력해야 한다. 정점들을 연결 요소 단위로 구분하기 위해서 재배치 범위에 따라 정점들을 묶어 주고, 재배치 범위 별로 별도의 파일을 생성하여 출력한다. 출력 파일은 '시작인덱스_끝인덱스' 형태의 파일명을 사용한다. 하나의 연결 요소가 여러 실행 머신에서 분할되어 처리되는 것이 아니므로 분할 식별자를 파일명에 포함하지 않아도 된다.

단일 정점 Spoke 분산 출력 프로세스. 분산 SlashBurn 알고리즘(그림 7)의 단계 1-2에서 파악된 단일 정점 spoke들은 단계 2에서 복수 정점 spoke들이 출력된 이후에 복수 정점 spoke 보다 뒤쪽 구간에 출력된다. 단일 정점 spoke들은 각각이 크기가 1인 연결 요소이므로 서로 간에 어떤 순서로 출력되어도 상관없다. 따라서 복수 정점 spoke와는 달리 단일 정점 spoke는 별도의 정점 재분배 과정 없이 단일 정점 spoke의 파티션별로 병렬적으로 바로 출력할 수 있다. 각 파티션에서는 '시작인덱스_끝인덱스_파티션인덱스' 형태의 파일명을 갖는 출력 파일에 파티션 안의 단일 정점 spoke들을 출력하여 여러 실행 머신에서 한 파일에 덮어쓰지 않게 한다. Hub 정점과 복수 정점 spoke들의 재배치 범위에 따른 단일 정점 spoke들의 시작 인덱스와 끝 인덱스에 대해서는 3.4장에서 정리하였다.

4. 실 험

이 장에서는 다양한 크기의 실세계 그래프와 가상 그래프에 대한 실험을 통해 다음 두 가지 질문에 대한 답변을 알아보고자 한다:

- 질문 1. 실행 시간(4.3장): 대용량 실세계 그래프에 대해 분산 SlashBurn은 어느 정도의 수행 시간이 필요하며 단일 머신 SlashBurn보다 얼마나 빠르게 동작하는가?
- 질문 2. 확장성(4.4장): 분산 SlashBurn은 그래프의 크기가 커짐에 따라 확장성 있게 동작하는가? 또한 머신 수의 변화에 따라 분산 SlashBurn의 성능은 어떻게 달라지는가?

4.1 실험 환경

분산 SlashBurn과 단일 버전 SlashBurn의 성능을 비교하기 위해 다음과 같은 환경에서 실험을 수행하였다.

- 분산 SlashBurn: Apache Spark 상에서 구현된 코드를 마스터(master) 머신 1대와 수행(executor) 머신 16대로 구성된 클러스터 상에서 실행하였다. 클러스터 각 머신의 사양은 쿼드 코어 3.5GHz CPU, 32GB RAM으로 동일하다. 마스터 머신에서는 16GB RAM을, 16대의 실행(executor) 머신에서는 각각 24GB RAM을 할당하였다. 분할 사이즈(splitSize)는 10,000으로 설정하였다.
- 단일 머신 SlashBurn: 단일 머신 SlashBurn 알고리즘은 MATLAB으로 구현되어 있다[5]. 코드는 분산 클러스터의 마스터 머신에서 실행되었으며, 실험에는 MATLAB R2015b 64bit 버전을 사용하였다.

4.2 실험 데이터

제안된 알고리즘의 성능 평가를 위해 다양한 크기의 실세계 그래프들과 합성 그래프인 R-MAT 그래프[7]들을 사용하였다. 각 그래프들의 정점과 간선 수는 표 2에서 볼 수 있다. 실험에 사용된 실세계 그래프들에 대한 간략한 설명은 다음과 같다.

- Amazon[1]: Amazon에서 함께 빈번하게 구매된 상품들의 네트워크 데이터. 그래프의 정점은 상품에 해당하고 빈번하게 함께 구매된 상품들에 해당되는 정점 사이에는 간선이 존재한다.
- Web-BS[2]: 2002년도 University of California, Berkeley와 Stanford University의 웹 페이지들과 웹 페이지 간의 하이퍼링크로 구성된 그래프 데이터이다.
- LiveJournal-Links[3]: 온라인 소셜 사이트인 LiveJournal의 사용자들 사이의 연결 관계에 대한 그래프이다.
- LiveJournal-Membership[4]: 온라인 소셜 사이트인 LiveJournal의 사용자들과 그룹 사이의 멤버십 네트워크로 이분(bipartite) 그래프이다.
- Orkut[5]: Orkut 사용자들과 그룹 사이의 멤버십 네트워크로 이분 그래프이다.
- Friendster[6]: 온라인 소셜 사이트인 Friendster의 친구 관계 네트워크 데이터이다.

R-MAT 그래프: Friendster와 같은 실세계 그래프들은 공동체 구조를 보이며, 그래프의 지름이 작고, 멱법칙 차수 분포를 따르는 등의 특징을 가진다[7]. R-MAT은 재귀적이고 랜덤한 방식으로 이러한 실세계 그래프들의 특징을 지니는 그래프를 생성하기 위한 모델이다.

R-MAT으로 그래프를 생성하는 과정은 다음과 같다[7]: 1) 그래프의 인접 행렬을 동일한 크기의 사 사분면으로 나누고, 각 사분면에 할당된 확률 a, b, c, d (a+b+c+d=1)에 따라 하나의 사분면을 고른다. 2) 선택된

1) http://snap.stanford.edu/data/com-Amazon.html
2) http://snap.stanford.edu/data/web-BerkStan.html
3) http://konect.uni-koblenz.de/networks/livejournal-links
4) http://konect.uni-koblenz.de/networks/livejournal-group memberships
5) http://konect.uni-koblenz.de/networks/orkut-groupmemberships
6) https://snap.stanford.edu/data/com-Friendster.html

표 2 실험에 사용된 그래프들의 크기
Table 2 Size of the graphs used for the experiments

Graph	# Vertices	# Edges
Amazon	334,863	925,872
Web-BS	685,230	7,600,595
LiveJournal-Links	5,204,176	49,174,620
LiveJournal-Membership	10,690,276	112,307,385
Orkut	11,514,053	327,037,487
Friendster	65,608,366	1,806,067,135
R-MAT 16	65,536	4,194,304
R-MAT 17	131,072	8,388,608
R-MAT 18	262,144	16,777,216
R-MAT 19	524,288	33,554,432
R-MAT 20	1,048,576	67,108,864
R-MAT 21	2,097,152	134,217,728
R-MAT 22	4,194,304	268,435,456
R-MAT 23	8,388,608	536,870,912
R-MAT 24	16,777,216	1,073,741,824
R-MAT 25	33,554,432	2,147,483,648

사분면을 동일한 과정을 따라 다시 네 개의 사분면으로 나누고 이 과정을 인접 행렬의 하나의 셀에 도착할 때까지 반복한다. 3) 이 셀에 간선을 할당하고 1부터 다시 반복한다. 단일 머신의 처리 용량을 초과하는 대규모 R-MAT 그래프는 TeGViz[8]와 같은 분산 그래프 생성 툴을 사용하여 생성할 수 있다.

본 논문의 실험에서는 각 사분면 확률인 (a, b, c, d)를 (0.57, 0.19, 0.19, 0.05)로 설정한 후 2^{22}, …, 2^{31}개의 간선을 가진 10개의 R-MAT 그래프들을 생성하여 사용하였다.

4.3 수행 시간

실세계 그래프, 특히 대용량 그래프에 대한 분산 SlashBurn 알고리즘과 단일 머신 SlashBurn 알고리즘의 성능을 비교해보기 위해 여섯 가지 서로 다른 크기의 그래프들에 대해 각 알고리즘의 수행 시간을 측정하였다. SlashBurn 알고리즘의 수행 시간은 매 라운드마다 선택하는 hub 정점의 개수(=전체 정점 개수 × hub 정점 선택 비율 R)에 따라 달라지는데, 크기가 큰 그래프들에 대해서는 크기가 작은 그래프들보다 큰 값의 R을 사용하여 조금 더 빨리 수렴할 수 있도록 하였다. 세 가지 서로 다른 R 값을 다음 그래프 들에 적용하였다:

- R=0.05: Amazon, Web-BS
- R=0.01: LiveJournal-Links, LiveJournal-Membership, Orkut
- R=0.05: Friendster

그림 9는 그래프 별 각 알고리즘의 수행 시간을 그래프 크기가 작은 순에서 큰 순으로 보여준다. 그림 9에서 상대적으로 크기가 작은 Amazon과 Web-BS 그래프들에 대해서는 단일 머신 SlashBurn 알고리즘이 분산 SlashBurn 알고리즘에 비해서 각각 7.5배, 2.4배 가량 빠르게 동작하는 것을 볼 수 있다. 크기가 작은 그래프의 경우, 연산에 필요한 데이터들을 메모리에 무리없이 적재하여 처리할 수 있는 반면, 분산 처리를 수행하는 데에는 네트워크 입출력 시간이나 분산 프레임워크 수행에 따른 오버헤드 등이 존재하므로 이는 예상 가능한 결과이다.

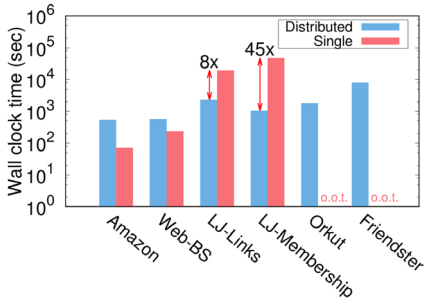


그림 9 실세계 그래프들에 대한 단일 머신 SlashBurn과 분산 SlashBurn 알고리즘간의 수행 시간 비교. 12시간 안에 종료되지 않은 실험은 o.o.t.로 표시한다.
Fig. 9 Comparison of running time between single machine SlashBurn and distributed SlashBurn algorithms for real-world graphs. Experiments not terminated within 12 hours are marked as o.o.t

한편 그래프 크기가 커질수록 두 알고리즘의 수행 시간 차이는 줄어들다가 LiveJournal-Links 그래프부터는 수행 시간이 역전되는 것을 볼 수 있다. LiveJournal-Membership 그래프의 경우 분산 SlashBurn이 단일 머신 SlashBurn 보다 45배 가량 빠르다. 한편 단일 머신 SlashBurn 알고리즘으로는 제한 시간인 12시간 안에 Orkut와 Friendster 그래프를 처리할 수 없었다. 반면에 분산 SlashBurn의 경우에는 이 두 가지 그래프에 대해서도 무리 없이 동작하였으며 수행 시간이 그래프 크기에 어느 정도 비례하여 늘어나는 것을 볼 수 있다.

단일 머신 SlashBurn이 처리할 수 있었던 가장 큰 데이터인 LiveJournal-Membership 데이터 결과에서 사용된 머신 개수(16대)에 비해 처리 속도의 향상폭(45배)이 훨씬 더 큰 것은 다음의 두 가지 이유가 있다. 첫 번째, 분산 컴퓨팅 프레임워크는 단일 머신 프로그램에 비해 효율적으로 대용량 데이터를 처리할 수 있다. 기본적으로 분산 프레임워크 상에서 프로그램 연산은 클러스터의 각 머신으로 분산되어 병렬 수행되므로 단일 머신에서만 수행되는 것에 비해 수행 시간은 줄어들게 된다. 나아가 내부적으로 정렬 연산과 같이 O(n) 이상의 시간 복잡도를 가지는 알고리즘들을 사용하는 분산 SlashBurn과 같

은 프로그램에서 데이터 크기가 1/n이 되면 수행 시간은 1/n 이상으로 줄어들므로 잘게 나누어진 데이터에 대한 연산을 여러 머신에서 동시에 수행하게 하면 보다 빠르게 데이터를 처리할 수 있다. 두 번째, 데이터가 일정 크기 이상 증가하면 단일 머신 알고리즘의 성능은 하락하게 된다. 데이터 크기가 작을 때는 모든 데이터를 메모리에 적재해서 효율적으로 처리할 수 있지만 데이터 크기가 커져 이것이 불가능한 상황이 되면 필요한 부분을 디스크로부터 메모리로 읽어 들이고 연산 수행 후 다시 디스크로 쓰는 과정을 반복 수행해야 하므로 프로그램 성능이 떨어지게 된다. 최악의 경우에는 메모리 부족 문제로 수행이 불가능하게 될 수도 있다.

4.4 확장성

분산 알고리즘의 확장성은 크게 데이터 크기의 증가에 따른 확장성과 머신 수의 증가에 따른 확장성의 두 가지 측면으로 나누어 볼 수 있다. 이 장에서는 이 두 가지 측면에서 분산 SlashBurn의 확장성에 대해 살펴본다.

그래프 크기 증가에 따른 확장성. R-MAT 16부터 R-MAT 25까지 서로 다른 크기의 R-MAT 그래프들을 사용하여 그래프의 크기 증가에 따른 확장성을 살펴보았다. Hub 정점 선택 비율 R은 모든 R-MAT 그래프에 대해 동일하게 0.05로 설정하였다. 그림 10의 실험 결과로부터 그래프의 간선 수가 증가할 때 분산 SlashBurn의 수행 시간이 거의 선형적으로 증가하는 것을 볼 수 있다. 반면에 단일 머신 SlashBurn의 경우 훨씬 가파르게 수행 시간이 증가하여 R-MAT 23 그래프부터는 제한 시간인 12시간 안에 처리할 수 없었다. 분산 SlashBurn은 단일 머신 SlashBurn보다 8배 큰 R-MAT 그래프를 문제없이 처리하였고, o.o.t.가 발생하기 직전의

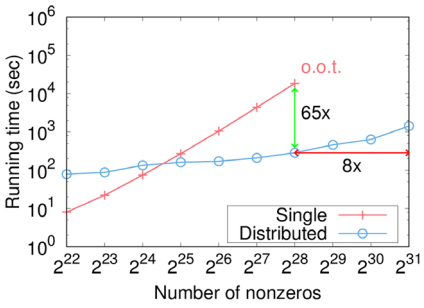


그림 10 R-MAT 그래프에서 인접 행렬의 0이 아닌 값의 개수에 따른 분산 SlashBurn의 확장성. 12시간 안에 종료되지 않은 실험은 o.o.t.로 표시한다.

Fig. 10 The scalability of distributed SlashBurn according to the number of non-zeros on R-MAT graphs. Experiments not terminated within 12 hours are marked as o.o.t.

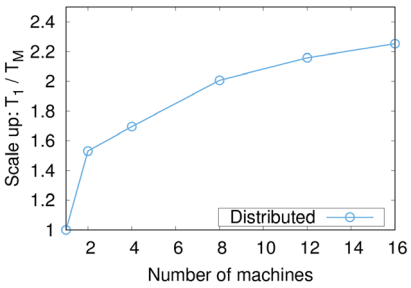


그림 11 LiveJournal-Membership 그래프에서 머신 수에 따른 분산 SlashBurn의 확장성. TM은 M개의 머신 사용시 걸린 시간을 의미한다.

Fig. 11 The scalability of distributed SlashBurn according to the number of machines on LiveJournal-Membership graph. TM indicates the running time taken with M machines

그래프인 R-MAT 22에 대해서는 단일 머신 SlashBurn보다 65배 가량 빠르게 동작하였다.

머신 수에 따른 확장성. LiveJournal-Membership 그래프에 대해 머신 수를 1대에서 16대까지 증가시키면서 분산 SlashBurn의 실행 시간이 어떻게 변화하는지를 측정하였다. 그림 11에서 y축은 M개의 머신을 사용할 때의 분산 SlashBurn의 성능이 한 개의 머신 만을 사용할 때에 비해 얼마나 개선되는지를 의미한다. 한 대에서 두 대의 머신으로 증가할 때의 성능 상승 폭이 가장 크고, 그 이후부터는 거의 선형적인 성능 확장이 있음을 볼 수 있다.

5. 관련 연구

5.1 실세계 그래프의 특성

실세계 그래프 구조에 대한 연구가 계속 되어 오면서, 실세계 그래프만의 몇 가지 특성이 알려졌다. 대표적인 실세계 그래프의 특성으로는 간선 분포가 멱법칙[9]을 따른다는 것이 있다. 이런 특성은 임의로 생성한 그래프에서는 나타나지 않기 때문에, 그래프의 공격 내성(attack tolerance) 특성[10]과 함께 그래프 조각화(graph scattering) 연구[11] 등 다양한 연구에 적용되어 왔다. 그러나 이러한 특성을 그래프 재배치나 그래프 압축에 활용한 연구는 SlashBurn[2]이 처음이다.

5.2 그래프 분할 및 압축

최근 그래프 데이터를 처리하기 위해 다양한 병렬 또는 분산 처리 기법이 도입됨에 따라, 그래프 분할의 중요도가 높아졌다. PageRank, 연결 요소, 그래프 내 최단거리 탐색, BFS 등 많은 그래프 알고리즘들이 행렬-벡터 곱셈으로 표현될 수 있기 때문에, 그래프를 표현하는 행렬과 벡터를 균등하게 분할하면 그래프 알고리즘

의 성능을 끌어 올릴 수 있다. PEGASUS[3]와 같은 그래프 연산 프레임워크가 이러한 행렬-벡터 곱셈을 빠르게 수행하기 위해 부분 행렬로 분할해 수행하는 방법을 사용했다. METIS[12], co-clustering[13], spectral clustering[14], shingle-ordering[15] 등 그래프 분할을 하기 위한 다양한 방법이 제안되었으나, 이러한 방법들이 실세계 그래프에는 잘 적용되지 않는다는 것[16]이 알려졌다. 실세계 그래프에 대한 그래프 분할 성능을 올리기 위해 간선 분포의 멱법칙 특성을 활용한 그래프 분할 방법[17]이 제안되었고, 실세계 그래프에 적용해 실험을 수행했을 때 기존 방법보다 더 균등하게 그래프 분할을 수행하는 것으로 나타났다.

대부분의 분산 그래프 알고리즘은 네트워크를 통한 데이터 전송이 병목 지점이다. 따라서 그래프를 표현하는 행렬을 압축해 네트워크 사용량을 줄이는 것이 좋은 성능 향상을 보이기 때문에 그래프 압축 역시 활발하게 연구되고 있는 분야 중 하나이다. 실세계 그래프 중 하나인 웹 그래프에 대해 지역성[18]을 활용해 압축을 수행하거나, 인접한 정점을 찾는 질의를 그래프 압축을 이용해 빠르게 수행[19]하는 등 다양한 방법과 응용이 제안되었다. SlashBurn[2]은 멱법칙 특성을 이용해 그래프 압축의 효율을 개선해 PEGASUS와 같은 그래프 연산 프레임워크의 성능을 높인다.

5.3 분산 처리 프레임워크

그래프 데이터의 용량이 점점 커짐에 따라 그래프 알고리즘 수행에도 분산 처리가 필요하게 되었다. 널리 쓰이고 있는 분산 처리 프로그래밍 프레임워크에는 MapReduce[20]가 있다. MapReduce는 오픈 소스 구현체인 Apache Hadoop[7]이 있다는 점과 map과 reduce만 구현하면 데이터 분배, 장애 복구, 스케줄링 등 다양한 분산 처리 이슈를 시스템이 자동으로 처리해주는 장점을 가진다. 하지만 중간 데이터를 하드 디스크에 저장하는 것에 따른 성능 하락이 있고 그래프 알고리즘과 같은 반복 작업을 수행함에 있어서 비효율적[21]이며 map과 reduce로 모든 알고리즘을 구성해야 한다는 단점을 지닌다.

최근에는 MapReduce의 단점을 보완하고 RDD(Resilient Distributed Datasets)의 개념을 도입한 Apache Spark[6]의 사용이 확산되고 있다. Apache Spark는 중간 결과물을 분산 메모리에 저장해 반복 작업이 많은 그래프 알고리즘이나 기계 학습 알고리즘에도 좋은 성능을 보이고, map과 reduce 이외에도 join, group by 등 다양한 연산을 지원한다. 분산 SlashBurn 알고리즘 역시 반복 작업이 많은 알고리즘이어서 Apache Spark을 통해 성능을 향상시킬 수 있었다.

7) http://hadoop.apache.org

6. 결론 및 향후 연구

본 논문에서는 수천만 개의 정점과 수십억 개의 간선들로 구성된 대용량 그래프에서의 정점 재배치를 위한 분산 SlashBurn 알고리즘을 제안하였다. 분산 SlashBurn은 효율적인 그래프 압축과 마이닝을 위한 대규모의 정점 재배치 프로세스를 분산 처리 함으로써 대용량 그래프를 기존의 단일 머신 SlashBurn보다 훨씬 빠르게 처리하며 확장성 있게 동작한다. 대용량 그래프에 대해 제한된 시간 동안 진행된 실험에서 분산 SlashBurn은 단일 머신 SlashBurn보다 45배 이상 빠르게 동작하였고, 16배 이상 큰 그래프를 처리하였다. 클러스터의 크기를 실험에 사용된 것 이상으로 확장함으로써 분산 SlashBurn은 보다 큰 크기의 그래프 또한 처리할 수 있을 것으로 기대된다.

분산 SlashBurn 알고리즘의 수행 시간 중 많은 부분은 반복적으로 그래프의 연결 요소를 찾는 데 사용된다. 분산 SlashBurn의 성능을 향상시키기 위한 후속 연구로 분산 SlashBurn에 사용되는 분산 연결 요소 알고리즘의 성능을 개선시키기 위한 연구를 향후 진행할 계획이다.

References

[1] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos, "Fully Automatic Cross-associations," *Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) 2004*, pp. 79-88, 2004.

[2] Y. Lim, U Kang, and C. Faloutsos, "SlashBurn: Graph Compression and Mining beyond Caveman Communities," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol. 26, No. 12, pp. 3077-3089, Apr. 2014.

[3] U Kang, C. E. Tsourakakis, and C. Faloutsos, "PEGASUS: A Peta-Scale Graph Mining System," *Proc. of the 9th IEEE International Conference on Data Mining (ICDM) 2009*, pp. 229-238, 2009.

[4] R. Kiveris, S. Lattanzi, V. Mirrokni, V. Rastogi, and S. Vassilvitskii, "Connected Components in MapReduce and Beyond," *Proc. of the ACM Symposium on Cloud Computing (SOCC) 2014*, 2014.

[5] U Kang, SlashBurn implementation in MATLAB [Online]. Available: http://datalab.snu.ac.kr/~ukang/SlashBurn-1.0.zip (downloaded 2016, May 1)

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, "Spark: Cluster Computing with Working Sets," *Proc. of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)*, 2010.

[7] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," *Proc. of the*

4th SIAM International Conference on Data Mining (SDM) 2004, pp. 442-446, 2004.
[8] B. Jeon, I. Jeon, and U Kang, "TeGViz: Distributed Tera-Scale Graph Generation and Visualization," *Proc. of the IEEE International Conference on Data Mining Workshop (ICDMW) 2015*, pp. 1620-1623, 2015.
[9] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *Proc. of the Conference on Appl., Technol., Archit., and Protocols for Comput. Commun. 1999*, pp. 251-262, 1999.
[10] R. Albert, H. Jeong, and A.-L. Barabasi, "Error and attack tolerance of complex networks," *Nature*, Vol. 406, No. 6794, pp. 378-382, 2000.
[11] A. P. Appel, D. Chakrabarti, C. Faloutsos, R. Kumar, J. Leskovec, and A. Tomkins, "Shatterplots: Fast tools for mining large graphs," *Proc. of the SIAM International Conference on Data Mining (SDM) 2009*, pp. 802-813, 2009.
[12] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," *Proc. of the 36th Annual ACM/IEEE Design Automation Conference (DAC) 1999*, pp. 343-348, 1999.
[13] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDM) 2003*, pp. 89-98, 2003.
[14] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, Vol. 17, No. 4, pp. 395-416, 2007.
[15] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDM) 2009*, pp. 219-228, 2009.
[16] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," *Proc. of the 17th International Conference on World Wide Web (WWW) 2008*, pp. 695-704, 2008.
[17] Y. Lim, W. J. Lee, H. J. Choi and U Kang, "Discovering large subsets with high quality partitions in real world graphs," *Proc. of the 2015 International Conference on Big Data and Smart Computing (BIGCOMP)*, pp. 186-193, 2015.
[18] P. Boldi and S. Vigna, "The webgraph framework I: compression techniques," *Proc. of the 13th International Conference on World Wide Web (WWW) 2008*, pp. 595-602, 2004.
[19] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD) 2012*, pp. 157-168, 2012.
[20] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Proc. of the 6th Symposium on Operating System Design and Implementation (OSDI) 2004*, pp. 137-150, 2004.
[21] V. Kalavri and V. Vlassov, "MapReduce: Limitations, Optimizations and Open Issues," *Proc. of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) 2013*, pp. 1031-1038, 2013.

박 남 용
2010년 서울대학교 컴퓨터공학부 졸업(학사)
2013년 서울대학교 컴퓨터공학부 졸업(석사)
2015년~현재 서울대학교 연구원

박 치 완
2016년 연세대학교 지구시스템 과학과, 컴퓨터과학과 졸업(학사). 2016년~현재 서울대학교 컴퓨터 공학부(석사과정)

강 유
정보과학회논문지
제 43 권 제 8 호 참조