(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2024년03월29일
(11) 등록번호 10-2651832
(24) 등록일자 2024년03월22일

(51) 국제특허분류(Int. Cl.)
G06F 16/901 (2019.01) *G06F 16/2453* (2019.01)
G06F 16/904 (2019.01)
(52) CPC특허분류
G06F 16/9024 (2019.01)
G06F 16/24539 (2019.01)
(21) 출원번호 10-2021-0081010
(22) 출원일자 2021년06월22일
심사청구일자 2021년06월22일
(65) 공개번호 10-2022-0170226
(43) 공개일자 2022년12월29일
(56) 선행기술조사문헌
Seongyun Ko 외 6명, "iTurboGraph: Scaling and Automating Incremental Graph Analytics" (2021.06.09.)*
Makoto Onizuka 외 4명, "Optimization for iterative queries on MapReduce" (2013.12.01.)*
KR102020725 B1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
포항공과대학교 산학협력단
경상북도 포항시 남구 청암로 77 (지곡동)
(72) 발명자
한욱신
경상북도 포항시 남구 청암로 77(지곡동, 포항공과대학교)
고성윤
대구광역시 달서구 월배로32안길 35, 102동 1302호(진천동, 동백맨션)
이태성
경상북도 포항시 남구 청암로 77, 기숙사10동 206호(지곡동, 포항공과대학교)
(74) 대리인
김태헌

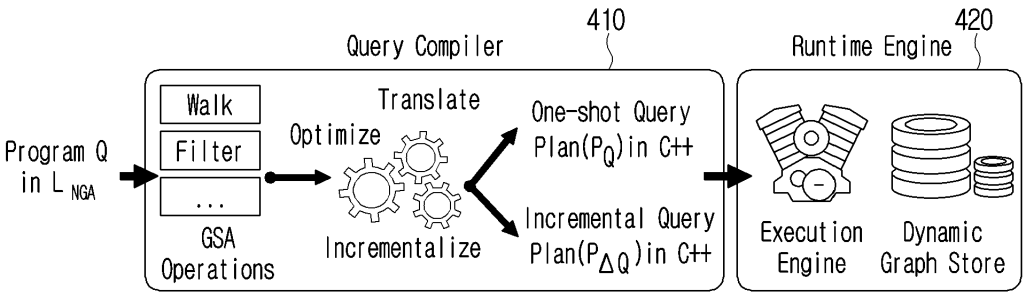
전체 청구항 수 : 총 15 항 심사관 : 안지현

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

(57) 요 약

전자 장치가 개시된다. 본 전자 장치는 복수의 정점과 엣지를 포함하는 그래프 데이터와 그래프 데이터에 대한 변경된 증분 정보를 저장하는 메모리, 그래프 데이터에 대한 쿼리를 수신하는 통신 장치, 및 수신한 쿼리에 기초하여, 쿼리에 대응되는 출력 데이터를 생성하고, 출력 데이터가 쿼리를 전송한 장치에 전송하도록 통신 장치를 제어하는 프로세서를 포함한다.

대 표 도 - 도4



(52) CPC특허분류
G06F 16/24542 (2019.01)
G06F 16/24547 (2019.01)
G06F 16/904 (2019.01)

공지예외적용 : 있음

명 세 서

청구범위

청구항 1

복수의 정점과 엣지를 포함하는 그래프 데이터와 상기 그래프 데이터에 대한 변경된 증분 정보를 저장하는 메모리;

상기 그래프 데이터에 대한 쿼리를 수신하는 통신 장치; 및

상기 수신한 쿼리에 기초하여, 상기 쿼리에 대응되는 출력 데이터를 생성하고, 상기 출력 데이터가 상기 쿼리를 전송한 장치에 전송하도록 상기 통신 장치를 제어하는 프로세서;를 포함하고,

상기 프로세서는,

상기 수신한 쿼리에 기초하여 상기 그래프 데이터에 적용할 원샷 쿼리 플랜 정보와 상기 증분 정보에 적용할 증분 쿼리 플랜 정보를 생성하고, 상기 생성한 원샷 쿼리 플랜 정보를 상기 그래프 데이터에 반영하여 제1 중간 데이터를 생성하고, 상기 생성한 증분 쿼리 플랜 정보를 상기 증분 정보에 반영하여 제2 중간 데이터를 생성하고, 상기 제1 중간 데이터 및 상기 제2 중간 데이터를 이용하여 결과 데이터를 생성하는 전자 장치.

청구항 2

제1항에 있어서,

상기 그래프 데이터는,

정점의 ID와 속성 값을 갖는 정점 스트림과 소스 및 대상 정점들의 ID를 갖는 에지 스트림의 쌍으로 정의되는 그래프 스트림인 전자 장치.

청구항 3

제2항에 있어서,

상기 프로세서는,

기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 로드하고, 로드된 정점 스트림 및 에지 스트림에 대응되는 상기 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 내의 워크를 일괄적으로 수행하는 전자 장치.

청구항 4

제1항에 있어서,

상기 수신한 쿼리는,

그래프 처리에 대한 다수의 처리 명령을 포함하는 기설정된 프로그램 언어로된 프로그램이며,

상기 프로세서는,

상기 프로그램을 컴파일링하여 원샷 쿼리 플랜 정보를 생성하고, 상기 생성된 원샷 쿼리 플랜 정보를 기설정된 증분 처리 규칙에 적용하여 상기 증분 쿼리 플랜 정보를 생성하는 전자 장치.

청구항 5

제4항에 있어서,

상기 프로세서는,

상기 생성한 원샷 쿼리 플랜 정보 및 상기 증분 쿼리 플랜 정보 각각에 대해서 기설정된 최적화 방식을 적용하여 최적화된 원샷 쿼리 플랜 정보 및 최적화된 증분 쿼리 플랜 정보를 수정하고,

상기 최적화 방식은,

트래버셜 리오더링(TR), 이웃 가지치기(NP), 시크/윈도우 쉐어링(SWS), CNT(Min with counting) 및 비용 기반 델타 유지 관리(COST) 중 적어도 하나인 전자 장치.

청구항 6

제4항에 있어서,

상기 기설정된 프로그램 언어는,

제약 조건이 있는 루프에 대한 중첩이 기설정된 홉 거리(k-홉 거리)의 이웃노드로 단순화하여 표현 가능한 프로그램 언어인 전자 장치.

청구항 7

제1항에 있어서,

상기 증분 정보는,

상기 그래프 데이터에 대한 정점 및/또는 엣지에 대한 그래프 변경 정보 및 상기 그래프 데이터에 포함된 복수의 정점 중 속성값이 변환한 정점의 속성값 변경 정보를 포함하는 전자 장치.

청구항 8

제1항에 있어서,

상기 프로세서는,

상기 증분 정보의 크기가 기설정된 크기 이상이면, 상기 증분 정보를 상기 그래프 데이터에 반영하여 그래프 데이터를 업데이트하는 전자 장치.

청구항 9

복수의 정점과 엣지를 포함하는 그래프 데이터와 상기 그래프 데이터에 대한 변경된 증분 정보를 저장하는 단계;

상기 그래프 데이터에 대한 쿼리를 수신하면, 상기 쿼리에 대응되는 출력 데이터를 생성하는 단계; 및

상기 출력 데이터가 상기 쿼리를 전송한 장치에 전송하는 단계;를 포함하고,

상기 출력 데이터를 생성하는 단계는,

상기 수신한 쿼리에 기초하여 상기 그래프 데이터에 적용할 원샷 쿼리 플랜 정보와 상기 증분 정보에 적용할 증분 쿼리 플랜 정보를 생성하고, 상기 생성한 원샷 쿼리 플랜 정보를 상기 그래프 데이터에 반영하여 제1 중간 데이터를 생성하고, 상기 생성한 증분 쿼리 플랜 정보를 상기 증분 정보에 반영하여 제2 중간 데이터를 생성하고, 상기 제1 중간 데이터 및 상기 제2 중간 데이터를 이용하여 결과 데이터를 생성하는 그래프 데이터 처리 방법.

청구항 10

제9항에 있어서,

상기 그래프 데이터는,

정점의 ID와 속성 값을 갖는 정점 스트림과 소스 및 대상 정점들의 ID를 갖는 에지 스트림의 쌍으로 정의되는 그래프 스트림인 그래프 데이터 처리 방법.

청구항 11

제10항에 있어서,

상기 출력 데이터를 생성하는 단계는,

기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 로드하고, 로드된 정점 스트림 및 에지 스트림에 대응되는 상기 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 내의 워크를 일괄적으로 수행하는 그래프 데

이터 처리 방법.

청구항 12

제9항에 있어서,

상기 수신한 쿼리는,

그래프 처리에 대한 다수의 처리 명령을 포함하는 기설정된 프로그램 언어로된 프로그램이며,

상기 출력 데이터를 생성하는 단계는,

상기 프로그램을 컴파일링하여 원샷 쿼리 플랜 정보를 생성하고, 상기 생성된 원샷 쿼리 플랜 정보를 기설정된 증분 처리 규칙에 적용하여 상기 증분 쿼리 플랜 정보를 생성하는 그래프 데이터 처리 방법.

청구항 13

제12항에 있어서,

상기 출력 데이터를 생성하는 단계는,

상기 생성한 원샷 쿼리 플랜 정보 및 상기 증분 쿼리 플랜 정보 각각에 대해서 기설정된 최적화 방식을 적용하여 최적화된 원샷 쿼리 플랜 정보 및 최적화된 증분 쿼리 플랜 정보를 수정하고,

상기 최적화 방식은,

트래버셜 리오더링(TR), 이웃 가지치기(NP), 시크/윈도우 쉐어링(SWS), CNT(Min with counting) 및 비용 기반 델타 유지 관리(COST) 중 적어도 하나인 그래프 데이터 처리 방법.

청구항 14

제9항에 있어서,

상기 증분 정보는,

상기 그래프 데이터에 대한 정점 및/또는 엣지에 대한 그래프 변경 정보 및 상기 그래프 데이터에 포함된 복수의 정점 중 속성값이 변환한 정점의 속성값 변경 정보를 포함하는 그래프 데이터 처리 방법.

청구항 15

제9항에 있어서,

상기 증분 정보의 크기가 기설정된 크기 이상이면, 상기 증분 정보를 상기 그래프 데이터에 반영하여 그래프 데이터를 업데이트하는 단계;를 더 포함하는 그래프 데이터 처리 방법.

발명의 설명

기 술 분 야

[0001] 본 개시는 그래프 데이터 처리 방법 및 그래프 데이터 처리 장치에 관한 것으로, 더욱 상세하게는 그래프 데이터를 증분 데이터로 분리하여 저장하고, 입력된 쿼리에 대해서 증분 데이터에 적용되는 증분 쿼리를 생성하여 분석을 수행할 수 있는 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.

[0003] 기존의 그래프 분석 시스템은 정적 그래프에 대한 일회성 분석에 초점을 맞추고 있기 때문에, 그래프가 변경될 때마다 분석을 재실행하여야 한다. 이에 따라 데이터가 변환하는 동적 그래프 데이터에 대한 처리시에 많은 오버헤드가 발생한다.

[0004] 이러한 점에서, 최근에는 그래프 데이터의 변경되는 경우에 전체 데이터에 대한 재실행을 방지하기 위하여, 증분 그래프 분석 방법을 이용하였다. 그러나 동적 그래프 저장을 지원하는 시스템에 대한 연구는 있었지만, 이들은 증분 그래프 분석 방법을 이용한 질의 처리는 지원하지 않았었다. 구체적으로, 기존 방식은 이전 계산의 중간 결과를 유지하여야 하기 때문에, 중간 결과들의 높은 리소스로 인해 효율성 낮은 문제가 있었다.

발명의 내용

해결하려는 과제

[0005] 본 개시는 이상과 같은 문제점을 해결하기 위하여 고안된 것으로, 본 개시의 목적은 그래프 데이터를 증분 데이터로 분리하여 저장하고, 입력된 쿼리에 대해서 증분 데이터에 적용되는 증분 쿼리를 생성하여 분석을 수행할 수 있는 그래프 데이터 처리 방법 및 그래프 데이터 처리 장치를 제공하는 데 있다.

과제의 해결 수단

[0006] 이상과 같은 목적을 달성하기 위하여, 본 개시의 일 실시 예에 따른 전자 장치는, 복수의 정점과 엣지를 포함하는 그래프 데이터와 상기 그래프 데이터에 대한 변경된 증분 정보를 저장하는 메모리, 상기 그래프 데이터에 대한 쿼리를 수신하는 통신 장치, 및 상기 수신한 쿼리에 기초하여, 상기 쿼리에 대응되는 출력 데이터를 생성하고, 상기 출력 데이터가 상기 쿼리를 전송한 장치에 전송하도록 상기 통신 장치를 제어하는 프로세서를 포함하고, 상기 프로세서는 상기 수신한 쿼리에 기초하여 상기 그래프 데이터에 적용할 원샷 쿼리 플랜 정보와 상기 증분 정보에 적용할 증분 쿼리 플랜 정보를 생성하고, 상기 생성한 원샷 쿼리 플랜 정보를 상기 그래프 데이터에 반영하여 제1 중간 데이터를 생성하고, 상기 생성한 증분 쿼리 플랜 정보를 상기 증분 정보에 반영하여 제2 중간 데이터를 생성하고, 상기 제1 중간 데이터 및 상기 제2 중간 데이터를 이용하여 결과 데이터를 생성한다.

[0007] 이 경우, 상기 그래프 데이터는 정점의 ID와 속성 값을 갖는 정점 스트림과 소스 및 대상 정점들의 ID를 갖는 에지 스트림의 쌍으로 정의되는 그래프 스트림일 수 있다.

[0008] 이 경우, 상기 프로세서는 기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 로드하고, 로드된 정점 스트림 및 에지 스트림에 대응되는 상기 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 내의 워크를 일괄적으로 수행할 수 있다.

[0009] 한편, 상기 수신한 쿼리는 상기 그래프 처리에 대한 다수의 처리 명령을 포함하는 기설정된 프로그램 언어로 된 프로그램이며, 상기 프로세서는 상기 프로그램을 컴파일링하여 원샷 쿼리 플랜 정보를 생성하고, 상기 생성된 원샷 쿼리 플랜 정보를 기설정된 증분 처리 규칙에 적용하여 상기 증분 쿼리 플랜 정보를 생성할 수 있다.

[0010] 여기서, 상기 프로세서는 상기 생성한 원샷 쿼리 플랜 정보 및 상기 증분 쿼리 플랜 정보 각각에 대해서 기설정된 최적화 방식을 적용하여 최적화된 원샷 쿼리 플랜 정보 및 최적화된 증분 쿼리 플랜 정보를 수정할 수 있다.

[0011] 한편, 상기 기설정된 프로그램 언어는 제약 조건이 있는 루프에 대한 중첩이 기설정된 홉 거리(k-홉 거리)의 이웃노드로 단순화하여 표현 가능한 프로그램 언어일 수 있다.

[0012] 한편, 상기 증분 정보는 상기 그래프 데이터에 대한 정점 및/또는 엣지에 대한 그래프 변경 정보 및 상기 그래프 데이터에 포함된 복수의 정점 중 속성값이 변환한 정점의 속성값 변경 정보를 포함할 수 있다.

[0013] 한편, 상기 프로세서는 상기 증분 정보의 크기가 기설정된 크기 이상이면, 상기 증분 정보를 상기 그래프 데이터에 반영하여 그래프 데이터를 업데이트할 수 있다.

[0014] 한편, 본 개시의 일 실시 예에 따른 그래프 데이터 처리 방법은 복수의 정점과 엣지를 포함하는 그래프 데이터와 상기 그래프 데이터에 대한 변경된 증분 정보를 저장하는 단계, 상기 그래프 데이터에 대한 쿼리를 수신하면, 상기 쿼리에 대응되는 출력 데이터를 생성하는 단계, 및 상기 출력 데이터가 상기 쿼리를 전송한 장치에 전송하는 단계를 포함하고, 상기 출력 데이터를 생성하는 단계는 상기 수신한 쿼리에 기초하여 상기 그래프 데이터에 적용할 원샷 쿼리 플랜 정보와 상기 증분 정보에 적용할 증분 쿼리 플랜 정보를 생성하고, 상기 생성한 원샷 쿼리 플랜 정보를 상기 그래프 데이터에 반영하여 제1 중간 데이터를 생성하고, 상기 생성한 증분 쿼리 플랜 정보를 상기 증분 정보에 반영하여 제2 중간 데이터를 생성하고, 상기 제1 중간 데이터 및 상기 제2 중간 데이터를 이용하여 결과 데이터를 생성할 수 있다.

[0015] 이 경우, 상기 그래프 데이터는 정점의 ID와 속성 값을 갖는 정점 스트림과 소스 및 대상 정점들의 ID를 갖는

에지 스트림의 쌍으로 정의되는 그래프 스트림일 수 있다.

[0016] 이 경우, 상기 출력 데이터를 생성하는 단계는 기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 로드하고, 로드된 정점 스트림 및 에지 스트림에 대응되는 상기 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 내의 워크를 일괄적으로 수행할 수 있다.

[0017] 기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 확인하고, 상기 확인된 영역에 기초하여 상기 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보를 생성할 수 있다.

[0018] 한편, 상기 수신한 쿼리는 상기 그래프 처리에 대한 다수의 처리 명령을 포함하는 기설정된 프로그램 언어로 된 프로그램이며, 상기 출력 데이터를 생성하는 단계는 상기 프로그램을 컴파일링하여 원샷 쿼리 플랜 정보를 생성하고, 상기 생성된 원샷 쿼리 플랜 정보를 기설정된 증분 처리 규칙에 적용하여 상기 증분 쿼리 플랜 정보를 생성할 수 있다.

[0019] 이 경우, 상기 출력 데이터를 생성하는 단계는, 상기 생성한 원샷 쿼리 플랜 정보 및 상기 증분 쿼리 플랜 정보 각각에 대해서 기설정된 최적화 방식을 적용하여 최적화된 원샷 쿼리 플랜 정보 및 최적화된 증분 쿼리 플랜 정보를 수정할 수 있다.

[0020] 한편, 상기 증분 정보는 상기 그래프 데이터에 대한 정점 및/또는 엣지에 대한 그래프 변경 정보 및 상기 그래프 데이터에 포함된 복수의 정점 중 속성값이 변환한 정점의 속성값 변경 정보를 포함할 수 있다.

[0021] 한편, 본 그래프 데이터 처리 방법은 상기 증분 정보의 크기가 기설정된 크기 이상이면, 상기 증분 정보를 상기 그래프 데이터에 반영하여 그래프 데이터를 업데이트하는 단계를 더 포함할 수 있다.

발명의 효과

[0022] 이상과 같은 본 개시의 다양한 실시 예들에 따르면, 본 개시에서 제안한 그래프 분석 방법은 데이터 증분에 따라 중간 데이터를 저장하지 않는바, 데이터 입출력 측면에서 유리하면, 증분 쿼리의 경우 기존보다 빠른 실행이 가능하다.

도면의 간단한 설명

[0023] 도 1은 본 개시의 일 실시 예에 따른 전자 장치의 구성을 도시한 블록도,

도 2 및 도 3은 소셜 네트워크 예를 설명하기 위한 도면,

도 4는 본 개시의 일 실시 예에 따른 그래프 데이터에 대한 분석 시스템의 구성을 설명하기 위한 도면,

도 5는 일 실시 예에 따른 데이터그래프의 예를 도시한 도면,

도 6은 도 5의 데이터 그래프에 대한 그래프 스트림의 예를 도시한 도면,

도 7은 데이터그래프의 삼각형 카운팅을 위한 중첩 그래프 윈도우의 예를 도시한 도면,

도 8은 3-홉 순회를 위해 3 개의 그래프 스트림(gs_1, gs_2, gs_3)을 취하는 W_{ALK} 연산자의 예를 도시한 도면,

도 9는 페이지링크의 트래버스의 컴파일링의 예를 도시한 도면,

도 10은 본 개시의 일 실시 예에 따른 GSA 증분 규칙을 설명하기 위한 도면,

도 11은 본 개시의 일 실시 예에 따른 일회성 쿼리와 증분성 쿼리를 처리하는 동작을 설명하기 위한 도면,

도 12 및 도 13은 본 개시의 일 실시 예에 따른 증분 그래프 분석의 실행 예를 설명하기 위한 도면,

도 14는 본 개시의 일 실시 예에 따른 재정렬된 트래버스의 예를 도시한 도면,

도 15는 다양한 그래프 데이터 처리 방법에 따른 처리 시간을 설명하기 위한 도면,

도 16은 그래프 크기에 따른 처리 시간의 변화를 설명하기 위한 도면,

도 17은 머신 수에 따른 처리 시간의 변화를 설명하기 위한 도면,

도 18은 다양한 부하량에 따른 실행 시간과 스루풋을 설명하기 위한 도면,

도 19는 최적화의 변화에 따른 실행 시간의 차이를 설명하기 위한 도면,

도 20은 증분 쿼리의 실행 시간과 최적화 후의 실행 시간을 설명하기 위한 도면, 그리고,

도 21은 본 개시의 일 실시 예에 따른 그래프 데이터 처리 방법을 설명하기 위한 흐름도이다.

발명을 실시하기 위한 구체적인 내용

[0024] 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.

[0025] 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.

[0026] 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 않은 것으로 이해되어야 할 것이다.

[0027] 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.

[0028] 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.

[0029] 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.

[0030] 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.

[0031] 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.

[0032] 도 1은 본 개시의 일 실시 예에 따른 전자 장치의 구성을 도시한 블록도이다.

[0033] 도 1을 참조하면, 전자 장치(100)는 통신 장치(110), 메모리(120), 디스플레이(130), 조작 입력 장치(140) 및 프로세서(150)를 포함할 수 있다.

[0034] 통신 장치(110)는 전자 장치(100)를 외부 장치(미도시)와 연결하기 위해 형성되고, 근거리 통신망(LAN: Local

Area Network) 및 인터넷망을 통해 외부 장치에 접속되는 형태뿐만 아니라, USB(Universal Serial Bus) 포트 또는 무선 통신(예를 들어, WiFi 802.11a/b/g/n, NFC, Bluetooth) 포트를 통하여 접속되는 형태도 가능하다. 이러한 통신 장치(110)는 송수신부(transceiver)로 지칭될 수도 있다.

[0035] 통신 장치(110)는 그래프 데이터를 외부 장치로부터 수신할 수 있으며, 그래프 데이터에 대한 쿼리를 요청받을 수 있다.

[0036] 그리고 통신 장치(110)는 요청된 쿼리에 대응되는 결과 데이터를 쿼리를 요청한 장치에 전송할 수 있다.

[0037] 메모리(120)는 전자 장치(100)를 구동하기 위한 O/S나 각종 소프트웨어, 데이터 등을 저장하기 위한 구성원소이다. 메모리(120)는 RAM이나 ROM, 플래시 메모리, HDD, 외장 메모리, 메모리 카드 등과 같은 다양한 형태로 구현될 수 있으며, 어느 하나로 한정되는 것은 아니다.

[0038] 메모리(120)는 그래프 데이터 및 그에 대한 증분 데이터를 저장할 수 있다. 그래프 데이터는 복수의 정점과 엣지를 포함하는 그래프에 대한 정보이고, 증분 정보는 그래프 데이터에 대한 변경된 증분 정보이다. 보다 상세하게는 그래프 데이터는, 정점의 ID와 속성 값을 갖는 정점 스트림과 소스 및 대상 정점들의 ID를 갖는 에지 스트림의 쌍으로 정의되는 그래프 스트림 정보이고, 증분 정보는 그래프 데이터에 대한 정점 및/또는 엣지에 대한 그래프 변경 정보 및 그래프 데이터에 포함된 복수의 정점 중 속성값이 변환한 정점의 속성값 변경 정보이다.

[0039] 이때, 메모리(120)에 저장되는 그래프 데이터는 그래프 데이터 전체일 수 있고, 대규모 그래프 데이터인 경우 대규모 그래프 데이터의 일부 그래프 데이터를 저장할 수도 있다.

[0040] 메모리(120)는 입력된 쿼리를 컴파일링 하기 위한 프로그램을 저장할 수 있으며, 입력된 쿼리에 대한 컴파일링 결과를 저장할 수 있다.

[0041] 디스플레이(130)는 전자 장치(100)가 지원하는 기능을 선택받기 위한 사용자 인터페이스 창을 표시한다. 구체적으로, 디스플레이(130)는 전자 장치(100)가 제공하는 각종 기능을 선택받기 위한 사용자 인터페이스 창을 표시할 수 있다. 이러한 디스플레이(130)는 LCD(liquid crystal display), OLED(Organic Light Emitting Diodes) 등과 같은 모니터일 수 있으며, 후술할 조작 입력 장치(140)의 기능을 동시에 수행할 수 있는 터치 스크린으로 구현될 수도 있다.

[0042] 디스플레이(130)는 처리 결과를 표시할 수 있다. 그리고 디스플레이(130)는 컴파일링 결과 등 중간 결과를 표시할 수도 있다.

[0043] 조작 입력 장치(140)는 사용자로부터 전자 장치(100)의 기능 선택 및 해당 기능에 대한 제어 명령을 입력받을 수 있다. 구체적으로, 조작 입력 장치(140)는 사용자로부터 그래프 처리 방식에 적용되는 다양한 인자를 입력받을 수 있다. 여기서 입력받을 수 있는 인자는 쿼리 플랜의 생성 과정에 이용될 최적화 방식, 증분 정보의 그래프 데이터로의 병합 등일 수 있다.

[0044] 프로세서(150)는 전자 장치(100) 내의 각 구성을 제어한다. 이러한 프로세서(150)는 CPU(central processing unit), ASIC(application-specific integrated circuit)과 같은 단일 장치로 구성될 수 있으며, CPU, GPU(Graphics Processing Unit) 등의 복수의 장치로 구성될 수도 있다.

[0045] 프로세서(150)는 그래프 데이터에 대한 변경 사항이 확인되면, 그에 대한 증분 데이터를 생성하여 메모리(120)에 저장할 수 있다. 이때, 프로세서(150)는 스트리밍 형태로 그래프 데이터와 증분 정보를 메모리(120)에 저장할 수 있다.

[0046] 이와 같은 동작을 위하여, 프로세서(150)는 소스 정점과 연결된 연결 관계를 식별할 수 있다. 즉, 프로세서(150)는 기 설정된 개수의 그래프 데이터의 소스 정점, 소스 정점에 기초한 엣지 리스트를 로딩하여 각각이 소스 정점과 연결된 제1 도착 정점을 식별할 수 있다.

[0047] 예를 들어, 프로세서(150)가 삼각형의 연결 관계를 식별하는 쿼리를 수행하는 경우, 프로세서(150)는 상술한 과정을 통해 제1 도착 정점을 식별할 수 있다. 그리고 프로세서(150)는 식별된 제1 도착 정점에 기초하여 엣지 리스트를 로딩할 수 있다. 프로세서(150)는 식별된 제1 도착 정점 및 로딩된 엣지 리스트에 기초하여 제2 도착 정점을 식별할 수 있다. 프로세서(150)는 소스 정점, 식별된 제1 도착 정점 및 식별된 제2 도착 정점에 기초하여 삼각형의 그래프를 식별하는 쿼리를 수행할 수 있다. 만일, 프로세서(150)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.

[0048] 그리고 프로세서(150)는 증분 정보가 크기가 기설정된 크기 이상이면, 즉 증분 정보를 이용한 그래프 처리 비용이 그래프 데이터만을 이용하는 것보다 커지면, 증분 정보를 그래프 데이터에 반영하는 업데이트를 수행할 수 있다.

[0049] 그리고 프로세서(150)는 그래프 데이터에 대한 쿼리를 수신하면, 수신된 쿼리에 대응되는 출력 데이터를 생성할 수 있다. 그리고 프로세서(150)는 해당 출력 데이터가 쿼리를 요청한 장치에 전송하도록 통신 장치(110)를 제어할 수 있다.

[0050] 구체적으로, 프로세서(150)는 수신한 쿼리에 기초하여 그래프 데이터에 적용할 원샷 쿼리 플랜 정보와 증분 정보에 적용할 쿼리 정보를 생성할 수 있다. 이때, 프로세서(150)는 기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 확인하고, 확인된 영역에 기초하여 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보를 생성할 수 있다.

[0051] 이때, 수신한 쿼리는 그래프 처리에 대한 다수의 처리 명령을 포함하는 기설정된 프로그램 언어로 된 프로그램일 수 있다. 여기서, 기설정된 프로그램 언어는 제약 조건이 있는 루프에 대한 중첩이 기설정된 홉 거리(k-홉 거리)의 이웃노드로 단순화하여 표현 가능한 프로그램 언어일 수 있다. 이러한 프로그램이 입력되면, 프로세서(150)는 프로그램을 컴파일링하여 원샷 쿼리 플랜 정보를 생성하고, 생성된 원샷 쿼리 플랜 정보를 기설정된 증분 처리 규칙에 적용하여 증분 쿼리 플랜 정보를 생성할 수 있다.

[0052] 그리고 규칙 정보가 생성되면, 프로세서(150)는 추가적으로 생성한 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 각각에 대해서 기설정된 최적화 방식을 적용하여 최적화된 원샷 쿼리 플랜 정보 및 최적화된 증분 쿼리 플랜 정보를 수정할 수 있다.

[0053] 그리고 프로세서(150)는 생성한 원샷 쿼리 플랜 정보를 그래프 데이터에 반영하여 제1 중간 데이터를 생성하고, 생성한 증분 쿼리 플랜 정보를 증분 정보에 반영하여 제2 중간 데이터를 생성할 수 있다. 이때, 프로세서(150)는 기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 로드하고, 로드된 정점 스트림 및 에지 스트림에 대응되는 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 내의 워크를 일괄적으로 수행할 수 있다.

[0054] 그리고 프로세서(150)는 제1 중간 데이터 및 제2 중간 데이터를 이용하여 결과 데이터를 생성할 수 있다.

[0055] 이상과 같이 본 개시에 따른 전자 장치는 변경된 데이터, 즉 증분 데이터를 별로로 관리하는바, 그래프 데이터를 효율적으로 관리할 수 있다. 또한, 이러한 환경에서의 쿼리가 입력된 경우에도, 입력된 쿼리를 원샷 쿼리와 증분 쿼리로 구분하고, 구분된 두 쿼리를 이용하여 그래프 데이터에 대한 처리를 수행할 수 있는바, 보다 빠른 검색이 가능하다. 또한, 그 과정에서 중간 데이터를 저장할 필요가 없는바 효율적인 리소스 관리가 가능하다.

[0057] 이하에서는 본 개시에 따른 쿼리 증분화 동작을 설명한다.

[0058] 쿼리 증분화는 수동 방식과 자동 방식으로 구분할 수 있다. 수동 방식은 사용자가 쿼리를 도출하여 사용하는 방식이나, 오류가 발생하기 쉬우면 정확성에 대해 추론하기 어려운 점이 있다. 따라서, 본 개시에서는 자동 방식으로 쿼리 증분화를 수행하는 것을 설명한다.

[0059] 그리고 NGA(Neighbor-centric graph analytics)를 대상으로 처리하는 것을 고려한다. 여기서, NGA는 사용자가 각 정점 주변의 다중 홉 이웃들에 대한 계산을 프로그래밍하는 것으로, 정점 중심 그래프 분석 방식의 일반화된 방식이다. 이하에서는 도 2 및 도 3을 참조하여, NGA에 대해서 설명한다.

[0061] 도 2는 소셜 네트워크 예를 설명하기 위한 도면이다.

[0062] 도 2를 참조하면, 도 2(a)는 소셜 네트워크에서 제1 사용자(Bob)에 대한 1-홉 거리 네트워크를 나타낸다. 정점 중심 분석은 순위를 위한 페이지랭크(PageRank) 및 연결된 구성 찾기와 같은 각 정점 주변의 1-홉 거리의 이웃 노드에 대한 계산을 지원한다.

[0063] NGA는 각 정점 주변의 로컬 구조를 조사하는데 사용할 수 있다. 예를 들어, 소셜 그래프의 삼각형은 친구 간의 응집력 있는 관계를 나타낸다. 이웃한 친구들은 서로 친구일 경우가 높다. 이러한 로컬 구조는 기본 구성 요소이며, 하향 분석 작업에 사용될 수 있다. 예를 들어, 로컬 클러스터링 계수는 각 정점 주변의 삼각형 수를 계산하고, 이웃의 연결성을 평가할 수 있다.

[0064] 도 2(a)와 같이 1-홉 거리의 이웃 노드만을 고려하면, 네트워크의 특성을 평가하기 어렵다. 반면에, 도 2(b)와 같이 제1 사용자(Bob)의 친구들(즉, 2- 홉 거리) 간의 연결을 추가로 나타내면, 네트워크의 특성을 평가할 수

있다.

[0066] 도 3은 로컬 클러스터링 계수를 사용한 커뮤니티 탐지 결과를 나타낸다.

[0067] 도 3을 참조하면, 탐색된 커뮤니티의 응집된 구조를 보여준다. 이와 같이 로컬 클러스터링 계수를 사용함으로써, 피드 추천 및 링크 예측에 탐지 결과를 사용할 수 있다.

[0068] 이와 같은 분석을 위한 본 개시에 따른 그래프 처리 시스템을 도 4를 참조하여 설명한다.

[0070] 도 4는 본 개시의 일 실시 예에 따른 그래프 데이터에 대한 분석 시스템의 구성을 설명하기 위한 도면이다.

[0071] 도 4를 참조하면, 본 개시에 따른 분석 시스템은 쿼리 컴파일러(410) 및 분산 런타임 엔진(420)을 포함할 수 있다. 본 개시에 따른 분석 시스템은 도 1과 같은 하나의 전자 장치로 구성될 수 있으며, 분석 시스템 내의 각 구성이 별도의 전자 장치로 구성될 수도 있다. 또한, 하나의 구성 또한, 복수의 전자 장치가 연동하여 수행할 수도 있다. 이하에서는 설명을 용이하게 하기 위하여, 쿼리 컴파일러와 분산 런타임 엔진이 서로 다른 하나의 전자 장치에서 수행하는 것으로 가정하여 설명한다.

[0072] 쿼리 컴파일러(410)는 L_{NGA}로 된 분석 프로그램(Q)을 입력받을 수 있다. 여기서 L_{NGA}는 NGA 프로그래밍의 어려움을 쉽게 하고자 설계된 프로그램 언어로, 제약 조건이 있는 루프에 대한 중첩은 K-홉 거리의 이웃 노드로 단순화된 언어이다. 예를 들어, 삼각형 카운팅 알고리즘은, 기존의 프로그램은 세 가지 슈퍼스텝에 걸쳐 서로 다른 두 종류의 메시지를 인코딩 및 디코딩하여야 하였다. 그러나 본 개시에 따른 L_{NGA}에서 3-홉 거리 순회는 중첩된 반복들로 직관적으로 표현된다.

[0073] 그리고 쿼리 컴파일러(410)는 분석 프로그램(Q)이 입력되면, 쿼리 계획 정보를 생성할 수 있다. 여기서, 쿼리 계획 정보는 원샷 쿼리 계획(One-shot Query Plan)(P_Q) 정보와 증분 쿼리 계획(incremental query plans)($P_{\Delta Q}$) 정보를 포함할 수 있다.

[0074] 구체적으로, 쿼리 컴파일러(410)는 먼저, GSA 연산자들을 원샷 쿼리 계획 정보로 컴파일링할 수 있다. 그리고 컴파일 이후에, 쿼리 컴파일러(410)는 쿼리 증분화 규칙에 원샷 쿼리 계획(P_Q)을 적용하여, 증분 쿼리 계획($P_{\Delta Q}$) 정보를 생성할 수 있다. 이에 대한 동작은 자세한 설명이 필요한바, 도 8을 참조하여 후술한다.

[0075] 그리고 쿼리 컴파일러(410)는 생성한 계획 정보를 최적화하여 업데이트할 수 있다. 여기서 최적화 방식으로 TR, NP, SWS, C_{NT}, C_{OST} 등이 이용될 수 있다. 각 최적화 방식에 대한 자세한 내용은 후술한다.

[0076] 분산 런타임 엔진(420)은 쿼리 컴파일러(410)가 출력한 쿼리 계획 정보(P_Q, $P_{\Delta Q}$)에 기초하여 그래프 분석을 수행할 수 있다. 스케줄링은 그래프 처리의 효율성에 큰 영향을 미친다. 예를 들어, 증분 분석은 여러 하위 쿼리로 구성되지만, 이들을 개별적으로 처리하면 반복되는 입출력으로 성능이 저하될 수 있다. 따라서, 분산 런타임 엔진(420)은 반복되는 데이터 입출력을 방지하기 위하여, 하위 쿼리들이 동시에 수행되도록 스케줄링할 수 있다.

[0077] 그리고 분산 런타임 엔진(420)은 효율적인 쿼리 실행을 위하여 동적 그래프를 저장할 수 있다.

[0078] 그리고 분산 런타임 엔진(420)은 그래프 데이터의 업데이트의 비용 증가를 방지하기 위하여, 정점 속성값과 그래프 변형의 변경 사항(즉, 정점의 추가 또는 삭제, 그리고, 기존의 정점들 간의 엣지 변화)을 증분 정보(또는 델타 정보, 업데이트 정보)로 저장할 수 있다. 또한, 변경 정보가 무한정으로 커지는 것을 방지하기 위하여, 분산 런타임 엔진(420)은 비용 기반 유지 관리 전략에 따라 증분 정보를 기본 데이터에 병합할 수 있다.

[0079] 한편, 도 4를 도시하고 설명함에 있어서, 분산 런타임 엔진(420)이 쿼리 실행과 그래프 저장을 모두 수행하는 것으로 도시하고 설명하였지만, 구현시에 분산 런타임 엔진은 복수의 장치로 구성되고, 각 기능을 별도로 수행할 수 있다. 또한, 하나의 기능에 대해서도 복수의 장치가 클라우드 방식으로 동작할 수 있다.

[0080] 이하에서는 본 개시에서 이용하는 프로그램 언어인 L_{NGA}에 대해서 자세히 설명한다.

[0081] L_{NGA}는 명령형 프로그래밍 인터페이스인 도메인 특정 언어이다. L_{NGA}에서 사용자는 중첩된 반복을 사용하여 각 정점 주변의 다중 홉 거리 내의 이웃들에 대한 그래프 순회를 표현할 수 있다.

[0082] L_{NGA}는 기본 데이터 타입(primitive data types), 누적형 데이터 타입(Accumulator types are) 및 복합 데이터 타입(composite data types)을 지원한다. 예를 들어, L_{NGA}는 5 가지 기본 데이터 타입{bool, int, long, float,

double}을 제공할 수 있다. 한편, 구현시에는 상술한 타입 이외의 데이터 타입을 이용할 수도 있으며, 상술한 타입 중 일부만을 이용할 수도 있다.

[0083] 누적형 데이터 타입은 Accm<primitive type, operation>으로 정의되며, 여기서 기본 데이터 타입은 5 가지 기본 데이터 타입 중 하나일 수 있으며, 연산은 S_{UM} 및 M_{IN}과 같은 임의 연산자일 수 있다. 이러한, 누적형 데이터 타입은 순회 및 업데이트 중에 데이터를 임시로 저장하기 위한 것이므로, 이들의 값은 각 슈퍼스텝의 시작 부분에서 연산자의 아이덴티티 요소로서 초기화될 수 있다.

[0084] 또한, 다양한 머신 학습 작업을 용이하게 하기 위해, Array; Array<type, size>와 같은 복합 데이터 타입도 지원할 수 있다.

[0085] 그리고 L_{NGA}는 정점 타입(vertex type)과 전역 변수 타입을 지원한다. 정점 타입은 각 정점에 대한 데이터를 저장하는 데이터 타입이고, 전역 변수 타입은 모든 정점들이 공유하는 데이터를 저장하는 데이터 타입이다.

[0086] 그리고 LNGA는 애플리케이션 로직을 설명하는데 사용되는 다음과 같은 5 개의 명령어를 지원한다.

[0087] 1. 식을 변수에 바인딩하기 위한 L_{ET}

[0088] 2. 식의 값을 변수에 할당하기 위한 A_{SSIGN}

[0089] 3. 식의 값을 변수에 누적하기 위한 $A_{CCUMULATE}$

[0090] 4. 세트 또는 범위에 대해 반복하기 위한 $F_{OR,}$

[0091] 5. 분기를 위한 I_F-E_{LSE}.

[0092] 이와 같은 프로그램 언어를 사용하여, 정점 중심 및 이웃 중심 분석을 쉽게 프로그래밍할 수 있다.

[0093] 한편, 확장 가능한 NGA 처리를 위해, 스트리밍 처리 모델을 확장하고 GSA(Graph Streaming Algebra)를 정의한다. 구체적으로, 저장소에 저장된 그래프 데이터를 스트림 형태로 모델링하고, NGA에 대한 그래프 순회를 스트림들에 대한 워크들의 나열으로 모델링할 수 있다.

[0094] 먼저, 고정된 메모리 크기로 워크를 효율적으로 처리하기 위하여, 중첩된 그래프 윈도우(Nested Graph Windows)와 두 개의 스트림 연산자(윈도우 탐색(Window Seek) 및 윈도우 조인(Window Join))을 이용한다.

[0095] GSA는 그래프를 방향성 그래프 G(V, E)로 모델링할 수 있다. 여기서 V와 E는 각각 정점과 에지의 집합을 나타낸다. 각 정점은 속성들을 갖는다.

[0096] 무방향성 그래프는 양방향에 에지들의 쌍들이 있는 방향성 그래프로 모델링될 수 있다. 이러한 점에서, 그래프를 중복이 허용되지 않는 단순한 그래프로 모델링할 수 있다.

[0097] 길이 k를 갖는 워크는 E의 에지((1 ≤ i < j ≤ k +1)가 되는 일부 i 및 j에 있어서 E의 (u_i, u_j))에 의해 연결되는 V, (u_1, ..., u_{k+1})의 (k +1) 정점들의 시퀀스로 표현할 수 있다.

[0098] 스트림(S)은 튜플(tuple)의 시퀀스이다. 그리고 각 튜플(r_m)에는 하나 이상의 데이터 열과 다중도(multiplicity) m ∈ {-1, 1}가 저장되어 있을 수 있다. 여기서, 다중도를 양수 또는 음수 값으로 허용함으로써, 널리 사용되고 있고 잘 이해되는 동일한 데이터 모델을 이용하여 삽입 및 삭제를 나타낼 수 있다.

[0099] 예를 들어, 에지(e)의 삽입 또는 삭제와 같은 그래프 변형은 e_1 또는 e_{-1}로 모델링될 수 있다. 이하에서는 설명을 용이하게 하기 위하여, 튜플 r_1을 단순히 r로 나타낸다.

[0100] 저장소에 저장된 그래프 데이터를 스트림으로 모델링할 수 있다. 중복이 허용되지 않는 단순한 그래프로 그래프를 모델링할 때, 정점 및 에지 튜플의 다중성은 1 또는 -1이다. 이하에서는, 정점 튜플들의 시퀀스와 에지 튜플들의 시퀀스를 각각 정점 스트림(vs)과 에지 스트림(es)으로 간주한다. 여기서, 정점 튜플은 정점 ID와 그것의 속성값으로 구성될 수 있다.

[0101] 에지 튜플은 소스 및 대상 정점 ID 들로 구성될 수 있다. 또한, 그래프 스트림(gs)을 정점 스트림(vs) 및 에지 스트림(es)의 쌍으로 정의한다(gs = (vs, es)). 개념적으로, 그래프 스트림 gs = (vs, es)는 디스크에 저장된

하위 그래프를 나타낸다.

[0102] 도 5는 일 실시 예에 따른 데이터그래프의 예를 도시한 도면이고, 도 6은 도 5의 데이터 그래프에 대한 그래프 스트림의 예를 도시한 도면이다.

[0103] 도 5를 참조하면, 튜플들의 정점 스트림(vs)을 나타내며, 정점 스트림(vs) 각각은 정점 ID 및 차수(degree) 속성값을 가질 수 있다. 여기서, V의 속성값들을 정점 스트림으로 간주할 수 있다.

[0104] 정점 스트림(vs)의 첫 번째 및 두 번째 튜플은 각각 v_0.degree = 2 및 v_1.degree = 2를 나타낸다. 도시된 바와 같이, 에지 목록을 에지 스트림으로 간주할 수 있다.

[0105] 그리고 정점 스트림(vs) 및 에지 스트림(es)은 정점 ID에 의해서 정렬될 수 있다.

[0106] 그래프 순회는 랜덤 액세스 및 반복으로 인해 높은 데이터 입출력 비용을 초래할 수 있다. 예를 들어, L_{NGA}의 중첩된 반복(nested for-loops)은 DFS와 유사한 순회를 표현할 수 있다.

[0107] 랜덤 액세스와 반복적인 데이터 입출력을 피하기 위해, 본 개시에서는 중첩된 그래프 윈도우를 이용한다. 중첩된 그래프 윈도우를 그래프 스트림들과 함께 사용할 경우, 스트림들에 대한 순차적 스캔을 수행으로 인해, 특정 정점이 로드되면 메모리에서 가능한 한 많은 워크를 나열하여 수행할 수 있다. 이와 같은 한 번의 데이터 로드를 통하여 여러 워크를 수행하는바, 데이터의 반복되는 입출력을 줄일 수 있다.

[0108] 또한, NGA의 확장 가능한 처리를 위해 메모리로 워크들을 나열할 수 있다.

[0109] 중첩된 그래프 윈도우에서, 스트림(S)에 대한 윈도우를 고정 크기(sz)를 갖는 스트림(S)의 하위 시퀀스로 정의하며, 윈도우를 $\{rm \in S\}_{sz}$로 표시한다.

[0110] 그래프 윈도우(gw)는 그래프 스트림 gs = (vs, es)에 대해서, 정점 스트림(vs)에 대한 정점 윈도우(vw)와 에지 스트림(es)에 대한 에지 윈도우(ew)의 쌍으로서 정의한다. 여기서, 윈도우(gw = (vw, ew))는 vw의 정점과 ew의 에지의 메모리에 로드되는 하위 그래프를 나타낸다.

[0111] 고정된 메모리 크기로 길이 k의 워크들을 나열하기 위해, 메모리 공간을 k 영역으로 분할할 수 있다. 그런 다음 i 번째 영역을 사용하여 $i \in [1, k]$에 대한 그래프 윈도우 gw_i를 읽을 수 있다. k + 1 그래프 윈도우 (gw_0, ..., gw_k)의 튜플을 중첩된 그래프 윈도우로서 참조하여 ngw_k로 표시한다.

[0112] 첫 번째 그래프 윈도우 gw_0은 가상 정점 v*와 v*에서 활성 정점 V_{active}까지의 에지로 구성된 가상 하위 그래프로 정의할 수 있다. $i \geq 1$에 대해 $ngw_{i1} = (gw_0, ..., gw_{i1})$이 주어지면, gw_i는 ngw_{i-1}로 표현되는 것들과 연결된 하위 그래프로서 재귀적으로 정의할 수 있다.

[0114] 도 7은 데이터그래프의 삼각형 카운팅을 위한 중첩 그래프 윈도우의 예를 도시한 도면이다. 구체적으로, 도 7은 도 5에서의 G_0이 있는 중첩된 그래프 윈도우와 각 윈도우에 로드된 정점을 나타낸다.

[0115] 도 7을 참조하면, 그래프 윈도우 gw_i는 그래프 스트림 $gs_i(i \in [1, 3])$에 대응한다. 그리고 gw0은 gs_1의 vs_1에 포함된 V_{active}에 의해 사전 정의된 속성인 active로서 정의된다.

[0116] gw_0에는 가상 정점 v* 및 v*에서 $V_{active} = \{v_0, ..., v_7\}$까지의 에지가 있다. 모든 윈도우($gw_0$ .. gw_3는 같은 크기를 가지며 각각 두 개의 정점과 해당 에지들을 보유할 수 있다.

[0117] $ngw_0 = (gw_0)$이 주어지면, v_0 및 v_1을 gw_1에 로드함으로써, v*를 생략한 길이 1의 워크((v_0, v_1), (v_0, v_5), (v_1, v_0) 및 (v_1, v_5))들을 나열할 수 있다.

[0118] $ngw_1 = (gw_0, gw_1)$이 주어지면, v_1 및 v_5를 gw_2에 로드함으로써, (v_0, v_1, v_5) 및 (v_0, v_1, v_0)과 같은 길이가 2인 워크들을 나열할 수 있다. 이러한 방식으로, k 메모리 영역에 대해 $ngw_i(i \in [1, k])$를 재귀적으로 정의함으로써 제한된 메모리 크기로 길이 k까지 워크들을 나열할 수 있다.

[0119] 이후에는 워크 생성 연산자인 $W_{ALK}(\omega)$를 사용하여 그래프 순회를 워크들의 나열로 모델링할 수 있다. 워크 생성 연산자(W_{ALK})는 n 개의 그래프 스트림을 입력으로 취하는 n 항 연산자 ($n \geq 2$)이다. 워크 생성 연산자는 윈도우

탐색(Window Seek) 및 윈도우 조인(Window Join)이라는 두 가지의 하위 연산자들을 가질 수 있다. 윈도우 탐색은 그래프 스트림에서 하위 그래프를 로드하기 위한 것이고, 윈도우 조인은 워크들을 나열하는 동안 ngw로 표시되는 메모리의 하위 그래프들에 대한 그래프 순회를 위한 것이다.

[0120] 윈도우 탐색($W\text{-}S_{EEK}$)은 메모리에 있는 것들과 연결된 다른 하위 그래프를 로드할 수 있다. 즉, $ngw_{i1}(i \geq 1)$이 주어지면, $W\text{-}S_{EEK}$는 gs_i에서 gw_i를 로드할 수 있다. 이때, 윈도우 탐색은 선택적으로 gs_i의 데이터에 적용되는 제약 조건 p_i를 받게 된다.

[0121] 예를 들어, 삼각형 카운팅(도 5)에서의 부분 순서 제약(u1 <u2 <u3)과 같은, 제약 조건을 정점에 적용할 수 있다. 구체적으로, $i \geq 1$에 대해 $ngw_{i1} = (gw_0, \ldots, gw_{i1})$이 주어지면, $W\text{-}S_{EEK}$는 $vw_i = \{u_i \in vs_i \mid p_i(u_0,\ldots, u_i)$을 만족하도록 $gs_i = (vs_i, es_i)$에서 $gw_i = (vw_i, ew_i)$를 로드할 수 있다. 여기서 [0, i-1]의 j에 대하여 $u_j \in vw_j$이고 일부 $l \in [0, i-1]\}$ sz에 대하여 $(u_l, u_i) \in ew_l$이며 $ew_i = \{e \in es_i \mid e.src \in vw_i\}$ sz'이다.

[0122] 윈도우 조인($W\text{-}J_{OIN}$)은 ngw_k로 표시되는 메모리의 하위 그래프에 대해 그래프 순회를 수행할 수 있다. 윈도우 조인($W\text{-}J_{OIN}$)의 출력값은 워크들의 스트림 $(u_1, \ldots, u_{k+1})$이다. 여기서 u_i는 i 번째 그래프 윈도우($i \in [1, k]$)의 정점이다.

[0123] 윈도우 조인($W\text{-}J_{OIN}$)은 선택적으로 나열할 워크들을 필터링하는 제약 조건 p'를 취할 수 있다. 구체적으로, $ngw_k = (gw_0, \ldots, gw_k)$가 주어지면, $W\text{-}J_{OIN}$은 워크들 $(u_1, \ldots, u_{k+1})$을 나열할 수 있다. 여기서 $i \in [1, k]$에 대하여 $u_i \in vw_i$이고, $(u_k, u_{k+1}) \in ew_k$이며, $p'(u_1, \ldots, u_{k+1})$은 참으로 평가될 수 있다.

[0125] 도 8은 3-홉 순회를 위해 3 개의 그래프 스트림(gs_1, gs_2, gs_3)을 취하는 W_{ALK} 연산자의 예를 나타낸다.

[0126] 도 8을 참조하면, 도시된 세 가지 $W\text{-}S_{EEK}$ 연산자는 다음과 같은 방식으로 수행할 수 있다.

[0127] $V_{active} = \{v_0, \ldots, v_7\}$인 ngw_0이 주어지면, gs_1에 대해 첫 번째 $W\text{-}S_{EEK}$를 수행하고 v_0 및 v_1을 gw_1에 로드할 수 있다. 그 결과, $ngw_1 = (gw_0, gw_1)$이 된다.

[0128] 그 다음, ngw_1을 사용하여, gs_2에 대한 두 번째 $W\text{-}S_{EEK}$를 수행하고 v_1 및 v_5가 gw_1의 v_0 및 v_1에 연결되어 있고, 이들이 제약 사건 $u_1 < u_2$를 충족하므로 이들을 gw_2에 로드할 수 있다.

[0129] 그 다음, ngw_2를 사용하여, gs_3에 대한 마지막 $W\text{-}S_{EEK}$를 유사하게 수행하고 v_5가 gw_2의 v_1에 연결되어 있고 이것이 제약 조건 $u_2 < u_3$을 충족하므로 이것을 gw_3에 로드할 수 있다. 이제 ngw_3이 모두 로드되었으므로, 이제 ngw_3에서 $W\text{-}J_{OIN}$ 연산을 수행할 수 있다.

[0130] 이에 따라, 4 개의 워크(walk) (v_0, v_1, v_5, v_0), (v_0, v_1, v_5, v_1), (v_0, v_1, v_5, v_2) 및 (v_0, v_1, v_5, v_4)를 생성할 수 있다.

[0131] 이중, 첫번째 워크로부터, 삼각형 $<v_0, v_1, v_5>$을 찾을 수 있고, 삼각형을 형성하는지를 확인할 수 있다.

[0132] 이하에서는 GSA에서 이용하는 연산자를 설명한다. 이하의 표 1에 GSA 연산자를 표시하였다.

표 1

[0133]

Input	Output	Name	Symbol
Window, Stream	*Window*	*Window Seek*	*W-Seek*
Window	*Stream*	*Window Join*	*W-Join*
Stream	*Stream*	*Walk*	ω
		Filter	σ
		Map	Π
		Union	$\cup$
		Difference	$\ominus$
		Assign	$\leftarrow$
		Accumulate	$\uplus$
Stream, Operator	*Stream*	*Apply*	α

[0134] 표 1을 참조하면, 윈도우 탐색과 윈도우 조인 이외에도, F_{ILTER}, M_{AP}, U_{NION}, 및 $D_{IFFERENCE}$를와 같은 연산자를 포함함을 확인할 수 있다.

[0135] A_{SSIGN} 및 $A_{CCUMULATE}$는 데이터 수정 연산자이다. 구체적으로, 속성값 또는 집계 값을 내부 상태로서 사용하는 상태 저장 연산자들이다. 예를 들어, A_{SSIGN}(←id, attr)은 기존 값을 삭제하고 새로운 값을 삽입함으로써 ID가 id인 정점의 속성을 업데이트할 수 있다. 즉, A_{SSIGN}은 속성의 기존 값과 새로운 값을 각각 포함하는 튜플들의 스트림을 입력값으로 가지며, 각 입력 튜플에 대해 두 개의 튜플을 출력하며, 하나는 기존 값을 삭제하기 위한 것이고 다른 하나는 새로운 값을 삽입하기 위한 것이다.

[0136] $A_{CCUMULATE}$(id,f(attr))는 Abelian monoid의 누적 함수 f를 사용하여 키(key) 열 id가 있는 입력 튜플들의 속성값들을 집계한다.

[0137] 고정 소수점 연산자와 같은 반복 실행을 위한 연산자를 정의하는 대신에, 실행 엔진 내에 BSP(Bulk Synchronous Parallel) 모델의 반복 실행 로직을 임베드하였다. 그리고 명시적 종료 조건으로서 고정 소수점 반복과 동일한 표현력을 갖는 활성 정점들의 존재를 사용할 수 있다.

[0138] 이하에서는 L_{NGA} 프로그램(Q)을 GSA 쿼리 계획으로 컴파일하는 방법에 대해 설명한다.

[0139] L_{NGA} 프로그램(Q)의 각 UDF(예를 들면, $T_{RAVERSE}$)를 GSA의 쿼리 계획으로 컴파일할 수 있다. 각 L_{NGA} 문을 해당 GSA 표현 식으로 변환함으로써 명령문 수준(statement level)에서 컴파일을 수행할 수 있다.

[0140] 그런 다음, 이 표현 식을 UDF를 나타내는 단일 GSA 표현 식으로 병합할 수 있다. L_{NGA} 프로그램(Q)에서의 초기화($I_{NITIALIZE}$), 순회($T_{RAVERSE}$) 및 업데이트(U_{PDATE}) 각각의 계획을 P_{Qinit}, P_{Qtrav} 및 P_{Qupd}로 표시한다.

[0141] 먼저, L_{NGA}의 명령문을 상관 하위 쿼리로 모델링할 수 있다. 상관 하위 쿼리는 외부 쿼리에 네스팅되는 쿼리이며, 외부 쿼리의 각 출력 튜플에 의해 매개 변수화될 수 있다. 예를 들어 F_{OR}는 스트림을 가져와서 Where 조건을 포함하는 특정 조건을 만족하는 것으로부터 튜플들을 출력하는 F_{ILTER} 연산으로 변환할 수 있다. 'For (v) in (u.nbrs)' 명령문은 정점 u에 의해 매개 변수화되어, 에지 스트림(es)을 가져와서 에지 $e \in es$를 (e.src = u.id)가 되도록 출력할 수 있다.

[0142] 이름 바꾸기 연산자($\rho(b_1,\ldots, b_n)/(a_1,\ldots, a_n)$)는 이름 그대로 a_i 열의 이름을 $b_i(i \in [1, n])$로 바꾸는데 사용될 수 있다.

[0143] 변환된 표현식들을 병합하고 UDF의 쿼리 계획을 구축하기 위해, 관계형 대수에서 상관 하위 쿼리의 실행을 모델링 하는데 일반적으로 사용되는 A_{PPLY} 연산자를 사용할 수 있다. 그리고 스트림(S)과 GSA 표현식 expr(r)을 입력으로 받도록 확장할 수 있다.

[0144] 스트림(S)에서 출력 스트림으로 각 튜플 r에 대한 표현식의 평가 결과를 출력할 수 있다. 즉, 각 $r_m \in S$에 대해, $\langle r,expr(r)\rangle_m$; $S\ \alpha\ expr = \{\langle r,expr(r)\rangle\ m \mid r_m \in S\}$을 출력할 수 있다.

[0145] A_{PPLY}에 의해 결합된 표현식은 GSA 표현식의 상관 실행을 나타낸다. 쿼리 역상관 기술을 활용하여, A_{PPLY} 연산자를 제거하고 단일 GSA 표현식을 얻을 수 있다.

[0146] 이하에서는, 도 9를 참조하여, PR, P_{prtrav}에서의 $T_{RAVERSE}$의 컴파일에 대해 설명한다.

[0148] 도 9는 페이지링크의 순회에 대한 컴파일링 예를 도시한 도면이다. 구체적으로, A_{PPLY} 연산자(α)가 있는 코드 및 표현 트리를 나타낸다.

[0149] 도 9를 참조하면, 좌측 코드의 7 행에서, $T_{RAVERSE}$(u)가 정점 스트림 vs의 각 활성 정점 u에 대해 호출되므로(즉, active = true), $vs' = \rho_{u/id}(\sigma_{active=true}(vs))$를 갖게 된다.

[0150] 좌측 코드의 8 행은 표현식 u.rank/u.차수를 로컬 변수 val에 바인딩한다. val에 대한 모든 후속 레퍼런스들은

이 표현식으로 대체할 수 있다.

[0151] 좌측 코드의 9 행은 u의 이웃들 v를 반복하는 반복을 나타낸다. 이것은 각 x invs′ 에 대해 e.src = x.u가 되도록 에지 스트림(es)에서 e를 선택하는 상관 표현식으로 A_{PPLY} 연산자를 사용하여 모델링 될 수 있다. 예를 들어, (vs′ α expr(x))와 같이 모델링 될 수 있다. 여기서 expr (x) = ρ(u,v)/(src,dst)(σsrc=x .u(es))이다. 이때, A_{PPLY} 연산자를 제거하기 위해, 쿼리 역상관 기술을 적용할 수 있다.

[0152] 결과적으로, 표현식 (vs′ α expr(x))는 W_{ALK} 연산자를 사용하여 Pω(vs, es) =ω(vs′ .u=es′ .u) (vs′ , es′)로 변환될 수 있다. 여기서 es′ = ρ(u,v)/(src,dst)(es)이다.

[0153] 좌측 코드의 10 행은 각 이웃 v의 합계에 대한 val의 값을 누적하는 명령이다. 이것은 W_{ALK}로부터 각 튜플에 대해 M_{AP}(Π)에 의해 방출된 튜플들을 갖는 합계에 대한 $A_{CCUMULATE}$()에 대응한다.

[0154] 마지막으로, 전체 표현 트리는 (v, Sum(val)) Π(v, rank/차수 as val) Pω(vs, es)로 변환될 수 있다.

[0155] GSA에서 L_{NGA} 프로그램을 나타냄으로써, 증분 쿼리를 자동으로 도출하고, 효율적인 실행을 위해 여러 최적화를 체계적으로 통합할 수 있다. 이하에서는 쿼리 증분화, 실행, 최적화 및 동적 그래프 저장에 대해서 순서대로 설명한다.

[0156] 스트림의 튜플이 양수 또는 음수 다중성을 가지므로, 그래프 데이터의 정점의 속성값 변화 및 그래프 형태의 변화는 델타 스트림으로 표현될 수 있다. 스트림(S)의 델타 스트림 (ΔS)는 스트림(S)에 대한 업데이트를 나타내는 스트림이다. 이러한 델타 스트림은 증분 정보로 표현될 수 있다.

[0157] 예를 들어, 정점 속성값에 대한 업데이트는 델타 스트림에서 두 개의 튜플로 표현되며, 하나는 이전 값을 삭제하기 위한 것이고 다른 하나는 새로운 값을 삽입하기 위한 것이다.

[0158] 증분 계산의 목표는 전체 그래프에서 비용이 많이 드는 재실행 없이 이전 분석 결과를 효율적으로 업데이트하는 것이다.

[0159] 쿼리 Q에 대한 증분 쿼리 ΔQ를 다음과 같이 정의한다.

[0160] N 그래프 스트림의 GSA 쿼리 Q인, $Q(gs_1, \ldots, gs_N)$이고, 델타 그래프 스트림 ($\Delta gs_1, \ldots, \Delta gs_N$)이 주어진 경우, 증분 쿼리 ΔQ는 다음과 같은 식에 의해서 정의될 수 있다.

[0161] [수학식 1]

[0162] $$Q(gs_1 \cup \Delta gs_1, \ldots, gs_N \cup \Delta gs_N) = Q(gs_1, \ldots, gs_N) \cup \Delta Q\ (gs_1, \ldots, gs_N, \Delta gs_1, \ldots, \Delta gs_N)$$

[0163] 여기서 ∪는 합집합 연산자이다. 수학식 1의 증분 쿼리 정의를 기반으로, 도 10에 도시된 바와 같이 GSA 연산자에 대한 증분 규칙을 도출할 수 있다.

[0164] 이 규칙 도출은 간단하기 때문에, 그 증명을 생략한다. 개별 $W\text{-}S_{EEK}$ 및 $W\text{-}J_{OIN}$ 연산자가 아닌 W_{ALK} 연산자 (ω)를 직접 증분화한다.

[0165] 원-샷 쿼리 계획(P_Q)에 증분화 규칙을 적용함으로써, (델타) 그래프 스트림에 대한 GSA의 증분 쿼리 계획 PΔQ를 얻을 수 있다. 예를 들어, 앞서 설명한 페이지랭크(PageRank, PR) 및 삼각형 카운팅(Triangle Counting, TC)의 컴파일된 순회 표현식을 고려하여 증분 쿼리 계획을 얻을 수 있다.

[0166] PR은, P_{prtrav}에서 도 10에 도시된 규칙 ⑥, ②, ⑦을 적용하여 $P_{\Delta prtrav} = \Delta(P_{prtrav})$를 도출할 수 있다.

[0167] 그리고 규칙 ⑥과 ②에 의해, $P_{\Delta prtrav}$ = (v,Sum(val)) Π(v, rank/차수 as val) (ΔPω)를 얻을 수 있다.

[0168] 규칙 ⑦에 의해, ΔPω = Δ (ω(vs1, es1)) = ω(Δvs1, es1) ∪ ω(vs′ 1, Δes1)을 얻을 수 있다.

[0169] 삼각형 카운팅(Triangle Counting, TC)의 경우, $P_{\Delta tctrav} = \Delta(P_{tctrav})$ = Sum (cnts) Π1 as cnts (ΔPω)이다. 여기서 ΔPω = ω(Δvs1, es1, es2, es3) ∪ ... ∪ ω(vs′ , es′ 1, es′ 2, Δes3)이다.

[0170] 도 10에 도시된 바와 같이, GSA는 증분화 상태에서 닫혀 있으며, GSA의 임의의 P_Q에 대해, $P_{\Delta Q}$도 GSA로 구성될

수 있다. 따라서, P_Q와 $P\Delta Q$ 모두는 GSA를 지원하는 동일한 쿼리 실행 엔진에서 처리될 수 있다.

[0172] 도 11은 본 개시의 일 실시 예에 따른 일회성 쿼리와 증분 쿼리를 처리하는 동작을 설명하기 위한 도면이다.

[0173] 구체적으로, 도 11을 참조하여 증분 NGA의 실행 메커니즘에 대해 설명한다.

[0174] 먼저, 표기법을 먼저 설명한다. 타임스탬프 $t \in N \cup \{0\}$에서 그래프의 스냅 샷을 G_t로 표시한다. 그래프 변형 $\Delta G_t = G_t - G_{t-1}$이고, ΔG_t는 타임스탬프 t에서 발생하는 그래프 변형 연산들에 대한 튜플들로 구성된다(에지 삽입 또는 삭제).

[0175] 그리고 $gs_{t,s} = (vs_{t,s}, es_t)$는 슈퍼스텝 s에서의 타임스탬프 t에서 그래프 스트림을 나타낸다. 여기서 $vs_{t,s}$ 및 es_t는 각각 해당 정점 스트림과 에지 스트림을 나타낸다.

[0176] 또한, vs_(t, s)의 정점 속성값들을 누적형 타입은 $A^{accm}_{t,s}$로, 비-누적형 타입은 $A_{t,s}$로 개별적으로 나타낸다. 예를 들어 PR(도 5)에서 속성들 active 및 rank는 비-누적형 타입이고, 속성 sum은 누적형 타입이다. 정점 v의 속성값은 $A_{t,s}(v)$로 나타낸다.

[0177] 원-샷 분석 모듈(1110)은 모든 슈퍼스텝 $s \geq 0$에서 원-샷 쿼리 Q를 실행할 수 있다. 구체적으로, 원-샷 분석 모듈(1110)은 제1 트래버스 모듈(111) 및 제1 업데이트 모듈(113)을 포함할 수 있다. 여기서, Q는 순회 및 업데이트의 쿼리 계획으로 구성될 수 있다.

[0178] 제1 트래버스 모듈(111)은 Q(에지 스트림(es_t)) 및 비-누적형 타입의 속성값들($A_{t,s}$)을 입력값들로 취하고, 다음 슈퍼스텝에 대한 비-누적형 타입의 속성값들($A_{t,s+1}$)(112)을 출력할 수 있다. 즉, (e_{st}, $A_{t,s}$)를 취하고 누적형 타입의 속성값들($A^{accm}_{t,s}$)을 출력할 수 있다.

[0179] 제1 업데이트 모듈(113)은 정점 속성값들을 업데이트하고, 다음 슈퍼스텝에 대한 정점 활성화를 수행할 수 있다. 즉, ($A_{t,s}$, $A^{accm}_{t,s}$)를 취하고, $A_{t,s+1}$을 출력한다. 동적 그래프 저장소는 효율적인 증분 계산을 위해 $A^{accm}_{t,s}$ 및 $A_{t,s}$를 모두 유지할 수 있다.

[0180] 증분 분석 모듈(1120)은 Q의 입력의 변화를 고려하여 Q의 출력의 변화를 계산할 수 있다. 즉, 각 슈퍼스텝 s에 대해, ΔQ는 델타 스트림 Δes_{t+1} 및 $\Delta A_{t+1,s}$를 추가적인 입력값들로 취하여 $\Delta A_{t+1,\ s+1}$를 출력한다. 여기서, ΔQ는 각각 증분화된 순회 및 업데이트를 나타내는 Δ순회 및 Δ업데이트로 구성될 수 있다.

[0181] 이러한 증분 분석 모듈(1120)은 제2 트래버스 모듈(1121), 제2 업데이트 모듈(1123)을 포함할 수 있다.

[0182] 제2 트래버스 모듈(1121)은 각 슈퍼스텝 s에 대해, $\Delta A_{t+1,s} = (A_{t+1,s}\ A_{t,s})$ 및 Δes_{t+1}이 주어지면, Δ순회는 그래프 순회를 수행하여 수정된 속성값 또는 에지 변경이 있는 정점들로 구성된 워크들을 나열할 수 있다. 나열된 워크들을 사용하여, 누적형 타입의 속성값들은 증분적으로 갱신되어, $\Delta A^{accm}_{t+1,s} = (A^{accm}_{t+1,s}\ A^{accm}_{t,s})$가 될 수 있다.

[0183] 제2 업데이트 모듈(1123)은 Δ업데이트가 속성값이 변경된 정점 v에 대해 실행될 수 있다. $A_{t+1,s}(v) \neq A_{t,s}(v)$ 또는 $A^{accm}_{t+1,s}(v) \neq A^{accm}_{t,s}(v)$. 그 결과 $\Delta A_{t+1,s+1} = (A_{t+1,s+1}\ A_{t,s+1})$이 된다. 동적 그래프 저장소에 $\Delta A_{t+1,s+1}$ 및 $\Delta A^{accm}_{t+1,s}$를 추가할 수 있다.

[0184] 한편, 증분 순회의 워크 나열은 성능에 큰 영향을 미치므로, 최적화가 필요하다. 증분 순회(Δ순회)의 처리는 수정된 속성값들 또는 에지 변경들이 있는 정점을 포함하는 새롭거나 무효화된 워크들을 나열해야 한다.

[0185] 도 10에 도시된 규칙 ⑦에 따르면, k 스트림이 있는 증분 W_{ALK}는 워크들의 k 하위 쿼리 ($\cup_{n=1}^{k} q_n$)의 합집합으로, 각각 k-1 스트림과 하나의 델타 스트림을 취한다.

[0186] 이하에서는 증분 NGA, 특히 Δ-walk 나열이라고 부르는 Δ-walk의 나열에 대한 기술을 적용한다. G_{t+1} 및 슈퍼스

텝 s에서, Δ-walk는 워크 (u_1, ..., u_{k+1})이며, 일부 i ∈ [1, k]에 대해, $A_{t+1,s}(u_i) \neq A_{t,s}(u_i)$이 되는 정점 u_i 또는 e.src = u_i가 되는 에지 e ∈ ΔG_{t+1}가 존재한다. Δ-walk의 다중도는 결합된 튜플들의 다중도 곱에 의해 1 또는 -1로 계산될 수 있다. 이하에서는 Δ-walk들의 시작 정점 세트(u_1)를 V_Δ로 표시한다.

[0187] 예를 들어, 삼각형 카운팅(Triangle Counting, TC) 쿼리를 고려한다.

[0188] 도 12는 에지 e = (v_3, v_5)가 삽입된 그래프 G_1을 보여주고, 도 13은 ΔG_1 이후에 새로 발견된 삼각형들을 보여준다. TC의 원-샷 분석을 위해 모든 활성 정점들로부터 그래프 순회를 수행하였다.

[0189] 그러나 증분 삼각형 카운팅은, 원-샷 분석에 사용되는 고정 순회 순서는 차선이 될 수 있으며, v_2 또는 v_3 이외의 정점에서 시작하는 순회는 삼각형이 없기 때문에 낭비이다. 이를 위해, 트래버셜 리오더링(traversal reordering)과 이웃 가지치기(neighbor pruning)를 이용하는 방법을 이하에서 설명한다.

[0190] 먼저, 증분 NGA에 적용하는 기본 결합 순서 최적화로서 트래버셜 리오더링에 대해 설명한다.

[0191] $P_{\Delta tc^{trav}}$ 은, 삼각형 <v3, v4, v5>를 찾는 하위 쿼리 q_4 = ω(vs′ , es′ $_1$, es′ $_2$, Δes_3)를 고려한다.

[0192] 도 14(a)은 원래 순회 순서 (u_1 → u_2 → u_3)를 사용한 q_4의 쿼리 계획을 나타낸다.

[0193] 도 14(a)를 참조하면, Δes_3를 통한 W-S_{EEK}까지, 모든 활성 정점에 대해 (vs′ , es′ $_1$, es′ $_2$)를 갖는 W-S_{EEK}의 전체 재실행을 수행하므로, 이것은 낭비이다.

[0194] 관계형 데이터베이스의 조인 리오더링과 유사하게, 그래프 순회를 재정렬하여 이 문제를 해결할 수 있다. 즉, 가능한 한 계획 초기에 델타 정점 또는 에지 스트림에 대해 W-S_{EEK}를 수행하도록 그래프 순회를 재정렬하는 것이다.

[0195] 도 14(b)은 새로운 순회 순서 (u_1 → u_3 → u_2)로 재정렬된 계획을 보여준다.

[0196] 도 14(b)를 참조하면, 재정렬된 계획을 통하여, e ∈ Δes_3를 통해 v_3에서 v_5로 이동하고, v_4 (< v_5 및 > v_3)로 이동하여, 불필요한 그래프 순회를 피하면서 새로운 삼각형을 찾아 낼 수 있다. 그리고 사용자 휴리스틱 및 선택 조건(예를 들면, 활성)의 이점을 얻을 수 있도록 시작 정점을 수정할 수 있다.

[0197] 이것은 또한 시스템 전반에 유익하며. 시작 정점의 데이터를 위한 메모리 공간은 모든 하위 쿼리에서 공유될 수 있다.

[0198] 다음으로, 이웃 가지치기에 대해 설명한다.

[0199] $P_{\Lambda tc^{trav}}$의 경우, 새로운 삼각형 <v_2, v_3, v_5>를 찾아내는 하위 쿼리 q_3 = ω(vs′ , es′ $_1$, Δes_2, es_3)을 고려한다. ΔG_1의 영향을 직접 받는 v_3의 1 홉 이웃인 v_2에서만 순회를 시작하여 새로운 삼각형이 발견되었는지 확인할 수 있다.

[0200] 즉, Δ-walk 나열에 대한 시작 정점을 수정된 속성값들 또는 에지 연결들이 있는 정점들의 인접 정점들로 제한하여 불필요한 그래프 순회를 제거할 수 있다.

[0201] q_3에 대한 예에서 이웃 가지치기는, ΔG_1의 영향을 받는 정점에서 다중 소스 BFS(MS-BFS)를 수행할 수 있다.

[0202] 도 14(c)은 MS-BFS 동안 이용되는 선택 조건들과 함께 q_3의 계획을 보여준다. 도 14(c)를 참조하면, 정점들 {u_3 | <u_2,u_3>$_m$ ∈ Δes} = {v_3, v_5}에서 MS-BFS를 시작한다. Depth 1에서는 e를 통해 v_3과 v_5에서 서로 역방향으로 순회할 수 있다. 선택 조건 u_2(= v_3) < u_3(= v_5)를 이용하여 v_5에서 v_3까지만 따른다.

[0203] Depth 2에서 동일한 방식으로 v_3에서 v_2(<v_3)로 순회하고 q_3의 VΔ에 대한 후보로서 v_2를 찾아낼 수 있다. q_3를 처리할 때, MS-BFS 동안 방문한 정점들로만 구성된 워크들을 나열하고, 새로운 삼각형 <v_2, v_3, v_5>를 찾을 수 있다.

[0204] 이웃 가지치기를 위한 MS-BFS의 세부 프로세스에 대해 설명한다. 피연산자 중 하나로서 Δes를 사용하는 하위

쿼리를 고려하여 X^0 = {e.dst | e_m ∈ Δes}에서 시작하여 최대 깊이 k까지 역방향 MS-BFS를 수행할 수 있다.

[0205] 깊이 i ≥ 1에서, X^{i1}에서 역방향 순회를 수행하고 X^i = {e.src | e.dst ∈ X^{i1} ∧ e ∈ es^i ∧ $Pred_{ei}$ (e) ∧ $Pred_{vi}$ (e.src)}를 확인할 수 있다. 여기서 es^i는 i = 0이면 Δes이고 그렇지 않으면 es′ 이다.

[0206] 또한, $Pred_{ei}$와 $Pred_{vi}$는 각각 해당 에지와 정점의 선택 조건을 나타낸다. Δvs가 있는 하위 쿼리의 경우, {v | <v, value>$_m$ ∈ Δvs}에서 시작하여 최대 깊이 (k 1)까지 MS-BFS를 수행할 수 있다. 그 후, Δ-walk들을 나열할 때, MS-BFS 동안 방문한 정점들만 따를 수 있다.

[0207] 마지막으로, 탐색/윈도우 쉐어링(seek/window sharing)이라는 기술을 설명한다.

[0208] $\mathbf{P}_{\Delta tc^{trav}}$ 에 대한 W_{ALK}의 네 개의 하위 쿼리가 있지만, 개별적으로 처리하면 반복되는 입출력으로 인해 성능이 저하되고 병렬 처리 수준이 낮아질 수 있다.

[0209] 즉, 일괄 처리 기술을 활용하여, 그래프 윈도우에 대한 입출력 및 메모리 공간을 공유하는 것을 달성하는 것이다.

[0210] Walk들의 k 개의 하위 쿼리, $\{q_n\}_{n=1}^{k}$가 주어지는 것으로 가정한다. 모든 하위 쿼리의 i 번째 W-S_{EEK} (1 ≤ i ≤ k)에 의해서 입출력 요청들을 병합하고 관련 하위 쿼리 ID들로 각 입출력 요청에 주석 달기(annotate)를 수행할 수 있다.

[0211] 그 후, 입출력이 제공되면, 주석이 달린 모든 하위 쿼리를 계산할 수 있다. 이러한 방식으로, 동일한 데이터를 반복적으로 로드하지 않고 CPU 사용률을 증가시킬 수 있다.

[0212] 이하에서는 증분 누적 동작에 대해서 설명한다.

[0213] 나열된 Δ-walk들을 사용하여 순회에서 사용되는 누적기를 증분적으로 갱신해야 한다. 예를 들어, 누적 함수가 f인 정점들의 누적기를 고려한다. 그러면 GSA의 증분 쿼리 계획은 (Π(ΔPω))로 표현되며 여기서 Π(ΔPω)는 ΔPω로부터의 새로운(m = 1) 또는 삭제된(m = -1) Δ-walk에 대한 튜플 <v,val>m을 방출할 수 있다.

[0214] ΔPω에 의해 생성되는 새로운 및 삭제된 Δ-walk를 각각 $\Delta^{+}P\omega$ 및 ΔPω로 표시한다. 그러면 v,f(val)(Πv,val(ΔPω)) = v,f(val)(Πv,val($\Delta^{+}P\omega$ ΔPω))가 된다.

[0215] 효율적인 증분을 위해, Abelian 그룹 또는 Abelian monoid를 형성하는 누적 함수들의 대수적 특성을 활용할 수 있다.

[0216] Abelian 그룹은 임의의 x ∈ M에 대해 f(x, 1) = f (1, x) = x가 되는 항등원(identity element) 1 ∈ M을 갖는 도메인 M에 대한 교환 및 연관 덧셈 연산 f : M × M → M을 갖는다. 예를 들면 S_{UM} 및 P_{RODUCT}이다. 임의의 x ∈ M에 대해 f(x,g(x)) = f(g(x), x) = 1이 되는 역(inverse) g : M → M을 가지므로, 삭제시 쉽게 증분 유지될 수 있다. 여기서, x의 누적은 g(x)의 누적에 의해서 상쇄될 수 있다. 그러나 MIN 또는 M_{AX}와 같은 Abelian monoid에는 역이 없다. 따라서, 삭제시 재계산이 필요하다.

[0217] f가 Abelian 그룹에 속하면, f의 역을 사용하여 증분 쿼리 계획을 다시 작성하는 것에 의해 삭제하더라도 재계산(예를 들면, Pagerank에 대한 S_{UM} 및 삼각형 카운팅)을 수행하지 않을 수 있다. 계획 v,f(val) (Πv,val(ΔPω)) = v,f(val)(Πv,val($\Delta^{+}P\omega$ ΔPω))을 v,f(val)(Πv,val($\Delta^{+}P\omega$) ∪ Πv,g(val)(ΔPω))로 다시 작성할 수 있다. 여기서, 삭제를 위하여, val에 대해 역 g를 취할 수 있다.

[0218] 페이지 랭크는, f는 S_{UM} 이고 그 역은 모든 x에 대해 g(x) = x이다. 에지(u, v)의 삽입 또는 삭제는 f에 의해 집계될 튜플 (v, val) 또는 (v, -val)의 삽입으로 이어질 수 있다. 삽입은 val 값이 이전 집계 결과에 가산될 수 있다. 그리고 삭제는 val 값에 역이 더해져서 차감될 수 있다. val 값의 업데이트는 이전 값을 빼고 이전 집계 결과에 새 값을 더하여 처리될 수 있다.

[0219] f가 Abelian monoid에 속하고 ΔPω = 인 경우, v,f(val) (Πv,val($\Delta^{+}P\omega$))를 갖게 된다. 이 경우, Abelian

그룹과 동일한 방식으로 처리될 수 있다. 예를 들어, $\Delta P\omega \neq$ 일 경우, $\Delta P\omega$; $V_{aff} = \{v \mid (v,val) \in \Pi_{v,val}(\Delta P\omega)\}$에서 Δ-walk들에 의해 영향을 받는 정점들(Vaff)에 대한 $A_{CCUMULATE}$의 재계산으로 되돌아갈 수 있다.

[0220] 재계산을 위한 후보 시작 정점 Vre를 찾기 위해, V_{aff}로부터 역방향 MS-BFS를 수행할 수 있다. 그리고 시작 정점들로서 (Vre ∧ Vactive)를 갖는 워크를 수행하고, $A_{CCUMULATE}$가 V_{aff}의 누적형 속성값들을 다시 계산할 수 있다.

[0221] 모든 삭제에 대해 재계산을 수행하는 것은 비용이 많이들 수 있으므로, 가능하면 재계산을 피하기 위해 개별 누적기의 대수적 특성을 활용할 수 있다. 예를 들어 M_{IN}을 사용하면, 삭제된 값이 누적된 값보다 클 경우, 재계산이 필요하지 않게 된다.

[0222] 또한, 누적된 값에 대한 지원 튜플 수를 유지함으로써, 중복 재계산을 피할 수 있다. $M_{IN}(\{1, 2, 5, 1\}) = 1$의 예를 고려한다. 최소값 1에 대해, 지원 튜플 수 2를 유지하면, 불필요한 재계산을 피할 수 있다. 공식적으로 정의된 GSA로 쿼리를 표현하기 때문에, 본 논문 범위 밖의 프로그램 분석을 기반으로 하는 쿼리 재작성 및 다양한 집계 유지와 같은 보다 진보되고 체계적인 기술을 통합할 수 있다.

[0223] 동적 그래프 저장소는 정점 속성값을 위한 정점 저장소와 에지 연결을 위한 에지 저장소로 구성될 수 있다.

[0224] 정점 속성값 및 에지 연결에 대한 업데이트를 지원해야 하지만, 디스크 상의 그래프의 유지보수 업데이트는 비용이 많이 든다. 이러한 비용이 많이 드는 인플레이스 업데이트를 방지하기 위해, 델타 기반 유지 관리 전략을 이용할 수 있다.

[0225] 페이지 제약 때문에, 비-누적형 속성들($A_{t,s}$)의 유지 관리를 설명하는데 중점을 둔다. 누적형 속성들은 유사하게 유지된다.

[0226] 정점 저장소는 정점 속성값의 변경 사항을 델타로 유지한다. G_0을 사용한 원-샷 분석 중에, 정점 속성값들은 슈퍼스텝들에서 변경될 수 있다. 각 슈퍼스텝 s > 0에 대해, 이전 슈퍼스텝에 대한 변경 사항, $A_{0,s}$ $A_{0,s1}$만을 저장할 수 있다. 구체적으로, $A_{0,s}(v) \neq A_{0,s1}(v)$가 되도록 정점들 v의 변경 $A_{0,s}(v)$을 저장할 수 있다.

[0227] Gt를 사용한 증분 분석(t > 0) 동안, 정점 속성값들은 스냅 샷들 및 슈퍼스텝들 모두에서 변경될 수 있다. 즉, $A_{t,s}(v) \neq A_{t,s1}(v)$ 또는 $A_{t,s}(v) \neq A_{t1,s}(v)$가 되도록 정점들 v의 변형된 정보를 저장할 수 있다. 여러 스냅 샷에 걸쳐, 동일한 슈퍼스텝의 델타들을 단일 파일에 저장할 수 있다.

[0228] 정점 저장소는 정점 속성값들의 효율적인 검색을 지원해야 한다. Gt의 경우, 실행 엔진이 메모리 내에 어레이로서 구체화된 $A_{t,s}$로 슈퍼스텝 s를 계산한 후 $A_{t,s+1}$의 검색을 발생시킨다. 그런 다음, 디스크에서 슈퍼스텝 s+1에 대한 델타들을 로드하고 어레이의 값들을 업데이트할 수 있다.

[0229] 비용 모델을 기반으로 델타를 병합할 시점을 결정하는 델타 유지 관리 전략에 대해 설명한다.

[0230] 증분 정보가 디스크 쓰기 양을 줄임으로써 변경 사항을 유지하는 동안, 증분 쿼리를 실행하려면 델타들을 반복해서 읽어 내야 한다. 이러한 트레이드-오프 관계를 비용 모델에 통합하여, 델타들을 병합할 시점을 결정할 수 있다.

[0231] Gt를 사용하는 모든 슈퍼스텝 s에 대해, 델타 병합의 쓰기 비용 $W_{merge}^{(t,s)}$와 델타 읽기 비용 $R_{delta}^{(t,s)}$을 비교할 수 있다.

[0232] $X^{(t,s)}$를 변경된 값들을 델타로 저장해야 하는 정점 집합인 것으로 가정한다; $X^{(t,s)} = \{v \mid (A_{t,s}(v) \neq A_{t,s1}(v)) \vee (A_{t,s}(v) \neq A_{t1,s}(v))\}$. $W_{merge}^{(t,s)} = |\cup_{\tau \leq t} X^{(\tau,s)}|$ 및 $R_{delta}^{(t,s)} = \Sigma_{0<\tau<t} (t - \tau) \times |X^{(\tau,s)}|$을 갖게 된다. 여기서, 읽기 비용은 그 생성부터 지금까지 반복되는 델타 읽기를 고려하여 계산될 수 있다. $W_{merge}^{(t,s)} < R_{delta}^{(t,s)}$인 경우에 델타들을 병합할 수 있다.

[0233] 에지 저장소는 각 t에 대해 G_0 및 ΔG_t(t> 0)를 개별적으로 유지할 수 있다. ΔG_t의 삽입 및 삭제 연산들이 별도

의 파일들에서 유지되므로, 실행 엔진은 에지 튜플의 다양성을 인식할 수 있다.

[0234] 각 파일은 CSR과 유사한 포맷으로 에지를 저장하므로, 실행 엔진은 초기 그래프 및 그래프 변형들에 동일하게 액세스할 수 있다. 문제는 삭제를 처리하는 방법이다. 즉, 실행 엔진이 G_t의 에지 스트림들을 스캔할 때, ΔG_τ ($\tau \leq t$)에 의해 삭제된 에지들은 보이지 않아야 한다. 그러나 디스크의 데이터에 대해 삭제를 빈번하게 수행하는 것은 비용이 많이 든다. 따라서, 메모리의 ΔG_t에서 삭제를 유지하고, 관련 디스크 페이지가 페이지 버퍼 풀에 로드될 때 해당 에지들을 삭제된 것으로 지연 표시할 수 있다.

[0235] 이하에서는 본 개시에 따른 그래프 처리 방법에 따른 효과를 실험 결과를 참조하여 설명한다.

[0236] 실험에 사용되는 사용된 데이터는 아래의 표 2와 같은 실제 그래프와 다양한 크기의 합성 그래프를 사용하였다.

표 2

[0237]

Dataset	|V|	|E|	Size
Twitter (TWT)	41.6 M	1.37 B	21.9 GB
GSH15 (GSH15)	988 M	33.9 B	542.4 GB
Clueweb12 (CW12)	6.3 B	66.8 B	1.06 TB
HyperLink (HL)	3.3 B	119 B	1.9 TB
$RMAT_X$ $(25 \leq X \leq 36)$	2^{X-4}	2^X	512 MB ~ 1 TB

[0238] 상술한 그래프에 대한 워크 부하는 그래프 데이터가 주어지면, 에지의 90%를 무작위로 균일하게 샘플링하여, 초기 그래프 G_0으로 사용하였다. 삽입 워크 부하 Δ^+G_i ($i > 0$)의 경우, 나머지 10%의 에지를 사용하였다.

[0239] 그리고 삭제 워크 부하 ΔG_i ($i > 0$)의 경우, G_{i1}의 에지를 무작위로 균일하게 샘플링하였다. 기본적으로, $|\Delta^+G_i| : |\Delta^-G_i| = 75 : 25$ [9] 및 $|\Delta G_i| = 100k$를 사용하였다. G_0에 대한 원-샷 쿼리의 실행 시간과 $i \in [1, 4]$ 인 G_i에 대한 4 개의 연속 증분 쿼리의 평균 실행 시간을 확인하였다.

[0240] 먼저, 평가된 시스템의 전체 성능을 실제 그래프와 비교하여 증분 그래프 분석에 대한 효율성을 설명한다. 본 개시에 따른 방식과 기존 방식(GraphBolt (GrB), Differential Dataflow (DD) 성능을 비교하였다.

[0241] 먼저, 단일 머신에서의 성능을 설명한다.

[0242] 표 3은 TWT에서의 실행시간을 나타낸다.

표 3

[0243]

Time [sec]	PR		LP	
	One-shot	Incremental	One-shot	Incremental
GrB	59.9	54.5	133.5	109.5
iTBGPP	53.2	23.8	139.6	29.8

[0244] 표 3을 참조하면, 원-샷 쿼리의 경우, 본 개시에 따른 방법(iT_{BGPP})은 계산과 입출력을 효율적으로 수행하는바, 디스크 입출력이 있는 경우에도 기존과 비슷한 성능을 발휘하는 것을 확인할 수 있다.

[0245] PR 및 LP의 증분 쿼리의 경우, 본 개시에 따른 방식은 증분 쿼리 방식에서는 불필요한 계산을 더 많이 제거하므로 기존 방식보다 뛰어남을 확인할 수 있다.

[0246] 기존 방식은 삽입되거나 삭제된 에지에서 시작하는 이웃 관계와 함께 그래프 변형의 전이적(transitive) 영향만 고려하므로 불필요한 세분화 계산을 수행하였다. 그러나 본 개시에 따른 방식은 정점 속성값의 실제 변경 사항을 추가로 고려하고 그 값이 이전 스냅 샷으로부터 변경되지 않으면 계산을 수행하지 않는다.

[0247] 그리고 기존 방식은 메모리의 모든 슈퍼스텝에 대해 대규모 정점 속성 어레이를 유지하지만 본 개시에 따른 방법은 메모리 부족 상태에서 속성값의 변경 사항만 보조 저장소에 델타로서 저장한다. 즉, 디스크 입출력과 메모

리 오버헤드를 상쇄시켰다.

[0248] 이하에서는 분산 시스템에서의 성능을 설명한다. 구체적으로, 25 대 머신 클러스터에서의 기존 방식과 본 개시에 따른 방법의 성능을 비교하였다.

[0249] 도 15는 다양한 그래프 데이터 처리 방법에 따른 처리 시간을 설명하기 위한 도면이다.

[0250] 도 15를 참조하면, 기존의 제1 방법(DD)은 중간 결과 크기가 작은 일부 워크 부하에 적합할 수 있다. 그러나 기존의 제1 방법(DD)은 조인(join)이 방대한 중간 결과를 생성할 수 있는 NGA에 대한 심각한 확장성 문제를 겪고 있다. 예를 들어, TC는 그래프 순회를 위해 에지 테이블에서 자체-조인을 수행하여 중간 결과 크기를 $O(|E|^2)$로 만든다. 그러나 본 개시에 따른 방법은 이러한 조인에 대한 중간 결과를 유지하지 않음으로써 증분 NGA의 확장 가능한 처리를 달성한다.

[0251] 도 15의 (a) 및 (b)는 각 방식에 따른 PR 및 LP의 실행 시간을 보여준다. 이를 참조하면, 본 개시에 따른 방식은 가장 큰 그래프로 확장되는 동안, 제1 방식인(DD)는 훨씬 작은 그래프에서 OOM 오류로 인해 충돌이 발생하여 정상 동작하지 못함을 확인할 수 있다.

[0252] 원-샷 쿼리의 경우, 기존의 제1 방식(DD)은 메시지를 다시 분할하는데 드는 네트워크 입출력 비용이 높기 때문에 실행 시간이 느리다. 모든 슈퍼스텝에서, 기존의 제1 방식(DD)은 정점 및 에지 테이블들과의 조인을 처리하여 메시지를 생성한다. 값 집계를 위한 메시지로 R_{EDUCE}를 처리하기 위해 메시지를 다시 분할한다. 특히, TWT에서 PR의 경우, 기존의 제1 방식(DD)의 네트워크 전송 크기는 본 개시의 따른 방식보다 16.3 배 더 크다. 이와 같이 본 개시에 따른 방식은 GSA에서 축적된 대수적 속성들을 활용하여 부분 사전 집계를 수행함으로써 네트워크 입출력 비용을 감소한다.

[0253] 그리고 기존의 제1 방식(DD)은 중간 결과를 유지하기 위해 메모리를 많이 사용하므로 TWT_5에서 디스크 스와프가 발생한다. TWT_5의 PR의 경우 25 대 이상의 머신에서 총 2.1TB의 메모리를 사용하며 이것은 입력 그래프 크기보다 24.4 배 더 큰 것이다.

[0254] 이와 같이 본 개시에 따른 방식은 원-샷 및 증분 쿼리를 모두 효율적으로 처리한다. 증분 계산에 의한 속도 향상은 PR의 경우 평균 5.3, LP의 경우 4.1이다. 증분 계산은 감소된 디스크 입출력에 의해서 실행 시간을 가속화한다. 예를 들어, HL에서 원-샷 쿼리에 비해 증분 쿼리의 디스크 입출력 바이트는 PR의 경우 16%, LP의 경우 56%로 줄어 든다.

[0255] 도 15의 (c) 및 (d)는 각각 WCC 및 BFS의 실행 시간을 보여준다. 전체적인 경향은 앞선 (a), (b)와 유사하다. 기존의 제1 방식(DD)은 제한된 효율성과 확장성을 나타낸다.

[0256] 본 개시에 따른 방식(iT_{BGPP})은 프로그램이 죽는 일 없이 가장 큰 그래프까지 효율적으로 처리함을 확인할 수 있다. 본 개시에 따른 방식은 WCC에 대해서는 평균 11.3, BFS의 경우 10.2의 속도 향상이 있으며, TWT에 대해서는 6.2 및 4.7로부터 WCC 및 BFS의 HL에서 각각 17.6 및 7.5로 속도 향상이 있다.

[0257] TWT_5에서 BFS의 경우 본 개시에 따른 방식은 기존 방식(DD)보다 0.12 초 더 느리다. 이것은 두 시스템의 설계 선택이 다르기 때문이다. 기존의 방식(DD)은 M_{IN} 집계에 대한 입력으로 정렬된 이전에 생성된 메시지를 유지 관리하므로 입력 그래프 크기보다 17.4 배 더 큰 1.4TB 힙 메모리 공간이 필요하다. 즉, 현재 최소값에 대한 메시지가 삭제되고 최소값이 변경되면 집계 값이 다음으로 큰 값으로 갱신된다. 그러나 본 개시에 따른 방식은 동일한 경우에 대해서 입력 메시지를 생성하여 M_{IN} 집계를 다시 계산해야 하며, 이과정에서 0.33 초의 추가 시간이 소요된다.

[0258] 그러나 본 개시에 따른 방식은 더 큰 그래프로 확장하는데 도움이 되는 반면 기존의 방식(DD)은 OOM 오류로 인해 실패한다.

[0259] 도 15 (e) 및 (f)는 각각 TC 및 LCC의 실행 시간을 나타낸다.

[0260] 기존의 방식은 가장 작은 TWT를 처리할 수 없어 OOM 오류가 발생한다. 기존의 방식은 더 빠른 증분 계산을 위해 중간 결과를 유지하지만 총 크기는 최대 $\Sigma_{v \in V} deg(v)^2$까지만 도달할 수 있다. 여기서 deg (v)는 정점 v의 차수이다.

[0261] 특히 TWT 및 HL의 제곱 합은 각각 199 조 및 3.6 경에 달하며, 이는 전체 중간 결과를 유지하는 것이 NGA에 적합하지 않음을 나타낸다. 반면에, 본 개시에 따른 방식은 충돌없이 가장 큰 그래프까지 처리함을 확인할 수 있다.

[0262] 이와 같이 증분 계산은 필요한 계산과 입출력을 줄일 수 있다. 예를 들어, HL에서 원-샷 쿼리에 비해 증분 쿼리에 대한 디스크 입출력 바이트는 TC의 경우 75.5 배, LCC의 경우 54 배 감소한다. clock_gettime()으로 측정된 총 CPU 시간은 TC의 경우 94 배, LCC의 경우 63.8 배 감소한다.

[0263] 이하에서는, 제한된 하드웨어 리소스로 큰 그래프를 처리하기 위해 시스템의 확장성을 도 16을 참조하여 설명한다.

[0264] 해당 실험은 먼저 25 대의 머신 클러스터로 RMAT 그래프의 크기를 변경하고, 그런 다음 고정 크기 그래프로 머신 수를 변경하였다.

[0265] 도 16은 RMAT 그래프의 크기에 따라 PR 및 TC의 실행 시간을 나타낸다.

[0266] 도 16을 참조하면, 본 개시에 따른 방식은 충돌없이 효율적으로 가장 큰 그래프로 확장됨을 확인할 수 있다. 그러나 기존의 방식은 모든 중간 결과를 메모리에 유지함으로써 확장성 문제를 보여준다. 구체적으로, PR 및 TC의 경우, DD는 입력 그래프 크기보다 26 배 및 1349 배 큰 2.9TB 및 4.9TB 힙 메모리 공간이 필요한 $RMAT_{33}$ 및 $RMAT_{28}$과의 디스크 스와프로 어려움이 발생하였다. 또한, 기존의 방식은 $RMAT_{34}$ 및 $RMAT_{29}$의 OOM 오류가 발생하였다.

[0267] 증분 계산의 이점은 그래프 크기가 지속적으로 커짐에 따라 증가하며, 이는 대규모 동적 그래프에 대한 증분 계산의 확장 가능한 처리에 대한 필요성을 보여준다. PR 및 TC의 경우 본 개시에 따른 방식의 증분 계산에 의한 속도 향상은 각각 평균 2.8 및 12.5이다. 가장 큰 그래프 $RMAT_{36}$의 경우 속도 향상이 각각 4.1 및 43.9까지 증가함을 확인할 수 있다.

[0268] 이후에는 머신 수를 변경한 경우의 성능을 도 17을 참조하여 설명한다.

[0269] 도 17을 설명함에 있어서, 기존의 방식은 모든 경우에서 OOM 오류가 발생하여, 그 실험 결과는 반영하지 않았다.

[0270] PR의 경우, 머신 수가 5 개에서 25 개로 증가함에 따라 본 개시에 따른 방식은 원-샷 및 증분 쿼리에 대해 각각 5.4 및 7.7 속도 향상을 달성한다. 증분 쿼리의 초 선형 속도 향상은 증분 쿼리가 그래프의 일부에만 액세스하고 더 많은 머신이 추가됨에 따라 해당 데이터가 클러스터 머신의 메모리에 맞춰져 디스크 입출력을 줄이기 때문이다.

[0271] TC의 경우, 본 개시에 따른 방식은 원-샷 및 증분 쿼리에 대해 각각 5.4 및 4.5 속도 향상을 발생함을 확인할 수 있다. 그리고 본 개시에 따른 방식은 중간 결과를 유지하지 않고 적은 수의 머신으로도 증분 NGA를 효율적으로 처리한다.

[0272] 이하에서는 TWT_{25}로 본 개시에 따른 방식의 성능에 대한 민감도를 설명한다. 먼저, 그래프 변형의 워크량을 다양화하여 다양한 알고리즘의 성능 특성을 연구하였다. 1) 삭제 수에 대한 삽입 수의 비율, 2) 스냅샷 당 그래프 변형의 수. 또한, 본 개시에서의 최적화 효과를 검토하였다.

[0273] 쿼리의 경우 집계 함수가 Abelian 그룹에 속하면, 삭제 후에도 재계산이 필요하지 않는다. 그러나 집계 함수가 Abelian monoid에 속하는 경우 삭제시 재계산이 필요할 수 있다. 이에 따라 삽입 횟수와 삭제 횟수의 비율을 변경하였다. $|\Delta^{+}G| : |\Delta^{-}G| = \{100:0, 75:25, 50:50, 25:75, 0:100\}$.

[0274] 도 18의 (a)는 삽입 전용 워크 부하의 실행 시간으로 정규화된 실행 시간을 보여준다. 여기서, PR 및 TC의 경우 집계 함수가 Abelian 그룹에 속하므로 재계산이 필요하지 않다. 따라서, 도 18(a)을 참조하면, PR과 TC는 다양한 비율에 대해 차이가 없다.

[0275] 그러나 WCC의 경우, 삭제시 재계산이 필요할 수 있는 Abelian monoid에 속하는 M_{IN} 누적기를 사용한다. 따라서 삭제량이 증가하면 실행 시간도 늘어난다. 유사한 결과를 확인할 수 있다.

[0276] 이하에서는 두 가지 측면에서 배치(batch) 크기에 따른 성능을 설명한다. (1) 메모리 부족 문제없이 큰 배치 크

기로 확장 가능한 처리 및 (2) 배치 내에서 계산 및 입출력을 공유하여 대규모 배치 크기에 대한 처리량 증가 .

[0277] 이를 위하여, 각 ΔG, |ΔG| = {100, 1k, 10k, 100k, 1M}에서 그래프 변형의 수를 변경하여 실험을 수행하였다. 모든 그래프 변형에 대해 증분 계산을 실행하는 것은 낭비일 수 있다. 따라서, 여러 그래프 변형을 일괄 처리하며, 본 개시에 따른 방식은 분석 결과를 새로 고치는 처리량을 크게 늘릴 수 있다. 도 18(b)는 |ΔG| = 100를 사용한 처리량으로 정규화된 처리량을 보여준다. 처리량 (초당 변경 수)은 |ΔG|를 실행 시간으로 나눈 값이다.

[0278] 도 18(b)을 참조하면, 배치 크기가 100에서 1M으로 증가함에 따라, 처리량은 PR, WCC 및 TC에 대해 각각 43894, 26782 및 101303 배 증가함을 확인할 수 있다.

[0279] 이하에서는 본 개시에 따른 최적화의 효과를 설명한다.

[0280] 선택한 최적화는 트래버셜 리오더링(TR), 이웃 가지치기(NP), 시크/윈도우 쉐어링(SWS), C_{NT}(Min with counting) 및 비용 기반 델타 유지 관리(C_{OST})이다.

[0281] 먼저, 다중 홉 NGA의 예로 TC 및 LCC에 대한 최적화(TR, NP 및 SWS)의 효과를 평가한다.

[0282] 도 19는 다양한 최적화를 위한 실행 시간을 보여준다. 모든 최적화가 비활성화된 방법을 B_{ASE}로 표시한다.

[0283] TC의 경우 B_{ASE} 가 증분 계산을 수행하지만 원-샷 쿼리보다 시간이 더 오래 걸린다. B_{ASE}에서 4 개의 하위 쿼리에 대한 중복 및 반복 입출력으로 인해 입출력에 병목 현상이 발생한다. TR은 B_{ASE}에 비해 네트워크 입출력을 49% 줄이지만 원-샷 쿼리보다 29% 빠르다. TR과 NP를 모두 적용하면 많은 중복 입출력을 피하고 원-샷 쿼리에 비해 13.1 배의 속도 향상을 달성할 수 있다. 마지막으로, 입출력을 공유하여 하위 쿼리를 일괄 처리하는 SWS를 적용함으로써 iT_{BGPP}는 28.9 배의 속도 향상을 달성한다. TR 및 NP는 중복 계산 및 입출력 비용을 효과적으로 줄이고 SWS는 입출력을 공유하여 입출력 비용을 더욱 줄이고 병렬 처리 수준을 증가시킨다.

[0284] LCC의 경우, 짝수 B_{ASE} 는 원-샷 쿼리보다 2.2 배 빠르다. 이는 원샷 쿼리에 대한 LCC의 CPU 계산 비용이 상당히 높기 때문이다. 증분 쿼리에서 병목 현상은 TC와 유사하게 입출력으로 변경된다. 모든 최적화를 적용함으로써 LCC는 52.7 배의 속도 향상을 달성한다.

[0285] 이하에서는 M_{IN}에 대한 증분 누적에 대한 최적화(C_{NT})의 효과를 설명한다.

[0286] 도 18(b)는 다른 모든 최적화가 활성화된 상태에서 C_{NT} 최적화를 적용할 때 달성된 속도 향상을 나타낸다. 삽입 전용 워크 부하(100 : 0)에서, WCC의 경우 2.4, BFS의 경우 1.4 속도 향상이 있음을 확인할 수 있다. 삽입 전용 워크 부하에서도 재계산이 발생할 수 있다. 예를 들어, 정점 v가 G_t의 슈퍼스텝 s에서 수렴했다고 가정한다. ΔG_{t+1}에 의한 약간의 삽입 후, v는 G_{t+1}의 슈퍼스텝 s′ (<s)에서 더 빠르게 수렴할 수 있다. 그 다음, 이전에 v에서 슈퍼스텝 s′′ (s′ < s′′ ≤ s)에서 이웃 항목으로 보낸 업데이트를 삭제해야 한다. 워크 부하에 더 많은 삭제가 포함됨에 따라 속도 향상은 WCC 및 BFS에 대해 각각 최대 3.2 및 2까지 증가한다.

[0287] 마지막으로, PR 및 LP에 대한 기술 효과(C_{OST})를 평가하고 1) 델타를 병합하지 않는 N_OM_{ERGE}와 2) 주기적으로 델타를 병합하는 $P_{ERIODIC}M_{ERGE}$ 두 가지 다른 접근 방식에 대해 테스트하였다. 구체적으로, 50 개의 스냅 샷을 그 기간 동안 사용한다.

[0288] 도 19는 i ∈ [1, 100]에 대한 |ΔG_i| = 1M을 사용한 최대 100 개의 스냅 샷에 대한 세 가지 접근 방식의 실행 시간을 보여준다. N_OM_{ERGE}의 실행 시간은 델타가 계속 누적됨에 따라 빠르게 증가한다. $P_{ERIODIC}M_{ERGE}$는 차선의 성능을 보여준다. 50번째 스냅 샷까지는 N_OM_{ERGE}와 함께 실행 시간이 늘어난다. 본 개시에 따른 방식에 따른 비용 기반 전략(C_{OST})은 많은 스냅 샷에서 일관된 실행 시간을 달성할 수 있다.

[0289] 이와 같이 본 개시에 따른 방식은 데이터 증분에 따라 중간 데이터를 저장하지 않는바, 데이터 입출력 측면에서 유리하면, 증분 쿼리의 경우 기존보다 빠른 실행이 가능하다.

[0291] 도 21은 본 개시의 일 실시 예에 따른 그래프 데이터 처리 방법을 설명하기 위한 흐름도이다.

[0292] 도 21을 참조하면, 먼저 동적 그래프를 생성하여 저장할 수 있다(S2110). 이러한 동적 그래프는 복수의 정점과 엣지를 포함하는 그래프 데이터와 그래프 데이터에 대한 변경된 증분 정보로 구성될 수 있다.

[0293] 그리고 그래프 데이터에 대한 쿼리를 수신하면, 쿼리에 대응되는 출력 데이터를 생성하기 위한 그래프 처리 동작을 수행할 수 있다(S2120). 구체적으로, 수신한 쿼리에 기초하여 그래프 데이터에 적용할 원샷 쿼리 플랜 정보와 증분 정보에 적용할 증분 쿼리 플랜 정보를 생성하고, 생성한 원샷 쿼리 플랜 정보를 그래프 데이터에 반영하여 제1 중간 데이터를 생성하고, 생성한 증분 쿼리 플랜 정보를 증분 정보에 반영하여 제2 중간 데이터를 생성하고, 제1 중간 데이터 및 제2 중간 데이터를 이용하여 결과 데이터를 생성할 수 있다. 이때, 기설정된 크기의 정점 스트림과 에지 스트림의 쌍이 저장된 영역을 로드하고, 로드된 정점 스트림 및 에지 스트림에 대응되는 원샷 쿼리 플랜 정보 및 증분 쿼리 플랜 정보 내의 워크를 일괄적으로 수행할 수 있다.

[0294] 그리고 출력 데이터가 쿼리를 전송한 장치에 전송할 수 있다(S2130).

[0295] 한편, 본 개시에 따른 처리 방법은 증분 정보의 크기가 기설정된 크기 이상이면, 증분 정보를 그래프 데이터에 반영하여 그래프 데이터를 업데이트할 수 있다.

[0296] 이상과 같이 본 실시 예에 따른 그래프 데이터 처리 방법은 데이터 증분에 따라 중간 데이터를 저장하지 않는바, 데이터 입출력 측면에서 유리하면, 증분 쿼리의 경우 기존보다 빠른 실행이 가능하다.

[0297] 한편, 상술한 다양한 실시 예에 따른 그래프 데이터 처리 방법은 각 단계들을 수행하기 위한 프로그램 코드 형태로 구현되어, 기록 매체에 저장되고 배포될 수도 있다. 이 경우, 기록 매체가 탑재된 장치는 상술한 암호화 방법 등의 동작들을 수행할 수 있다.

[0298] 이러한 기록 매체는, ROM, RAM, 메모리 칩, 메모리 카드, 외장형 하드, 하드, CD, DVD, 자기 디스크 또는 자기 테이프 등과 같은 다양한 유형의 컴퓨터 판독 가능 매체가 될 수 있다.

[0299] 이상 첨부 도면을 참고하여 본 개시에 대해서 설명하였지만 본 개시의 권리범위는 후술하는 특허청구범위에 의해 결정되며 전술한 실시 예 및/또는 도면에 제한되는 것으로 해석되어서는 안 된다. 그리고 특허청구범위에 기재된 개시의, 당업자에게 자명한 개량, 변경 및 수정도 본 개시의 권리범위에 포함된다는 점이 명백하게 이해되어야 한다.

부호의 설명

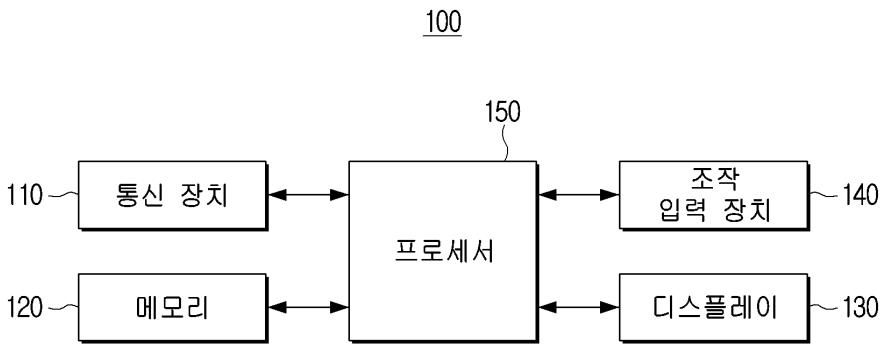
[0300] 100: 전자 장치 110: 통신 장치

120: 메모리 130: 디스플레이

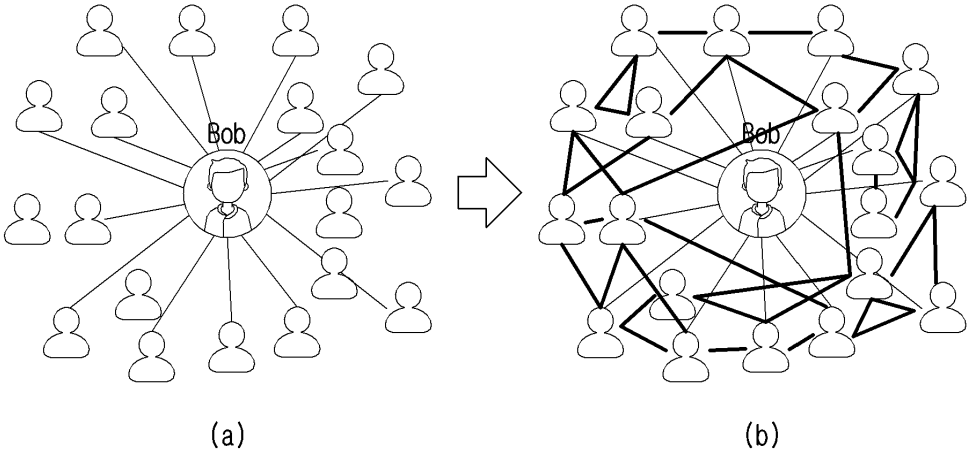
140: 조작 입력 장치 150: 프로세서

도면

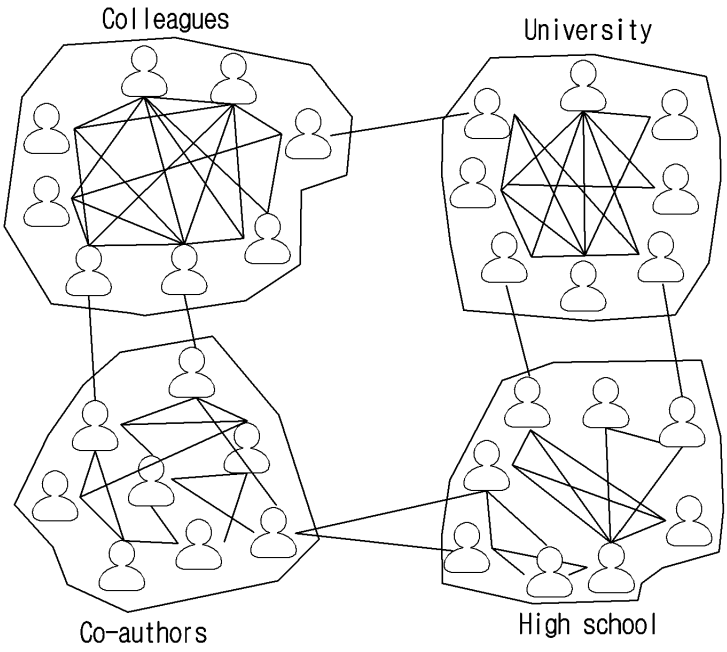
도면1



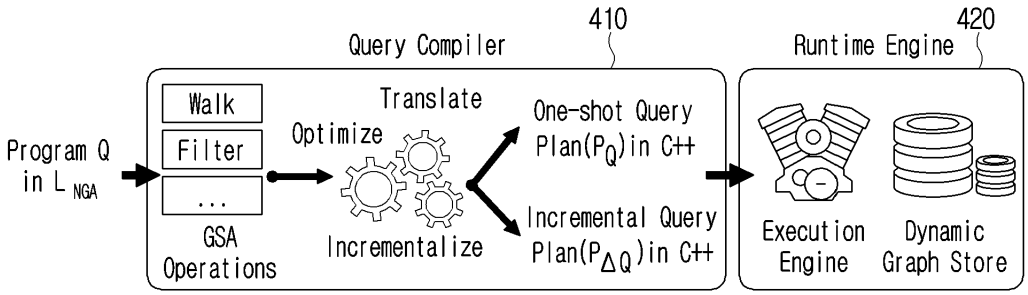
도면2


Bob
Bob
(a)
(b)

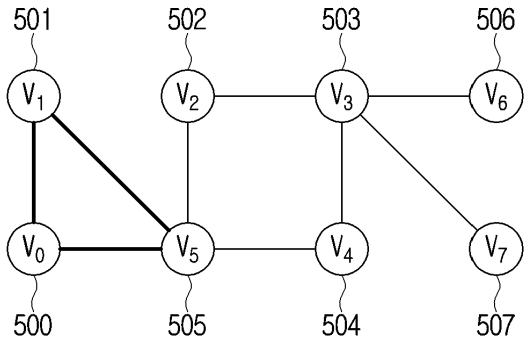
도면3


Colleagues
University
Co-authors
High school

도면4


410
Query Compiler
420
Runtime Engine
Program Q
in L NGA
Walk
Filter
...
GSA
Operations
Translate
Optimize
Incrementalize
One-shot Query
Plan(PQ)in C++
Incremental Query
Plan(PΔQ)in C++
Execution
Engine
Dynamic
Graph Store

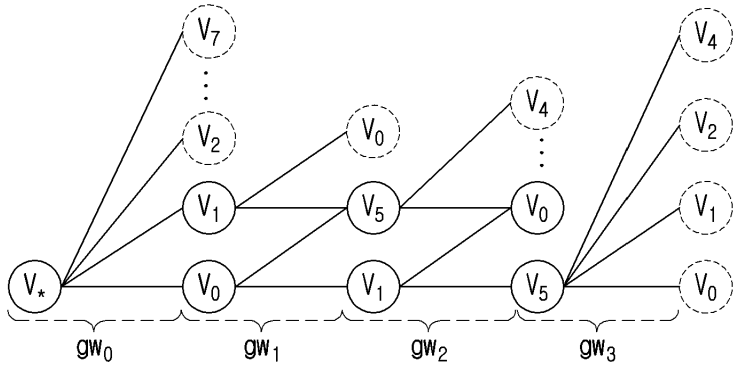
도면5



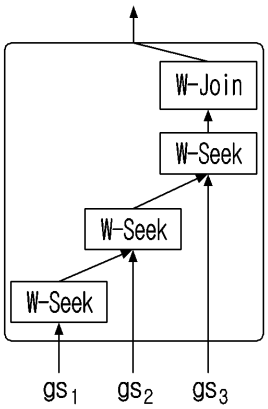
도면6

vs	es
<V_0, 2>	<V_0, V_1>
<V_1, 2>	<V_0, V_5>
...	<V_1, V_0>
	<V_1, V_5>
	...

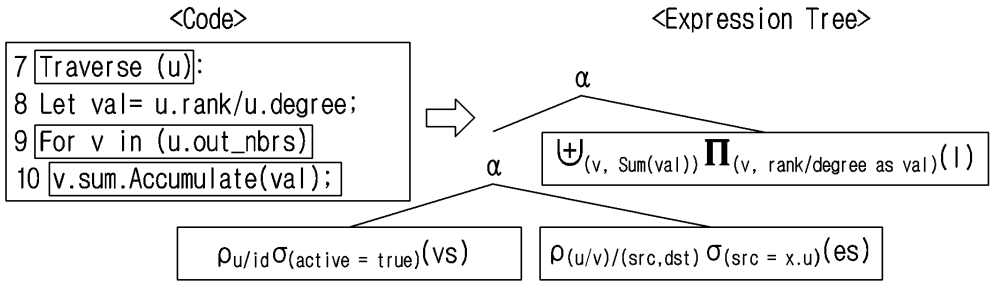
도면7



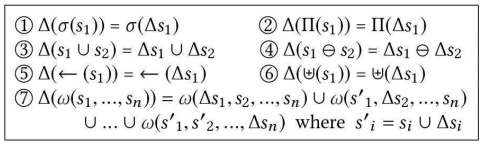
도면8



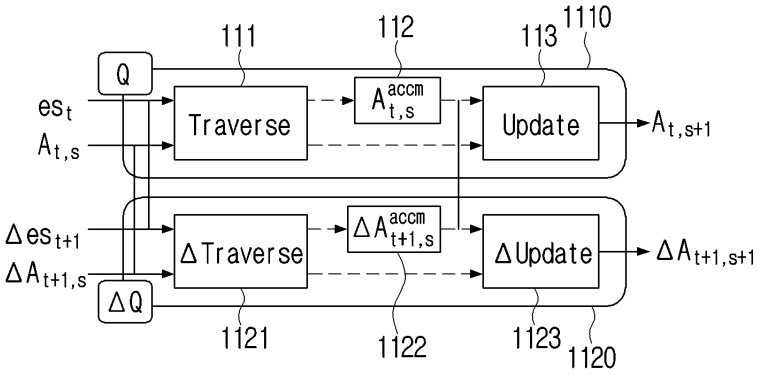
도면9



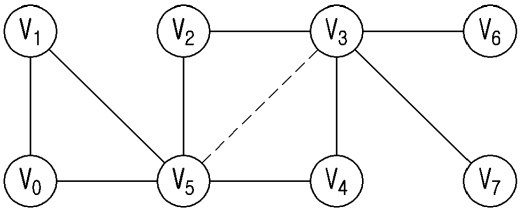
도면10



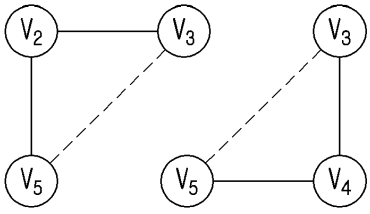
도면11



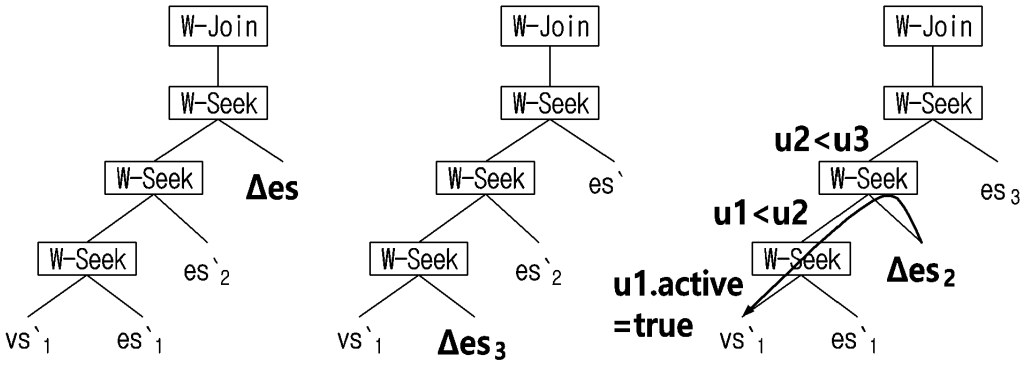
도면12



도면13

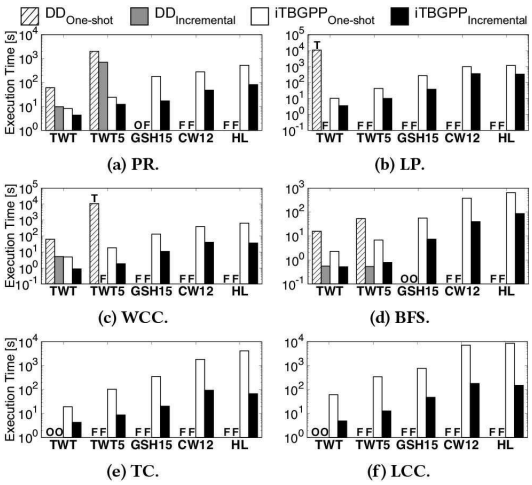


도면14



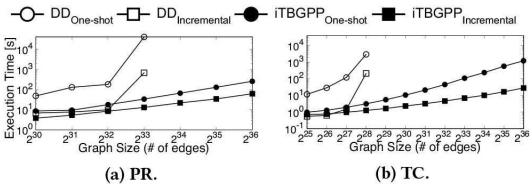
(a) q4. (b) reordered q4. (c) q3.

도면15



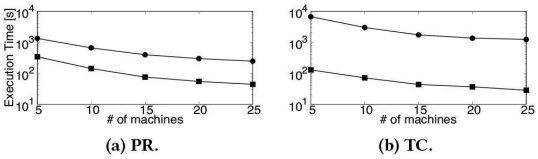
(a) PR. (b) LP. (c) WCC. (d) BFS. (e) TC. (f) LCC.

도면16



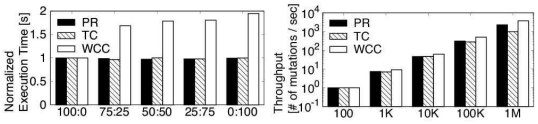
(a) PR. (b) TC.

도면17



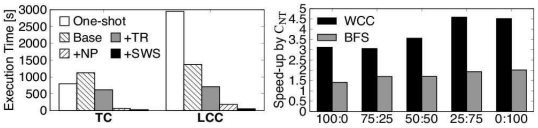
(a) PR. (b) TC.

도면18



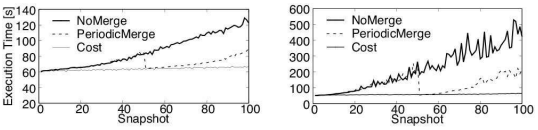
(a) Varying $|\Delta^{+}G| : |\Delta^{-}G|$. (b) Varying $|\Delta G|$.

도면19



(a) TC and LCC. (b) WCC and BFS.

도면20



도면21