(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2021년11월11일
(11) 등록번호 10-2325047
(24) 등록일자 2021년11월05일

(51) 국제특허분류(Int. Cl.)
*G06F 16/901* (2019.01)
(52) CPC특허분류
*G06F 16/9024* (2019.01)
(21) 출원번호 10-2019-0067897
(22) 출원일자 2019년06월10일
심사청구일자 2019년06월10일
(65) 공개번호 10-2020-0141208
(43) 공개일자 2020년12월18일
(56) 선행기술조사문헌
JP2017509043 A*
*(뒷면에 계속)*
전체 청구항 수 : 총 15 항

(73) 특허권자
**포항공과대학교 산학협력단**
경상북도 포항시 남구 청암로 77 (지곡동)
(72) 발명자
**한욱신**
대전광역시 서구 둔산로 155, 106동 508호(둔산동, 크로바아파트)
**고성윤**
경상북도 포항시 남구 청암로 77, 포스빌 2동 205호(지곡동, 포항공과대학교)
(74) 대리인
**김태헌**

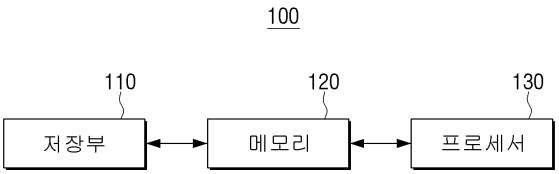
심사관 : 김경완

(54) 발명의 명칭 **그래프 데이터 처리 방법 및 그래프 데이터 처리 장치**

## (57) 요 약

그래프 데이터 처리 장치 및 그래프 데이터 처리 방법이 개시된다. 그래프 데이터 처리 방법은 저장부에 저장된 복수의 정점과 엣지를 포함하는 그래프 데이터 중 메모리 요구량에 기초하여 기 설정된 개수의 소스 정점, 소스 정점에 기초한 엣지 리스트를 포함하는 서브 그래프 데이터를 로딩하는 단계, 각각의 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행하는 단계, 제1 도착 정점이 식별된 소스 정점에 기초하여 엣지 리스트를 로딩하는 단계, 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하는 제2 레벨 과정을 수행하는 단계 및 소스 정점, 제1 도착 정점 및 제2 도착 정점에 기초하여 쿼리를 처리하는 단계를 포함한다.

*대 표 도* - 도1



(56) 선행기술조사문헌
KR1020160014111 A*
KR1020160072305 A*
KR1020160093237 A*
US20170124452 A1*
*는 심사관에 의하여 인용된 문헌

공지예외적용 : 있음

# 명 세 서

## 청구범위

### 청구항 1

저장부에 저장된 복수의 정점과 엣지를 포함하는 그래프 데이터 중 메모리 요구량에 기초하여 기 설정된 개수의 소스 정점, 상기 소스 정점에 기초한 엣지 리스트를 포함하는 상기 메모리 요구량에 대응되는 서브 그래프 데이터를 로딩하는 단계;

상기 기 설정된 개수의 소스 정점이 식별되면, 상기 각각의 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행하는 단계;

상기 제1 레벨 과정이 수행되면, 상기 제1 도착 정점이 식별된 소스 정점에 기초하여 엣지 리스트를 로딩하여, 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하는 제2 레벨 과정을 수행하는 단계; 및

상기 소스 정점, 상기 제1 도착 정점 및 상기 제2 도착 정점에 기초하여 쿼리를 처리하는 단계;를 포함하며,

상기 쿼리를 처리하는 단계는,

상기 소스 정점, 상기 제1 도착 정점 및 상기 제2 도착 정점에 기초하여 3개 보다 많은 n개의 꼭지점을 가지는 다각형 연결 관계 쿼리 및 연결 관계 패턴 검색 쿼리를 처리하는 그래프 데이터 처리 방법.

### 청구항 2

제1항에 있어서,

제k-1 도착 정점이 식별된 소스 정점에 기초하여 엣지 리스트를 로딩하는 단계;

상기 제k-1 도착 정점이 식별된 소스 정점과 연결된 제k 도착 정점을 식별하는 제k 레벨 과정을 수행하는 단계;를 더 포함하는, 그래프 데이터 처리 방법.

여기서, $3 \leq k \leq n-1$ 인 자연수

### 청구항 3

제1항에 있어서,

상기 제1 레벨 과정을 수행하는 단계는,

상기 소스 정점과 연결된 기 설정된 개수 이내의 엣지를 포함하는 제1 윈도우를 설정하고, 상기 설정된 제1 윈도우를 순차적으로 슬라이딩시키며, 상기 제1 윈도우가 위치한 영역에 포함된 소스 정점 및 엣지 리스트에 기초하여 제1 도착 정점을 식별하는, 그래프 데이터 처리 방법.

### 청구항 4

제3항에 있어서,

상기 제2 레벨 과정을 수행하는 단계는,

상기 제1 도착 정점이 식별된 소스 정점을 기초로 상기 소스 정점과 연결된 엣지를 포함하는 제2 윈도우를 설정하고, 상기 설정된 제2 윈도우를 순차적으로 슬라이딩시키며, 상기 제2 윈도우가 위치한 영역에 포함된 상기 제1 도착 정점이 식별된 소스 정점 및 엣지 리스트에 기초하여 제2 도착 정점을 식별하는, 그래프 데이터 처리 방법.

### 청구항 5

제3항에 있어서,

상기 제2 레벨 과정을 수행하는 단계는,

현재 위치한 제2 윈도우의 영역 내의 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하고, 상기 제2 윈도우가 다음에 위치할 영역에 대응되는 상기 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 로딩하며, 통신 인터페이스가 이전에 위치했던 상기 제2 윈도우의 영역에서 식별된 제2 도착 정점에 대한 업데이트 정보를 전송함으로써 상기 저장부, 그래프 데이터 처리 장치의 프로세서 및 통신인터페이스가 병렬적으로 동시에 동작하는, 그래프 데이터 처리 방법.

**청구항 6**

제1항에 있어서,

상기 제1 레벨 과정을 수행하는 단계는,

상기 소스 정점과 상기 제1 도착 정점 각각에 부여된 번호에 기초하여 상기 소스 정점과 상기 제1 도착 정점을 각각 올림차순으로 배열하고,

상기 제2 레벨 과정을 수행하는 단계는,

상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점 각각에 부여된 번호에 기초하여 상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점을 각각 올림차순으로 배열하는, 그래프 데이터 처리 방법.

**청구항 7**

제6항에 있어서,

상기 제1 레벨 과정을 수행하는 단계는,

상기 소스 정점의 번호가 상기 제1 도착 정점의 번호보다 작은 경우 상기 소스 정점의 번호보다 큰 번호의 인접 소스 정점을 기초로 연결 관계를 식별하고, 상기 제1 도착 정점의 번호가 상기 소스 정점의 번호보다 작은 경우 상기 제1 도착 정점의 번호보다 큰 번호의 인접 제1 도착 정점을 기초로 연결 관계를 식별하며,

상기 제2 레벨 과정을 수행하는 단계는,

상기 제1 도착 정점이 식별된 소스 정점의 번호가 상기 제2 도착 정점의 번호보다 작은 경우 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 큰 번호의 인접 제1 도착 정점이 식별된 소스 정점을 기초로 연결 관계를 식별하고, 상기 제2 도착 정점의 번호가 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 작은 경우 상기 제2 도착 정점의 번호보다 큰 번호의 인접 제2 도착 정점을 기초로 연결 관계를 식별하는, 그래프 데이터 처리 방법.

**청구항 8**

복수의 정점과 엣지를 포함하는 그래프 데이터를 저장하는 저장부;

상기 그래프 데이터 중 메모리 요구량에 기초하여 상기 메모리 요구량에 대응되는 서브 그래프 데이터를 로딩하는 메모리; 및

상기 로딩된 서브 그래프 데이터에 기초하여 쿼리를 처리하는 프로세서;를 포함하고,

상기 메모리는,

상기 저장부에 저장된 복수의 정점과 엣지를 포함하는 그래프 데이터 중 상기 메모리 요구량에 기초하여 기 설정된 개수의 상기 서브 그래프 데이터의 소스 정점, 상기 소스 정점에 기초한 엣지 리스트를 로딩하고, 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 로딩하며,

상기 프로세서는,

상기 기 설정된 개수의 소스 정점이 식별되면, 상기 메모리에 로딩된 소스 정점에 기초한 엣지 리스트에 기초하여 상기 각각의 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행하고,

상기 제1 레벨 과정이 수행되면, 상기 메모리에 로딩된 상기 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트에 기초하여 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하는 제2 레벨 과정을 수행하며,

상기 소스 정점, 상기 제1 도착 정점 및 상기 제2 도착 정점에 기초하여 3개 보다 많은 n개의 꼭지점을 가지는

다각형 연결 관계 쿼리 및 연결 관계 패턴 검색쿼리를 처리하는, 그래프 데이터 처리 장치.

**청구항 9**

제8항에 있어서,

상기 메모리는,

제k-1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 로딩하며,

상기 프로세서는,

상기 메모리에 로딩된 상기 제k-1 도착 정점이 식별된 소스 정점과 연결된 제k 도착 정점을 식별하는 제k 레벨 과정을 수행하는, 그래프 데이터 처리 장치.

여기서, $3 \leq k \leq n - 1$ 인 자연수

**청구항 10**

제8항에 있어서,

상기 프로세서는,

상기 제1 레벨 과정에서, 상기 소스 정점과 연결된 기 설정된 개수 이내의 엣지를 포함하는 제1 윈도우를 설정하고, 상기 설정된 제1 윈도우를 순차적으로 슬라이딩시키며, 상기 제1 윈도우가 위치한 영역에 포함된 소스 정점 및 엣지 리스트에 기초하여 제1 도착 정점을 식별하는, 그래프 데이터 처리 장치.

**청구항 11**

제10항에 있어서,

상기 프로세서는,

상기 제2 레벨 과정에서, 상기 제1 도착 정점이 식별된 소스 정점을 기초로 상기 제1 도착 정점이 식별된 소스 정점과 연결된 엣지를 포함하는 제2 윈도우를 설정하고, 상기 설정된 제2 윈도우를 순차적으로 슬라이딩시키며, 상기 제2 윈도우가 위치한 영역에 포함된 상기 제1 도착 정점이 식별된 소스 정점 및 엣지 리스트에 기초하여 제2 도착 정점을 식별하는, 그래프 데이터 처리 장치.

**청구항 12**

제10항에 있어서,

통신 인터페이스;를 더 포함하고,

상기 프로세서는,

상기 제2 레벨 과정에서, 현재 위치한 제2 윈도우의 영역 내의 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하고, 상기 제2 윈도우가 다음에 위치할 영역에 대응되는 상기 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 상기 메모리에 로딩하도록 상기 저장부를 제어하며, 이전에 위치했던 상기 제2 윈도우의 영역에서 식별된 제2 도착 정점에 대한 업데이트 정보를 전송하도록 상기 통신 인터페이스를 제어함으로써 상기 저장부, 상기 프로세서 및 상기 통신 인터페이스를 병렬적으로 동시에 동작시키는, 그래프 데이터 처리 장치.

**청구항 13**

제8항에 있어서,

상기 프로세서는,

상기 제1 레벨 과정에서, 상기 소스 정점과 상기 제1 도착 정점 각각에 부여된 번호에 기초하여 상기 소스 정점과 상기 제1 도착 정점을 각각 올림차순으로 배열하고,

상기 제2 레벨 과정에서, 상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점 각각에 부여된 번호에 기초하여 상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점을 각각 올림차순으로 배열하는, 그래프

데이터 처리 장치.

**청구항 14**

제13항에 있어서,

상기 프로세서는,

상기 제1 레벨 과정에서, 상기 소스 정점의 번호가 상기 제1 도착 정점의 번호보다 작은 경우 상기 소스 정점의 번호보다 큰 번호의 인접 소스 정점을 기초로 연결 관계를 식별하고, 상기 제1 도착 정점의 번호가 상기 소스 정점의 번호보다 작은 경우 상기 제1 도착 정점의 번호보다 큰 번호의 인접 제1 도착 정점을 기초로 연결 관계를 식별하며,

상기 제2 레벨 과정에서, 상기 제1 도착 정점이 식별된 소스 정점의 번호가 상기 제2 도착 정점의 번호보다 작은 경우 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 큰 번호의 인접 상기 제1 도착 정점이 식별된 소스 정점을 기초로 연결 관계를 식별하고, 상기 제2 도착 정점의 번호가 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 작은 경우 상기 제2 도착 정점의 번호보다 큰 번호의 인접 제2 도착 정점을 기초로 연결 관계를 식별하는, 그래프 데이터 처리 장치.

**청구항 15**

제8항의 그래프 데이터 처리 장치를 복수 개 포함하고,

상기 복수 개 그래프 데이터 처리 장치 각각은,

소스 정점 및 연결된 엣지의 개수에 기초하여 라운드 로빈 방식으로 상기 소스 정점 및 상기 엣지를 포함하는 그래프 데이터를 분산하여 저장하는, 복수의 그래프 데이터 처리 장치를 포함하는 시스템.

## *발명의 설명*

### *기 술 분 야*

[0001] 본 개시는 그래프 데이터 처리 방법 및 그래프 데이터 처리 장치에 관한 것으로, 더욱 상세하게는 대규모 그래프를 분산 처리하는 그래프 데이터 처리 방법 및 그래프 데이터 처리 장치에 관한 것이다.

### *배 경 기 술*

[0002] 그래프 데이터의 크기가 빠르게 증가함에 따라 대규모의 그래프 데이터를 효율적으로 분석하기 위한 많은 연구가 진행되고 있다. 대규모 그래프 데이터를 분석하는 연구는 크게 두 가지 방법으로 구분될 수 있다.

[0003] 첫번째 방법은 분산 컴퓨팅 노드들의 메모리에 입력 그래프를 나누어 로딩할 수 있을 때까지, 컴퓨팅 노드의 수를 증가시키는 메인 메모리 기반의 처리 방식이다. 대표적인 시스템으로 GraphX, Pregel+, GRAPE, 그리고 Gemini가 있다. 하지만, 메인 메모리 기반의 처리 방식은 분산 처리를 위해 정점과 간선(엣지)의 중복을 중복하여 저장하고 중간 계산 결과를 유지하는 자료 구조의 크기가 폭발적으로 증가하기 때문에 그래프 처리에 요구되는 실제 메모리의 크기가 입력 그래프보다 훨씬 클 수도 있다. 또한, 메인 메모리 기반의 처리 방식은 임의 그래프에서 임의 질의(쿼리)를 처리하기 위해 필요한 메모리 크기를 정확히 계산하는 것은 매우 어렵기 때문에, 컴퓨팅 노드와 전체 메모리 크기가 충분한지를 판단하기에 어려움이 있다. 따라서 기존의 시스템들은 대규모 그래프 데이터 분석에 있어 메모리 부족 문제를 가진다.

[0004] 두번째 방법은 디스크 등의 외부 저장 장치를 활용하는 것이다. 큰 그래프 데이터는 디스크에 저장해두고 디스크와 메모리 사이의 입출력을 통해서 그래프를 부분적으로 처리하는 방식이다. 대표적인 시스템으로 Chaos와 HybridGraph가 있다. 하지만 이러한 시스템들은 앞서 언급한 메인 메모리 기반 시스템들에 비해 처리 효율이 매우 떨어져 질의 처리 속도가 매우 느린 단점을 가진다.

[0005] 분산 대규모 그래프 처리 시스템의 평가에 있어서 대규모 그래프 처리 뿐만 아니라 빠른 질의 처리도 중요한 요소이다. Chaos는 디스크를 활용해 메모리 부족 문제 없이 대규모 그래프를 처리할 수 있는 우수한 확장성을 가진 시스템이다. 하지만, Chaos는 질의를 처리할 때, 디스크 입출력이 지나치게 많고 데이터 처리를 비효율적으로 수행하기 때문에 최신 메인 메모리 기반의 시스템들(Gemini와 Pregel+)에 비해서 처리 속도가 현저하게 느리

다는 단점이 있다. 또한, 다른 디스크 기반의 시스템인 HybridGraph는 메시지를 디스크에 입출력하지 않는 방법을 통해 처리 속도를 개선하고자 하였으나, 여전히 최신 메인 메모리 기반의 시스템과 비교하여 처리 시간이 매우 느리다는 단점을 가진다.

[0006] 분산 환경에서 그래프를 처리하는 여러 시스템들이 분산 컴퓨팅 노드들 간의 균형 있는 워크로드를 성취하기 위해 다양한 분할 기법을 제안하였다. 대부분의 분할 기법들은 분산 환경에서의 네트워크 I/O 트래픽을 줄이는데 초점을 맞추었다. 하지만, 고속 네트워크 장비가 장착된 현대의 클러스터 환경에서 네트워크 트래픽은 더이상 병목이 아니다. 또한 최신 분할 프로그램 중 하나인 METIS는 분할시 메모리 요구량이 엄청나게 많아서 작은 그래프를 분할할 때도 메모리 부족 현상으로 프로그램 수행에 실패하는 경우가 있다.

[0007] 따라서, 대규모 그래프를 안정적으로 분할하고, 메모리 부족없이 대규모 그래프 데이터를 처리할 수 있는 기술에 대한 필요성이 존재한다.

## 발명의 내용

### 해결하려는 과제

[0008] 본 개시는 상술한 문제점을 해결하기 위한 것으로, 본 개시의 목적은 질의 수행시 메모리 요구량을 한정하고, 대규모 그래프를 복수의 장치에서 분산 처리할 때 장치간 균형있는 워크로드를 성취할 수 있는 그래프 데이터 분할의 안정적 처리 방법과 그래프 데이터 처리 장치를 제공하는 것이다.

### 과제의 해결 수단

[0009] 이상과 같은 목적을 달성하기 위한 본 개시의 일 실시 예에 따르면, 그래프 데이터 처리 방법은 저장부에 저장된 복수의 정점과 엣지를 포함하는 그래프 데이터 중 메모리 요구량에 기초하여 기 설정된 개수의 소스 정점, 상기 소스 정점에 기초한 엣지 리스트를 포함하는 서브 그래프 데이터를 로딩하는 단계, 상기 각각의 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행하는 단계, 상기 제1 도착 정점이 식별된 소스 정점에 기초하여 엣지 리스트를 로딩하는 단계, 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하는 제2 레벨 과정을 수행하는 단계 및 상기 소스 정점, 상기 제1 도착 정점 및 상기 제2 도착 정점에 기초하여 쿼리를 처리하는 단계를 포함할 수 있다.

[0010] 그리고, 그래프 데이터 처리 방법은 상기 제k-1 도착 정점이 식별된 소스 정점에 기초하여 엣지 리스트를 로딩하는 단계, 상기 제k-1 도착 정점이 식별된 소스 정점과 연결된 제k 도착 정점을 식별하는 제k 레벨 과정을 수행하는 단계를 더 포함하고, 상기 쿼리를 처리하는 단계는 상기 소스 정점, 상기 제1 도착 정점, 상기 제2 도착 정점 및 상기 제k-1 도착 정점 및 상기 제k 도착 정점에 기초하여 3개 보다 많은 n개의 각을 가지는 다각형 연결 관계 쿼리 및 연결 관계 패턴 검색 쿼리를 처리할 수 있다. 여기서, k는 $3 \leq k \leq n-1$ 인 자연수이다.

[0011] 또한, 상기 제1 레벨 과정을 수행하는 단계는 상기 소스 정점과 연결된 기 설정된 개수 이내의 엣지를 포함하는 제1 윈도우를 설정하고, 상기 설정된 제1 윈도우를 순차적으로 슬라이딩시키며, 상기 제1 윈도우가 위치한 영역에 포함된 소스 정점 및 엣지 리스트에 기초하여 제1 도착 정점을 식별할 수 있다.

[0012] 그리고, 상기 제2 레벨 과정을 수행하는 단계는 상기 제1 도착 정점이 식별된 소스 정점을 기초로 상기 소스 정점과 연결된 엣지를 포함하는 제2 윈도우를 설정하고, 상기 설정된 제2 윈도우를 순차적으로 슬라이딩시키며, 상기 제2 윈도우가 위치한 영역에 포함된 상기 제1 도착 정점이 식별된 소스 정점 및 엣지 리스트에 기초하여 제2 도착 정점을 식별할 수 있다.

[0013] 또한, 상기 제2 레벨 과정을 수행하는 단계는 상기 프로세서가 현재 위치한 상기 제2 윈도우의 영역 내의 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하고, 상기 저장부가 상기 제2 윈도우가 다음에 위치할 영역에 대응되는 상기 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 상기 메모리로 로딩하며, 통신 인터페이스가 이전에 위치했던 상기 제2 윈도우의 영역에서 식별된 제2 도착 정점에 대한 업데이트 정보를 전송함으로써 상기 저장부, 상기 프로세서 및 상기 통신인터페이스가 병렬적으로 동시에 동작할 수 있다.

[0014] 한편, 상기 제1 레벨 과정을 수행하는 단계는 상기 소스 정점과 상기 제1 도착 정점 각각에 부여된 번호에 기초하여 상기 소스 정점과 상기 제1 도착 정점을 각각 올림차순으로 배열하고, 상기 제2 레벨 과정을 수행하는 단계는 상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점 각각에 부여된 번호에 기초하여 상기 제1

도착 정점이 식별된 소스 정점과 상기 제2 도착 정점을 각각 올림차순으로 배열할 수 있다.

[0015] 그리고, 상기 제1 레벨 과정을 수행하는 단계는 상기 소스 정점의 번호가 상기 제1 도착 정점의 번호보다 작은 경우 상기 소스 정점의 번호보다 큰 번호의 인접 소스 정점을 기초로 연결 관계를 식별하고, 상기 제1 도착 정점의 번호가 상기 소스 정점의 번호보다 작은 경우 상기 제1 도착 정점의 번호보다 큰 번호의 인접 제1 도착 정점을 기초로 연결 관계를 식별하며, 상기 제2 레벨 과정을 수행하는 단계는 상기 제1 도착 정점이 식별된 소스 정점의 번호가 상기 제2 도착 정점의 번호보다 작은 경우 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 큰 번호의 인접 제1 도착 정점이 식별된 소스 정점을 기초로 연결 관계를 식별하고, 상기 제2 도착 정점의 번호가 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 작은 경우 상기 제2 도착 정점의 번호보다 큰 번호의 인접 제2 도착 정점을 기초로 연결 관계를 식별할 수 있다.

[0016] 이상과 같은 목적을 달성하기 위한 본 개시의 일 실시 예에 따르면, 그래프 데이터 처리 장치는 복수의 정점과 엣지를 포함하는 그래프 데이터를 저장하는 저장부, 상기 그래프 데이터 중 메모리 요구량에 기초하여 서브 그래프 데이터를 로딩하는 메모리 및 상기 로딩된 서브 그래프 데이터에 기초하여 쿼리를 처리하는 프로세서를 포함하고, 상기 메모리는 기 설정된 개수의 상기 서브 그래프 데이터의 소스 정점, 상기 소스 정점에 기초한 엣지 리스트를 로딩하고, 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 로딩하며, 상기 프로세서는 상기 메모리에 로딩된 소스 정점에 기초한 엣지 리스트에 기초하여 상기 각각의 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행하고, 상기 메모리에 로딩된 상기 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트에 기초하여 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하는 제2 레벨 과정을 수행하며, 상기 소스 정점, 상기 제1 도착 정점 및 상기 제2 도착 정점에 기초하여 쿼리를 처리할 수 있다.

[0017] 그리고, 상기 메모리는 제k-1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 로딩하며, 상기 프로세서는 상기 메모리에 로딩된 상기 제k-1 도착 정점이 식별된 소스 정점과 연결된 제k 도착 정점을 식별하는 제k 레벨 과정을 수행하고, 상기 소스 정점, 상기 제1 도착 정점, 상기 제2 도착 정점 및 상기 제k-1 도착 정점 및 상기 제k 도착 정점에 기초하여 3개 보다 많은 n개의 꼭지점을 가지는 다각형 연결 관계 쿼리 및 연결 관계 패턴 검색 쿼리를 처리할 수 있다. 여기서, k는 $3 \leq k \leq n - 1$ 인 자연수이다.

[0018] 그리고, 상기 프로세서는 상기 제1 레벨 과정에서, 상기 소스 정점과 연결된 기 설정된 개수 이내의 엣지를 포함하는 제1 윈도우를 설정하고, 상기 설정된 제1 윈도우를 순차적으로 슬라이딩시키며, 상기 제1 윈도우가 위치한 영역에 포함된 소스 정점 및 엣지 리스트에 기초하여 제1 도착 정점을 식별할 수 있다.

[0019] 또한, 상기 프로세서는 상기 제2 레벨 과정에서, 상기 제1 도착 정점이 식별된 소스 정점을 기초로 상기 제1 도착 정점이 식별된 소스 정점과 연결된 엣지를 포함하는 제2 윈도우를 설정하고, 상기 설정된 제2 윈도우를 순차적으로 슬라이딩시키며, 상기 제2 윈도우가 위치한 영역에 포함된 상기 제1 도착 정점이 식별된 소스 정점 및 엣지 리스트에 기초하여 제2 도착 정점을 식별할 수 있다.

[0020] 한편, 그래프 데이터 처리 장치는 통신 인터페이스를 더 포함하고, 상기 프로세서는 상기 제2 레벨 과정에서, 현재 위치한 상기 제2 윈도우의 영역 내의 상기 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하고, 상기 제2 윈도우가 다음에 위치할 영역에 대응되는 상기 제1 도착 정점이 식별된 소스 정점에 기초한 엣지 리스트를 상기 메모리에 로딩하도록 상기 저장부를 제어하며, 이전에 위치했던 상기 제2 윈도우의 영역에서 식별된 제2 도착 정점에 대한 업데이트 정보를 전송하도록 상기 통신 인터페이스를 제어함으로써 상기 저장부, 상기 프로세서 및 상기 통신 인터페이스를 병렬적으로 동시에 동작시킬 수 있다.

[0021] 한편, 상기 프로세서는 상기 제1 레벨 과정에서, 상기 소스 정점과 상기 제1 도착 정점 각각에 부여된 번호에 기초하여 상기 소스 정점과 상기 제1 도착 정점을 각각 올림차순으로 배열하고, 상기 제2 레벨 과정에서, 상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점 각각에 부여된 번호에 기초하여 상기 제1 도착 정점이 식별된 소스 정점과 상기 제2 도착 정점을 각각 올림차순으로 배열할 수 있다.

[0022] 그리고, 상기 프로세서는 상기 제1 레벨 과정에서, 상기 소스 정점의 번호가 상기 제1 도착 정점의 번호보다 작은 경우 상기 소스 정점의 번호보다 큰 번호의 인접 소스 정점을 기초로 연결 관계를 식별하고, 상기 제1 도착 정점의 번호가 상기 소스 정점의 번호보다 작은 경우 상기 제1 도착 정점의 번호보다 큰 번호의 인접 제1 도착 정점을 기초로 연결 관계를 식별하며, 상기 제2 레벨 과정에서, 상기 제1 도착 정점이 식별된 소스 정점의 번호가 상기 제2 도착 정점의 번호보다 작은 경우 상기 제1 도착 정점이 식별된 소스 정점의 번호보다 큰 번호의 인접 상기 제1 도착 정점이 식별된 소스 정점을 기초로 연결 관계를 식별하고, 상기 제2 도착 정점의 번호가 상기

제1 도착 정점이 식별된 소스 정점의 번호보다 작은 경우 상기 제2 도착 정점의 번호보다 큰 번호의 인접 제2 도착 정점을 기초로 연결 관계를 식별할 수 있다.

[0023] 이상과 같은 목적을 달성하기 위한 본 개시의 일 실시 예에 따르면, 시스템은 상술한 그래프 데이터 처리 장치를 복수 개 포함하고, 상기 복수 개 그래프 데이터 처리 장치 각각은 소스 정점 및 연결된 엣지의 개수에 기초하여 라운드 로빈 방식으로 상기 소스 정점 및 상기 엣지를 포함하는 그래프 데이터를 분산하여 저장할 수 있다.

## 발명의 효과

[0024] 이상 설명한 바와 같이, 본 개시의 다양한 실시 예에 따르면, 그래프 데이터 처리 방법 및 그래프 데이터 처리 장치는 메모리 부족 현상없이 대규모 그래프 데이터를 처리할 수 있다.

[0025] 그리고, 그래프 데이터 처리 방법 및 그래프 데이터 처리 장치는 대규모 그래프 데이터를 안정적으로 분할할 수 있고, 장치간 균형있는 워크로드를 성취할 수 있다.

[0026] 본 발명의 효과들은 이상에서 언급한 효과들로 제한되지 않으며, 언급되지 않은 또 다른 효과들은 아래의 기재로부터 통상의 기술자에게 명확하게 이해 될 수 있을 것이다.

## 도면의 간단한 설명

[0027] 도 1은 본 개시의 일 실시 예에 따른 그래프 데이터 처리 장치의 블록도이다.

도 2a는 본 개시의 일 실시 예에 따른 정점과 엣지를 식별하는 알고리즘을 나타내는 도면이다.

도 2b는 본 개시의 일 실시 예에 따른 전역 수집(global gather) 작업 알고리즘을 나타내는 도면이다.

도 2c는 본 개시의 일 실시 예에 따른 업데이트 수집 작업 알고리즘을 나타내는 도면이다.

도 2d는 본 개시의 일 실시 예에 따른 적용(apply) 작업 알고리즘을 나타내는 도면이다.

도 3은 데이터 그래프의 일 실시 예를 설명하는 도면이다.

도 4는 본 개시의 일 실시 예에 따른 페이지 링크 쿼리를 처리하는 과정을 설명하는 도면이다.

도 5는 본 개시의 일 실시 예에 따른 삼각형 카운팅 쿼리를 처리하는 과정을 설명하는 도면이다.

도 6은 본 개시의 일 실시 예에 따른 하드웨어 병렬 처리 과정을 설명하는 도면이다.

도 7은 복수의 장치에 균형적으로 그래프 데이터를 분산 저장하는 일 실시 예를 설명하는 도면이다.

도 8a은 다양한 시스템의 전처리 시간 테스트 결과를 나타내는 도면이다.

도 8b 및 도 8c는 대규모 그래프 데이터에 대한 다양한 시스템의 쿼리 처리 시간 테스트 결과를 나타내는 도면이다.

도 8d 내지 도 8h는 현실에 존재하는 공개 그래프 데이터에 대한 다양한 시스템의 쿼리 처리 시간 테스트 결과를 나타내는 도면이다.

도 9는 본 개시의 일 실시 예에 따른 그래프 데이터 처리 방법 흐름도를 설명하는 도면이다.

## 발명을 실시하기 위한 구체적인 내용

[0028] 이하에서는 첨부된 도면을 참조하여 다양한 실시 예를 보다 상세하게 설명한다. 본 명세서에 기재된 실시 예는 다양하게 변형될 수 있다. 특정한 실시 예가 도면에서 묘사되고 상세한 설명에서 자세하게 설명될 수 있다. 그러나, 첨부된 도면에 개시된 특정한 실시 예는 다양한 실시 예를 쉽게 이해하도록 하기 위한 것일 뿐이다. 따라서, 첨부된 도면에 개시된 특정 실시 예에 의해 기술적 사상이 제한되는 것은 아니며, 발명의 사상 및 기술 범위에 포함되는 모든 균등물 또는 대체물을 포함하는 것으로 이해되어야 한다.

[0029] 제1, 제2 등과 같이 서수를 포함하는 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 이러한 구성요소들은 상술한 용어에 의해 한정되지는 않는다. 상술한 용어는 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다.

[0030] 본 명세서에서, "포함한다" 또는 "가지다" 등의 용어는 명세서상에 기재된 특징, 숫자, 단계, 동작, 구성요소,

부품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다. 어떤 구성요소가 다른 구성요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는, 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다.

[0031] 한편, 본 명세서에서 사용되는 구성요소에 대한 "모듈" 또는 "부"는 적어도 하나의 기능 또는 동작을 수행한다. 그리고, "모듈" 또는 "부"는 하드웨어, 소프트웨어 또는 하드웨어와 소프트웨어의 조합에 의해 기능 또는 동작을 수행할 수 있다. 또한, 특정 하드웨어에서 수행되어야 하거나 적어도 하나의 프로세서에서 수행되는 "모듈" 또는 "부"를 제외한 복수의 "모듈들" 또는 복수의 "부들"은 적어도 하나의 모듈로 통합될 수도 있다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다.

[0032] 본 발명의 설명에 있어서 각 단계의 순서는 선행 단계가 논리적 및 시간적으로 반드시 후행 단계에 앞서서 수행되어야 하는 경우가 아니라면 각 단계의 순서는 비제한적으로 이해되어야 한다. 즉, 위와 같은 예외적인 경우를 제외하고는 후행 단계로 설명된 과정이 선행단계로 설명된 과정보다 앞서서 수행되더라도 발명의 본질에는 영향이 없으며 권리범위 역시 단계의 순서에 관계없이 정의되어야 한다. 그리고 본 명세서에서 "A 또는 B"라고 기재한 것은 A와 B 중 어느 하나를 선택적으로 가리키는 것뿐만 아니라 A와 B 모두를 포함하는 것도 의미하는 것으로 정의된다. 또한, 본 명세서에서 "포함"이라는 용어는 포함하는 것으로 나열된 요소 이외에 추가로 다른 구성요소를 더 포함하는 것도 포괄하는 의미를 가진다.

[0033] 본 명세서에서 수행되는 정보(데이터) 전송 과정은 필요에 따라서 암호화/복호화가 적용될 수 있으며, 본 명세서 및 특허청구범위에서 정보(데이터) 전송 과정을 설명하는 표현은 별도로 언급되지 않더라도 모두 암호화/복호화하는 경우도 포함하는 것으로 해석되어야 한다. 본 명세서에서 "A로부터 B로 전송(전달)" 또는 "A가 B로부터 수신"과 같은 형태의 표현은 중간에 다른 매개체가 포함되어 전송(전달) 또는 수신되는 것도 포함하며, 반드시 A로부터 B까지 직접 전송(전달) 또는 수신되는 것만을 표현하는 것은 아니다.

[0034] 본 명세서에서는 본 발명의 설명에 필요한 필수적인 구성요소만을 설명하며, 본 발명의 본질과 관계가 없는 구성요소는 언급하지 아니한다. 그리고 언급되는 구성요소만을 포함하는 배타적인 의미로 해석되어서는 아니되며 다른 구성요소도 포함할 수 있는 비배타적인 의미로 해석되어야 한다.

[0035] 그 밖에도, 본 발명을 설명함에 있어서, 관련된 공지 기능 혹은 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우, 그에 대한 상세한 설명은 축약하거나 생략한다. 한편, 각 실시 예는 독립적으로 구현되거나 동작될 수도 있지만, 각 실시 예는 조합되어 구현되거나 동작될 수도 있다.

[0036] 도 1은 본 개시의 일 실시 예에 따른 그래프 데이터 처리 장치의 블록도이다.

[0037] 도 1을 참조하면, 그래프 데이터 처리 장치(100)는 저장부(110), 메모리(120) 및 프로세서(130)를 포함할 수 있다.

[0038] 저장부(110)는 그래프 데이터를 저장할 수 있다. 그래프 데이터는 그래프 데이터 전체일 수 있고, 대규모 그래프 데이터인 경우 대규모 그래프 데이터의 일부 그래프 데이터를 저장할 수도 있다. 그래프 데이터는 정점과 정점으로부터 다른 정점으로 연결된 엣지(간선)를 포함할 수 있다. 즉, 저장부(110)는 복수의 정점과 엣지를 포함하는 그래프 데이터를 저장할 수 있다.

[0039] 메모리(120)는 그래프 데이터 중 메모리 요구량에 기초하여 서브 그래프 데이터를 로딩할 수 있다. 서브 그래프 데이터는 소스 정점, 소스 정점에 기초한 엣지 리스트를 포함할 수 있다. 일 실시 예로서, 메모리(120)는 영역을 나누고 각 영역에 소스 정점을 저장할 수 있다.

[0040] 예를 들어, 저장부(110)는 하드디스크, SSD, 플래시 메모리, 멀티미디어 카드, 자기 메모리, 자기 디스크, 광디스크 등을 포함할 수 있고, 메모리(120)는 램, 롬, 프로세서(120) 내부의 버퍼 또는 캐쉬 등을 포함할 수 있다.

[0041] 프로세서(120)는 소스 정점과 연결된 연결 관계를 식별할 수 있다. 즉, 프로세서(120)는 기 설정된 개수의 서브 그래프 데이터의 소스 정점, 소스 정점에 기초한 엣지 리스트를 로딩하여 각각이 소스 정점과 연결된 제1 도착 정점을 식별할 수 있다. 한편, 프로세서(120)는 사용자의 명령에 따라 쿼리(질의)를 수행할 수 있다. 만일, 쿼리가 삼각형(또는, 다각형)의 연결 관계를 식별하는 쿼리인 경우, 프로세서(120)는 상술한 과정과 유사한 과정

을 반복할 수 있다.

[0042] 예를 들어, 프로세서(120)가 삼각형의 연결 관계를 식별하는 쿼리를 수행하는 경우, 프로세서(120)는 상술한 과정을 통해 제1 도착 정점을 식별할 수 있다. 그리고, 프로세서(120)는 식별된 제1 도착 정점에 기초하여 엣지 리스트를 로딩할 수 있다. 프로세서(120)는 식별된 제1 도착 정점 및 로딩된 엣지 리스트에 기초하여 제2 도착 정점을 식별할 수 있다. 프로세서(120)는 소스 정점, 식별된 제1 도착 정점 및 식별된 제2 도착 정점에 기초하여 삼각형의 그래프를 식별하는 쿼리를 수행할 수 있다. 만일, 프로세서(120)가 다각형의 그래프를 식별하는 쿼리를 수행하는 경우, 상술한 과정을 반복하여 다각형의 그래프를 식별하는 쿼리를 수행할 수 있다. 삼각형 그래프를 식별하는 쿼리를 수행하는 구체적인 예는 후술한다.

[0043] 한편, 그래프 데이터 처리 장치(100)는 통신 인터페이스(미도시)를 더 포함할 수 있다. 통신 인터페이스는 쿼리 수행 결과 업데이트된 정보를 다른 장치로 전송함으로써 각 정점 간의 정보를 동기화시킬 수 있다.

[0044] 도 2a는 본 개시의 일 실시 예에 따른 정점과 엣지를 식별하는 알고리즘을 나타내는 도면이다.

[0045] 분산 데이터 처리 과정은 분산(scatter) 단계, 수집(gather) 단계, 적용(apply) 단계의 순서로 수행될 수 있다. 도 2a에 개시된 알고리즘은 다음과 같은 과정으로 수행되는 알고리즘이다.

[0046] 도 2a에 개시된 ProcessNWSM 함수는 레벨 l과 l번째 레벨의 정점 스트림($vs^{l}$), l번째 레벨의 인접 리스트 스트림($adjs^{l}$)을 인자로 받을 수 있다. 레벨 l은 도착 정점을 식별하는 과정을 의미할 수 있다. 예를 들어, 상술한 제1 도착 정점을 식별하는 과정은 제1 레벨일 수 있고, 제2 도착 정점을 식별하는 과정은 제2 레벨일 수 있다. 정점 스트림은 연결 관계를 식별할 초기 정점일 수 있다. 예를 들어, 상술한 제1 레벨의 정점 스트림은 적어도 하나의 소스 정점일 수 있고, 제2 레벨의 정점 스트림은 적어도 하나의 제1 도착 정점일 수 있다. 인접 리스트 스트림은 초기 정점으로부터 연결된 엣지 리스트일 수 있다.

[0047] 본 개시의 실시 예는 그래프 데이터 처리에 필요한 메모리 요구량에 기초하여 서브 그래프 데이터를 로딩할 수 있다. 최소 메모리 요구량은 분할 매개 변수인 q와 관계가 있다. 따라서, 프로세서는 메모리 부족 현상 없이 쿼리를 수행하기 위해 최소 q 값은 $q_{new}$를 산출할 수 있다. q는 각 장치에 균형적으로 그래프 데이터를 분할하기 위해 사용되는 매개 변수일 수 있다. 만일, $q_{new}$값이 q보다 크면 분할은 더 세분화되어야 한다. $q_{new}$값이 q보다 작으면 현재 그래프 분할이 이용될 수 있다.

[0048] 분산 단계를 시작하기 전에, 각 장치는 분산 단계로부터 생성될 메시지를 모으기 위한 전역 수집 비동기 태스크를 수행할 수 있다. l번째 레벨의 분산 단계에서 프로세서는 $vs^{l}$로부터 $vw^{l}$을 리딩할 수 있다. $vw^{l}$은 상술한 분할된 임시 메모리 영역을 의미할 수 있고, 정점 연결 관계를 식별하기 위해 정점 집합에 순차적으로 슬라이딩하는 윈도우를 의미할 수 있다. 즉, 윈도우는 분할된 메모리 영역일 수 있다. 그리고, 프로세서는 $adjs^{l}$로부터 $adjw^{l}$을 리딩할 수 있다. $adjw^{l}$도 엣지 리스트의 윈도우를 의미할 수 있다. 도 2a에 도시된 알고리즘은 $adjw^{l}$의 인접 리스트에 대한 입출력(I/O)를 완료하였을 때, 인접 리스트에 대한 회신 함수인 $adj_scatter^{l}$을 호출할 수 있다. 즉, 프로세서 연산은 네트워크 I/O, 원격 메모리(예, 외부 장치 디스크) I/O와 중첩될 수 있다. 모든 장치들이 분산 단계를 동시에 수행하기 때문에 시스템 전체의 프로세서, 네트워크, 저장부 자원들은 완전히 중첩되어 작업이 수행될 수 있다.

[0049] 분산 단계에서는 프로세서는 다른 장치로 업데이트 정보를 전송하기 전에 메모리 상에서 전역 수집 과정을 지원하기 위해 각 레벨마다 전역 수집 버퍼($LGB^{l}$)를 할당할 수 있다. $adj\_scatter^{l}$ 함수에서 타겟 정점이 업데이트될 때, 업데이트 스트림을 줄이기 위해 메모리가 전역 수집 과정을 수행할 수 있다. 상술한 과정을 통해, 본 개시는 수집 단계와 전역 수집 단계를 중첩시킬 수 있다.

[0050] 도 2b는 본 개시의 일 실시 예에 따른 전역 수집(global gather) 작업 알고리즘을 나타내는 도면이다.

[0051] 도 2b를 설명하면, 전역 수집 작업 알고리즘은 각 업데이트 u에 대해 u가 메모리에서 수집될 수 있는 경우, 전역 수집 버퍼에 u를 누적시킬 수 있다. 업데이트 u가 메모리에서 수집될 수 없는 경우, 저장부에 저장될 수 있다. 전역 수집 작업 알고리즘은 장치들의 수집 단계에서 생성된 메시지를 수집할 수 있다. 전역 수집 작업 알고리즘은 $vs^{l}$과 $adjs^{l}$을 모두 처리한 후, 전역 장벽을 통해 모든 업데이트들이 메모리 또는 저장부에 모일 때까지 대기할 수 있다. 이후, 전역 수집 작업 알고리즘은 저장부에 저장된 나머지 업데이트 정보들을 수집하는 작업을

수행하고, 적용 작업을 수행할 수 있다.

[0052] 전역 수집 작업을 수행하는 장치가 생산자이고, 적용 작업을 수행하는 장치가 소비자라면, 메모리는 전역 수집 버퍼를 추가로 할당함으로써 생산자와 소비자를 위한 두 개의 버퍼를 포함할 수 있다. 그리고, 장치는 수집 단계의 수행과 적용 단계의 수행을 중첩시킬 수 있다.

[0053] 도 2c는 본 개시의 일 실시 예에 따른 업데이트 수집 작업 알고리즘을 나타내는 도면이며, 도 2d는 본 개시의 일 실시 예에 따른 적용(apply) 작업 알고리즘을 나타내는 도면이다.

[0054] 도 2c를 참조하면, GatherSpilledUpdates 함수가 저장부로부터 업데이트들을 로딩하고 다른 전역 수집 버퍼에 누적하는 동안, 적용 작업 알고리즘은 전역 수집 버퍼를 이용하여 적용 작업이 즉시 수행될 수 있다. 적용 단계는 현재의 소스 정점 ID 범위에 해당하는 저장부에 저장된 결과들의 병합이 종료될 때까지 대기하여 생산자와 소비자를 동기화시킬 수 있다. 적용 작업 알고리즘은 정점 스트림의 각 정점들에 대해 최종적으로 vertex_apply를 호출할 수 있다.

[0055] 아래에서는 그래프 탐색을 수행하는 구체적인 과정을 설명한다.

[0056] 도 3은 데이터 그래프의 일 실시 예를 설명하는 도면이고, 도 4는 본 개시의 일 실시 예에 따른 페이지 링크 쿼리를 처리하는 과정을 설명하는 도면이며, 도 5는 본 개시의 일 실시 예에 따른 삼각형 카운팅 쿼리를 처리하는 과정을 설명하는 도면이다.

[0057] 정점 v가 또 다른 정점 u로부터 k-hop reachable하다는 것은 u에서 v로의 길이 k이하인 경로가 있다는 것을 의미한다. Walk는 정점의 배열(a sequence of vertices)를 의미하는 패스에서 동일한 정점이 배열에 여러 번 나타날 수 있는 일반적인 경우를 의미한다.

[0058] 각 정점으로부터 k-hop 떨어진 이웃 정점들을 포함하는 서브그래프를 k-hop neighborhood라고 한다. k-hop neighborhood에서의 그래프 탐색을 지원하는 일반적인 쿼리 클래스로 k-walk neighborhood query를 정의할 수 있다. K-walk neighborhood query는 각 정점의 k-hop neighborhood에 포함된 모든 정점들로 이어지는 Walk에 대해서 연산 수행을 필요로 한다. 예를 들어, 페이지 랭크 쿼리는 1-walk neighborhood query이며 삼각형 찾기 쿼리는 2-walk neighborhood query이다.

[0059] 도 3 및 도 4를 참조하여, 도 3의 그래프에 대한 페이지 랭크 쿼리를 처리하는 예를 설명한다. 도 4를 참조하면, 소스 정점이 메모리 요구량에 기초하여 메모리 로딩될 수 있다. 일 실시 예로서, 메모리 요구량은 정점의 개수에 기초하여 설정될 수 있다. 도 3의 정점은 v0 내지 v5까지 6개가 있고, v0 내지 v2(vw1-1)과 v3 내지 v5(vw1-2)로 나뉘어서 처리될 수 있다. 먼저, v0 내지 v2(vw1-1)이 메모리에 로딩될 수 있다. 그리고, $adjs^1$이 엣지 리스트의 소스 정점이 포함된 정점 윈도우에 맞게 분할될 수 있다. vw1-1에는 정점 v0 내지 v2에 해당하는 정점의 특성 값(attribute value)이 로딩되고, adjw1-11의 엣지 리스트가 처리되면서 페이지 랭크 값은 업데이트될 수 있다. 다음으로 $vs^1$과 $adjs^1$의 윈도우를 슬라이딩하여 vw1-2 및 adjw1-21이 처리될 수 있다. 이후, 생성된 업데이트 정보들을 병합(aggregation)하는 수집(gather) 과정이 수행되고, 병합된 값으로 정점 값을 업데이트하는 적용(apply) 과정이 수행될 수 있다. vw1-2 및 adjw1-21은 vw1-1 및 adjw1-11이 처리된 이후 메모리에 로딩될 수 있다. 또는, vw1-2 및 adjw1-21은 vw1-1 및 adjw1-11과 함께 메모리에 로딩된 후 윈도우를 vw1-2 및 adjw1-21로 슬라이딩시켜 처리될 수도 있다.

[0060] 도 3 및 도 5를 참조하여, 도 3의 그래프에 대한 삼각형 카운팅 쿼리를 처리하는 예를 설명한다. 삼각형 카운팅 쿼리는 2-hop neighborhood query로서, 두 개의 정점 윈도우(Vertex Window)와 엣지 리스트 윈도우(Adjacency List Window)의 쌍을 겹쳐서 처리될 수 있다.

[0061] 상술한 바와 같이, 소스 정점은 메모리 요구량에 기초하여 메모리에 로딩될 수 있다. 제1 레벨에서는 vw1-1과 adjw1-11이 메모리에 로딩될 수 있다. 그러나, 프로세서는 설정된 엣지 개수에 기초하여 하나의 작업을 수행할 수 있다. 도 5에는 하나의 작업에 5개의 엣지가 처리될 수 있는 케이스가 도시되어 있다. 도 5에서 5개의 엣지가 처리될 수 있으므로 정점 윈도우는 vw1-1에 위치시키고, 엣지 리스트 윈도우는 ajdw1-11에 위치될 수 있다. adjw1-11에 로딩된 엣지 중 Partial Order Constraint를 만족하는 엣지를 따라서 도착 정점이 식별될 수 있다. 식별된 도착 정점은 vs2로 구성되어 제2 레벨에서 처리될 수 있다. 제2 레벨에서는 vw2-1과 adjw2-11이 로딩될 수 있다. 제2 레벨에 로딩된 각 정점 u로부터 역방향 횡단(backward traversal)이 수행되고, 제1 레벨에 있는 부모 정점 v를 검색할 수 있다. 이후 u와 v의 엣지 리스트에 대해 교집합 연산이 수행되어 삼각형 그래프를 식별할 수 있다.

[0062] 즉, 메모리는 저장부에 저장된 그래프 데이터 중 메모리 요구량에 기초하여 기 설정된 개수의 소스 정점, 소스 정점에 기초한 엣지 리스트를 포함하는 서브 그래프 데이터를 로딩할 수 있다. 예를 들어, 메모리 요구량은 정점의 개수에 기초하여 설정될 수 있고, 엣지의 개수에 기초하여 설정될 수도 있다. 프로세서는 메모리에 로딩된 소스 정점과 엣지 리스트에 기초하여 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행할 수 있다. 도 5에 도시된 바와 같이, 메모리는 vw1-1, adjw1-11 및 adjw1-12를 로딩할 수 있다. 5개의 엣지가 처리될 수 있으므로 프로세서는 정점 윈도우는 vw1-1에 위치시키고, 엣지 리스트 윈도우는 adjw1-11에 위치될 수 있다. 메모리에 로딩된 vw1-1과 adjw1-11에 기초하여 제1 도착 정점을 식별할 수 있다. 다음으로 프로세서는 정점 윈도우의 위치를 vw1-1로 유지시키고, 엣지 리스트 윈도우를 adjw1-12로 슬라이딩시켜 v2에 기초한 제1 도착 정점을 식별할 수 있다.

[0063] 그리고, 메모리는 제1 도착 정점이 식별된 소스 정점(v1, v2)에 기초하여 엣지 리스트를 로딩할 수 있다. 제2 레벨에서 프로세서는 제1 도착 정점이 식별된 소스 정점에 기초하여 제2 도착 정점을 식별할 수 있다. 따라서, 도 5에 도시된 바와 같이, 메모리는 vw2-1 영역에 v1, v2 정점을 로딩할 수 있다. 프로세서는 vw2-1 영역에 정점 윈도우를 위치시키고, adjw2-11에 엣지 리스트 윈도우를 위치시켜 v1에 대한 제2 도착 정점을 식별할 수 있다. 그리고, 프로세서는 정점 윈도우 위치를 유지시키고, adjw2-12로 엣지 리스트 윈도우를 슬라이딩시켜 v2에 대한 제2 도착 정점을 식별할 수 있다. 상술한 과정을 통해 프로세서는 소스 정점 v0, v1, v2에 대해 소스 정점, 식별된 제1 도착 정점, 식별된 제2 도착 정점에 기초하여 삼각형 검색 쿼리를 수행할 수 있다.

[0064] 메모리는 제1 레벨에서 로딩된 정점 및 엣지 리스트를 유지할 수 있다. 그리고, 메모리는 검색된 삼각형 정보를 제외하고 제2 레벨에서 로딩된 정점 및 엣지 리스트는 제거할 수 있다.

[0065] 프로세서는 동일한 방식으로 다른 정점에 대해서도 삼각형 검색 쿼리를 수행할 수 있다. 메모리는 vw1-2의 정점을 로딩하고 프로세서는 상술한 과정을 반복하여 삼각형 검색 쿼리를 수행할 수 있다.

[0066] 한편, 프로세서는 제1 레벨에서 소스 정점과 제1 도착 정점 각각에 부여된 번호에 기초하여 소스 정점과 제1 도착 정점을 각각 순서대로 배열할 수 있다. 일 실시 예로서, 프로세서는 제1 레벨에서 소스 정점과 제1 도착 정점 각각에 부여된 번호에 기초하여 소스 정점과 제1 도착 정점을 각각 올림차순으로 배열할 수 있다.

[0067] 그리고, 프로세서는 소스 정점의 번호가 제1 도착 정점의 번호보다 작은 경우 소스 정점의 번호보다 큰 번호의 인접 소스 정점을 기초로 연결 관계를 식별할 수 있다. 프로세서는 제1 도착 정점의 번호가 소스 정점의 번호보다 작은 경우 제1 도착 정점의 번호보다 큰 번호의 인접 제1 도착 정점을 기초로 연결 관계를 식별할 수 있다.

[0068] 또한, 프로세서는 제2 레벨에서 제1 도착 정점이 식별된 소스 정점과 제2 도착 정점 각각에 부여된 번호에 기초하여 제1 도착 정점이 식별된 소스 정점과 제2 도착 정점을 각각 올림차순으로 배열할 수 있다. 그리고, 프로세서는 제1 도착 정점이 식별된 소스 정점의 번호가 제2 도착 정점의 번호보다 작은 경우 제1 도착 정점이 식별된 소스 정점의 번호보다 큰 번호의 인접 제1 도착 정점이 식별된 소스 정점을 기초로 연결 관계를 식별할 수 있다. 프로세서는 제2 도착 정점의 번호가 제1 도착 정점이 식별된 소스 정점의 번호보다 작은 경우 제2 도착 정점의 번호보다 큰 번호의 인접 제2 도착 정점을 기초로 연결 관계를 식별할 수 있다.

[0069] 또는, 소스 정점, 제1 도착 정점이 식별된 소스 정점 및 제2 도착 정점의 번호를 순서대로 배열하고 중간 값을 기초로 연결 관계를 식별할 수도 있다.

[0070] 본 개시는 소스 정점 및 엣지 리스트에 기초하여 연결 관계를 식별하기 때문에 모든 정점에 대한 비트 연산이 필요하지 않다. 또한, 본 개시는 정점들을 순차적으로 배열하기 때문에 중복 연산을 피할 수 있는 장점이 있다. 따라서, 본 개시는 연산 속도에서 장점을 가질 수 있다.

[0071] 도 5는 일 실시 예로서 삼각형 쿼리 처리 과정을 설명하였으나, 본 개시는 레벨을 추가하고 동일한 과정을 반복하여 사각형 또는 오각형과 같은 다각형 연결 관계를 포함하여 일반적인 연결 관계 패턴 파악에 대한 쿼리 작업도 수행할 수 있다.

[0072] 한편, 본 개시는 하드웨어 자원을 병렬적으로 사용함으로써 전체 처리 효율을 높일 수 있다.

[0073] 도 6은 본 개시의 일 실시 예에 따른 하드웨어 병렬 처리 과정을 설명하는 도면이다.

[0074] 상술한 바와 같이, 그래프 데이터는 사용자가 정의한 UDF(User-defind Function)에 기초하여 각 정점에 대해 UDF를 장치에서 수행함으로써 처리될 수 있다. 장치에서 UDF를 처리함에 있어, 프로세서는 엣지 리스트를 순회하면서 각 엣지의 타겟 정점에의 메시지(업데이트)를 생성하고, 저장부(storage)는 정점 및 엣지 리스트를 메모

리로 로딩하며, 통신 인터페이스는 네트워크를 통해 UDF에서 생성된 메시지를 다른 장치로 전송할 수 있다.

[0075] 따라서, 저장부, 프로세서 및 통신 인터페이스가 서로 다른 작업을 동시에 수행하는 경우, 하드웨어 자원은 효율적으로 사용될 수 있으므로 그래프 데이터 처리 효과는 극대화될 수 있다.

[0076] 도 6에 도시된 바와 같이, 저장부는 다음에 처리될 엣지 리스트를 로딩하고, 프로세서는 UDF를 처리하며, 통신 인터페이스는 업데이트 정보를 다른 장치로 전송할 수 있다.

[0077] 일 실시 예로서, 저장부는 윈도우가 다음에 위치할 영역에 포함되는 정점에 기초한 엣지 리스트를 메모리로 로딩(11)할 수 있다. 동시에, 프로세서는 현재 위치한 윈도우의 영역 내의 정점과 연결된 도착 정점을 식별(12)할 수 있다. 그리고, 동시에 통신 인터페이스는 이전에 위치했던 윈도우 영역에서 식별된 도착 정점에 대한 업데이트 정보를 네트워크를 통해 다른 장치로 전송(13)할 수 있다. 상술한 바와 같이, 저장부는 다음 작업을 준비하고, 프로세서는 현재 작업을 진행하며, 통신 인터페이스는 이전 작업 결과 처리를 동시에 진행함으로써 본 개시는 효율적으로 하드웨어 자원을 사용하고 그래프 데이터 처리 효과를 높일 수 있다.

[0078] 한편, 대규모 그래프 데이터는 여러 장치에 분산되어 저장될 수 있다. 즉, 복수의 그래프 데이터 처리 장치가 하나의 시스템으로 구현될 수도 있다. 따라서, 분할된 그래프 데이터가 복수의 장치에 편향되지 않고 균형적으로 배치되어야 한다. 아래에서는 그래프 데이터를 복수의 장치에 분할하여 저장하는 방법에 대해 설명한다.

[0079] 도 7은 복수의 장치에 균형적으로 그래프 데이터를 분산 저장하는 일 실시 예를 설명하는 도면이다.

[0080] 도 7(a)를 참조하면, 편향된 degree 분포를 가지는 입력 그래프가 도시되어 있다. 색이 표시된 i번째 행과 j번째 열의 한 칸은 정점 ID i인 정점으로부터 정점 ID j인 정점을 향하는 엣지가 존재함을 나타낸다. 장치들 간의 엣지 수와 높은 degree를 가진 정점과 낮은 degree를 가진 정점 수의 균형을 맞추기 위해 정점은 degree 값 순서로 정렬되고, 순차 순환 대기 방식(라운드-로빈 방식)으로 각 장치로 분배될 수 있다. p개의 장치가 존재하는 경우 그래프 데이터는 p개로 분할될 수 있다. 그리고, 각 장치에 할당된 정점에 연속적인 정점 ID가 부여되고, 정점들의 degree 순서 정보는 정점 ID로부터 식별될 수 있다. 각 장치에 있는 정점은 degree 값이 감소하는 순서대로 정점 ID를 부여받을 수 있고, 각 장치에서 높은 degree 값을 가질수록 낮은 정점 ID를 부여받을 수 있다.

[0081] 도 7(b)는 두 개의 장치에서 균형적으로 분배된 결과를 나타낸다.

[0082] 다음으로, 각 장치에 있는 엣지들은 동일 크기의 소스 정점 ID 영역과 타겟 정점 ID 영역으로 나눌 수 있다. 소스 정점은 q개의 영역으로 분할되고, 타겟 정점들은 p * q개의 영역으로 분할될 수 있다. q는 각 장치에서 쿼리를 처리하기 위해 필요한 메모리 버퍼의 크기와 사용 가능한 메모리 크기에 의해 결정될 수 있다. 각 엣지 분할들은 엣지 청크라고 부를 수 있고, 각 장치는 q * pq 개의 엣지 청크를 가질 수 있다. 도 7(c)에는 q가 2일 때 분할 결과가 도시되어 있다.

[0083] 추가적으로 메모리의 수집 연산에서 발생하는 CAS(Compare-And-Swap) 연산에 의한 NUMA 노드 간의 동기화 비용을 줄일 수 있다. 각 엣지 청크는 타겟 정점 ID 영역으로 추가 분할될 수 있다. 각 엣지 청크는 r개의 엣지 부분 청크로 분할될 수 있고, r은 NUMA 노드의 개수와 동일할 수 있다. 각 부분 청크 간의 엣지 개수도 정점들의 degree 정보를 이용하여 균형이 맞도록 조절될 수 있다. 도 7(d)에는 장치 당 두 개의 NUMA 노드가 있을 때 분할 결과가 도시되어 있다.

[0084] 도 8a은 다양한 시스템의 전처리 시간 테스트 결과를 나타내는 도면이고, 도 8b 및 도 8c는 대규모 그래프 데이터에 대한 다양한 시스템의 쿼리 처리 시간 테스트 결과를 나타내는 도면이며, 도 8d 내지 도 8h는 현실에 존재하는 공개 그래프 데이터에 대한 다양한 시스템의 쿼리 처리 시간 테스트 결과를 나타내는 도면이다.

[0085] 도 8a를 참조하면, TurboGraph++이 본 개시이고, 본 개시의 전처리 시간은 최신 그래프 처리 엔진들의 전처리 시간과 유사한 결과가 도시되어 있다. 도 8b 내지 도 8h를 참조하면, TurboGraph++가 본 개시이고, 본 개시의 쿼리 처리 시간은 다른 기법이 적용된 쿼리 처리 수행 시간에 비해 우수한 성능을 나타내는 결과가 도시되어 있다.

[0086] 지금까지 그래프 데이터 처리 장치 및 그래프 데이터 처리 방법의 다양한 실시 예를 설명하였다. 아래에서는 그래프 데이터 처리 방법의 흐름도를 설명한다.

[0087] 도 9는 본 개시의 일 실시 예에 따른 그래프 데이터 처리 방법 흐름도를 설명하는 도면이다.

[0088] 그래프 데이터 처리 장치는 저장부에 저장된 복수의 정점과 엣지를 포함하는 그래프 데이터 중 메모리 요구량에

기초하여 기 설정된 개수의 소스 정점, 소스 정점에 기초한 엣지 리스트를 포함하는 서브 그래프 데이터를 로딩할 수 있다(S910).

[0089] 그래프 데이터 처리 장치는 각각의 소스 정점과 연결된 제1 도착 정점을 식별하는 제1 레벨 과정을 수행할 수 있다(S920). 그래프 데이터 처리 장치는 소스 정점과 연결된 기 설정된 개수 이내의 엣지를 포함하는 제1 윈도우를 설정할 수 있다. 그리고, 그래프 데이터 처리 장치는 설정된 제1 윈도우를 순차적으로 슬라이딩시키며, 제1 윈도우가 위치한 영역에 포함된 소스 정점 및 엣지 리스트에 기초하여 제1 도착 정점을 식별할 수 있다.

[0090] 그래프 데이터 처리 장치는 제1 도착 정점이 식별된 소스 정점에 기초하여 엣지 리스트를 로딩할 수 있다(S930). 그래프 데이터 처리 장치는 제1 도착 정점이 식별된 소스 정점과 연결된 제2 도착 정점을 식별하는 제2 레벨 과정을 수행할 수 있다(S940). 그래프 데이터 처리 장치는 제1 도착 정점이 식별된 소스 정점을 기초로 소스 정점과 연결된 엣지를 포함하는 제2 윈도우를 설정할 수 있다. 그리고, 그래프 데이터 처리 장치는 설정된 제2 윈도우를 순차적으로 슬라이딩시키며, 제2 윈도우가 위치한 영역에 포함된 제1 도착 정점이 식별된 소스 정점 및 엣지 리스트에 기초하여 제2 도착 정점을 식별할 수 있다.

[0091] 그래프 데이터 처리 장치는 소스 정점, 제1 도착 정점 및 제2 도착 정점에 기초하여 쿼리를 처리할 수 있다(S950).

[0092] 한편, 일 실시 예로서, 그래프 데이처 처리 장치는 삼각형 쿼리 처리 과정을 설명하였으나, 본 개시는 레벨을 추가하고 동일한 과정을 반복하여 사각형 또는 오각형과 같은 다각형 연결 관계에 대한 쿼리 작업도 수행할 수 있다. 즉, 그래프 데이터 처리 장치는 쿼리 작업이 요청된 다각형에 따라 제k 레벨까지 동일한 과정을 반복할 수 있다. 예를 들어, 그래프 데이터 처리 장치는 사각형 쿼리 처리 요청이 입력되는 경우, 제3 레벨 과정까지 수행할 수 있고(k=3), 오각형 쿼리 처리 요청이 입력되는 경우, 제4 레벨 과정까지 수행할 수 있다(k=4). 즉, k는 요청된 다각형의 꼭지점 수(n)보다 1 작은 수일 수 있다(k = n -1).

[0093] 상술한 다양한 실시 예에 따른 그래프 데이터 처리 방법은 컴퓨터 프로그램 제품으로 제공될 수도 있다. 컴퓨터 프로그램 제품은 S/W 프로그램 자체 또는 S/W 프로그램이 저장된 비일시적 판독 가능 매체(non-transitory computer readable medium)를 포함할 수 있다.

[0094] 비일시적 판독 가능 매체란 레지스터, 캐쉬, 메모리 등과 같이 짧은 순간 동안 데이터를 저장하는 매체가 아니라 반영구적으로 데이터를 저장하며, 기기에 의해 판독(reading)이 가능한 매체를 의미한다. 구체적으로는, 상술한 다양한 어플리케이션 또는 프로그램들은 CD, DVD, 하드 디스크, 블루레이 디스크, USB, 메모리카드, ROM 등과 같은 비일시적 판독 가능 매체에 저장되어 제공될 수 있다.

[0095] 또한, 이상에서는 본 발명의 바람직한 실시 예에 대하여 도시하고 설명하였지만, 본 발명은 상술한 특정의 실시 예에 한정되지 아니하며, 청구범위에서 청구하는 본 발명의 요지를 벗어남이 없이 당해 발명이 속하는 기술분야에서 통상의 지식을 가진 자에 의해 다양한 변형실시가 가능한 것은 물론이고, 이러한 변형실시들은 본 발명의 기술적 사상이나 전망으로부터 개별적으로 이해되어져서는 안될 것이다.

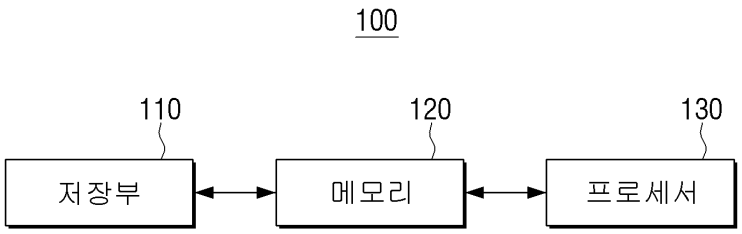
## 부호의 설명

[0096] 100: 그래프 데이터 처리 장치 110: 저장부

120: 메모리 130: 프로세서

## 도면

### 도면1



도면2a

**Algorithm 1:** PROCESSNWSM

**Input:** Level $l$, VertexStream $vs^l$, AdjStream $adjs^l$

$q_{new}$ = COMPUTEQVALUE(k)
**if** ($q_{new} > q$) **then**
  EXECUTEBBP($q_{new}$)
**end**
**if** ($l = 1$) **then**
  **spawn** GATHER();
**end**
// overlap scatter phase with gather phase
**while** $vw^l$ = READ ($vs^l$.NEXTWINDOWRANGE()) **do**
  **foreach** $j \in [1, pq/\lfloor \frac{q}{q_{new}} \rfloor]$ **do**
    LGB$^l$.CLEAR();
    **while** $adjw^l$ = ASYNCREAD ($adjs^l$.NEXTWINDOWRANGE(), GETRANGE($vw^l$), GETRANGE($j$), $adj\_scatter^l$) **do**
      **wait** until the above ASYNCREAD completes;
      **if** ($l \neq k$) **then**
        PROCESSNWSM ($l + 1$, VertexStream($voi^{l+1}$), AdjStream($voi^{l+1}$));
      **end**
    **end**
    ASYNCSEND(LGB$^l$)
  **end**
**end**
**if** ($l = 1$) **then**
  GLOBALBARRIER();
  **spawn** GATHERSPILLEDUPDATES(UpdateStream(updates in disk));
  **spawn** APPLY(VertexStream(all vertices in my machine));
**end**

도면2b

**Algorithm 2:** GATHER

**Input:** UpdateStream $us$

mid = GETMYMACHINEID();
**foreach** $Update\ u \in us$ **do**
  **if** $u.vid \in [\frac{|V|}{p}(mid-1), \frac{|V|}{p}(mid-1) + \frac{|V|}{p*q_{new}})$ **then**
    apply $u$ to the gather buffer
  **else**
    spill $u$ to disk with partitioning
  **end**
**end**

도면2c

**Algorithm 3:** GATHERSPILLEDUPDATES

**Input:** UpdateStream[] $us$

**foreach** *PartitionID* $pid \in [2, q_{new}]$ **do**
  **foreach** $Update\ u$ in $us[i]$ **do**
    apply $u$ to the gather buffer for the current $vw$
  **end**
**end**

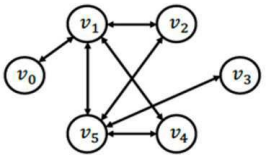
도면2d

**Algorithm 4:** APPLY

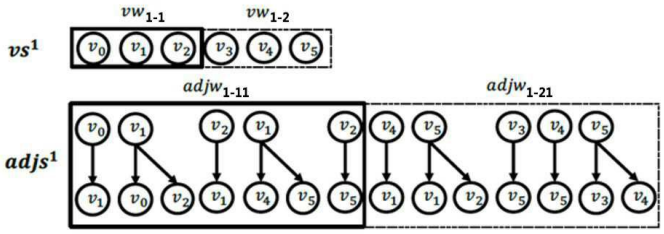
**Input:** VertexStream $vs$

**while** $vw$ = READ($vs$.NEXTWINDOWRANGE()) **do**
  **wait** until the gather task finishes merging spilled results for $vw$;
  **foreach** $v \in vw$ **do**
    $vertex\_apply(v)$;
  **end**
**end**

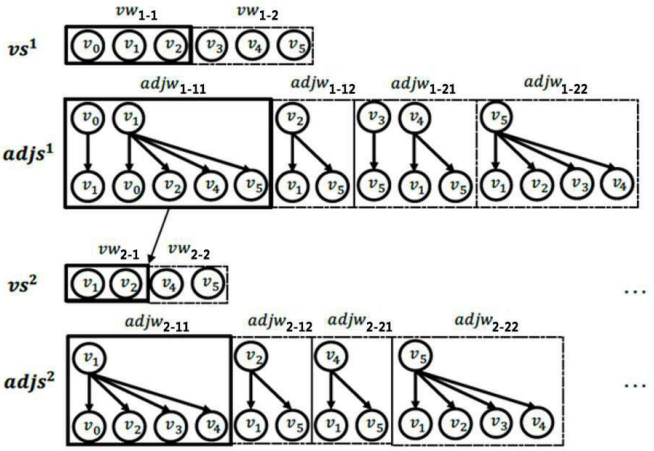
도면3



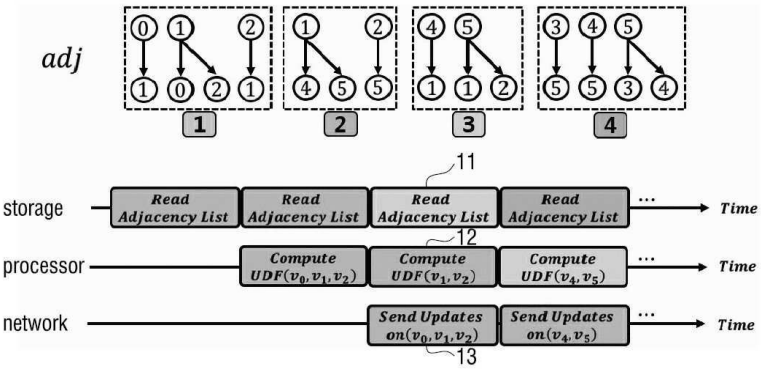
도면4



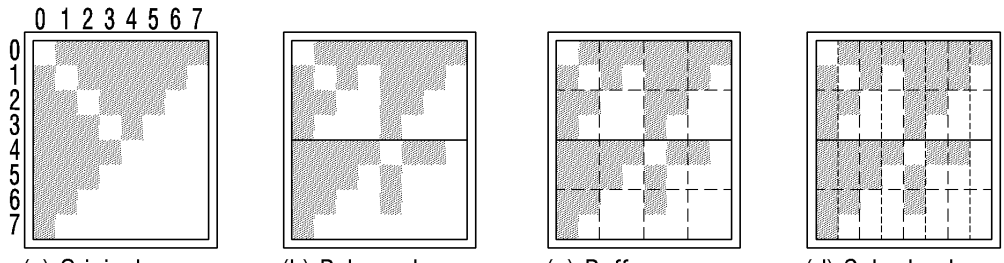

도면5




도면6

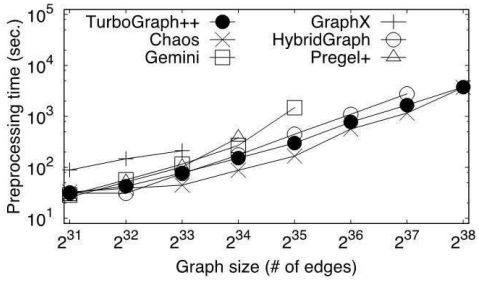



도면7

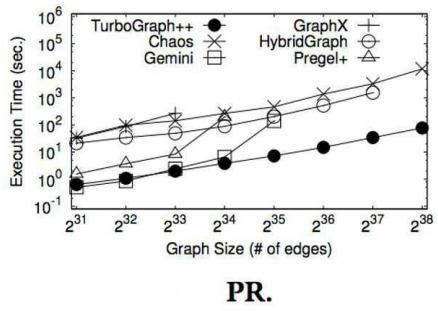



(a) Original graph (b) Balanced partitioning (c) Buffer-aware partitioning (d) Sub-chunk partitioning

도면8a

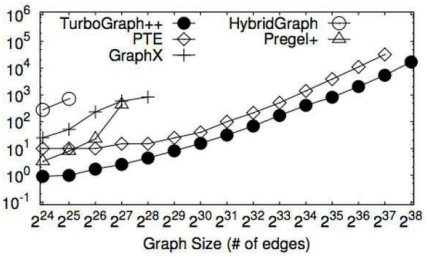



도면8b



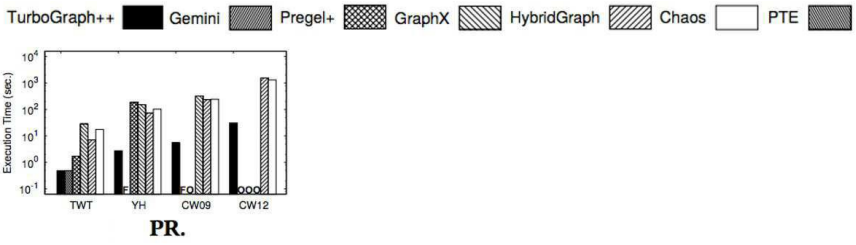

PR.

도면8c



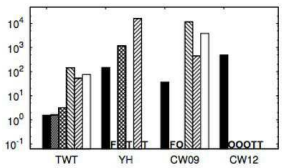

TC.

도면8d



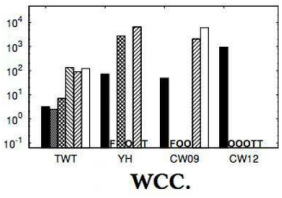

PR.

도면8e



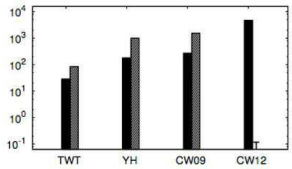

SSSP.

도면8f



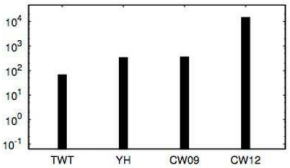
WCC.

도면8g



TC.

도면8h



LCC.

도면9