(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2024년08월05일
(11) 등록번호 10-2690965
(24) 등록일자 2024년07월29일

(51) 국제특허분류(Int. Cl.)
*G06F 16/909* (2019.01) *G06F 16/23* (2019.01)
*G06F 16/29* (2019.01) *G06F 16/901* (2019.01)
*G06F 16/903* (2019.01) *H04W 4/02* (2018.01)
(52) CPC특허분류
*G06F 16/909* (2019.01)
*G06F 16/2308* (2019.01)
(21) 출원번호 10-2023-0011981
(22) 출원일자 2023년01월30일
심사청구일자 2023년01월30일
(56) 선행기술조사문헌
KR1020200024074 A*
KR1020080071228 A*
KR101502871 B1
KR1020190057827 A
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
**고려대학교 산학협력단**
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
**정연돈**
서울특별시 중구 청계천로 400, 105-812
**최달수**
경기도 수원시 영통구 봉영로1517번길 73, 933동 1102호
(74) 대리인
**김등용**

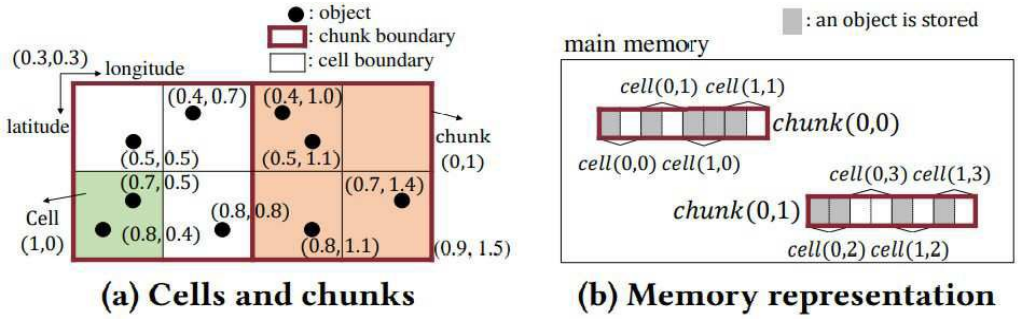
전체 청구항 수 : 총 10 항 심사관 : 박승철

(54) 발명의 명칭 **자율 운영 그리드 인덱스를 위한 시스템 및 방법**

## (57) 요 약

움직이는 물체를 위한 인메모리 그리드 인덱스 및 인덱스를 재구성하는 방법 등이 개시된다. 적어도 프로세서와 메모리를 포함하는 컴퓨팅 장치에 의해 수행되는 그리드 인덱스 대체 방법은, 사용자 쿼리를 처리하는 단계, 그리드 인덱스를 정의하는 놉 세팅(knob setting)을 결정하는 단계, 및 기존의 그리드 인덱스에 포함되는 물체들 각각을 상기 놉 세팅에 대응하는 새로운 그리드 인덱스로 전이하는 단계를 포함하고, 상기 전이하는 단계는 상기 새로운 그리드 인덱스로 전이하는 과정 중에 발생하는 사용자 쿼리를 처리한다.

*대 표 도* - 도1



(52) CPC특허분류
*G06F 16/235* (2019.01)
*G06F 16/29* (2019.01)
*G06F 16/901* (2019.01)
*G06F 16/903* (2019.01)
*H04W 4/02* (2020.05)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711169469
과제번호 2020R1A2C2013286
부처명 과학기술정보통신부
과제관리(전문)기관명 한국연구재단
연구사업명 개인기초연구(과기정통부)
연구과제명 대규모 분산 딥러닝을 위한 인메모리 텐서 데이터베이스
기 여 율 1/3
과제수행기관명 고려대학교 산학협력단
연구기간 2022.03.01 ~ 2023.02.28
이 발명을 지원한 국가연구개발사업
과제고유번호 1345347378
과제번호 2021R1A6A1A13044830
부처명 교육부
과제관리(전문)기관명 한국연구재단
연구사업명 이공학학술연구기반구축
연구과제명 융합소프트웨어연구소
기 여 율 1/3
과제수행기관명 고려대학교 산학협력단
연구기간 2022.03.01 ~ 2023.02.28
이 발명을 지원한 국가연구개발사업
과제고유번호 1711193663
과제번호 2020-0-01819-004
부처명 과학기술정보통신부
과제관리(전문)기관명 정보통신기획평가원
연구사업명 정보통신방송혁신인재양성
연구과제명 ICT명품인재양성(고려대학교)
기 여 율 1/3
과제수행기관명 고려대학교산학협력단
연구기간 2023.01.01 ~ 2023.12.31
공지예외적용 : 있음

# 명 세 서

## 청구범위

### 청구항 1

적어도 프로세서와 메모리를 포함하는 컴퓨팅 장치에 의해 수행되는 그리드 인덱스 대체 방법에 있어서,

사용자 쿼리를 처리하는 단계;

기존의 그리드 인덱스인 제1 그리드 인덱스를 대체하는 제2 그리드 인덱스를 정의하는 놉 세팅(knob setting)을 결정하는 단계; 및

상기 제1 그리드 인덱스에 포함되는 물체들 각각을 상기 놉 세팅에 대응하는 상기 제2 그리드 인덱스로 전이하는 단계를 포함하고,

상기 전이하는 단계는, 상기 제2 그리드 인덱스로 전이하는 과정 중에 발생하는 사용자 쿼리를 처리하고,

상기 놉 세팅은,

위도 방향의 셀(cell)의 개수, 경도 방향의 셀의 개수, 하나의 셀에 저장할 수 있는 물체의 최대 개수, 하나의 청크(chunk)에 포함되는 위도 방향의 셀 개수, 및 하나의 청크에 포함되는 경도 방향의 셀 개수를 포함하는,

그리드 인덱스 대체 방법.

### 청구항 2

제1항에 있어서,

상기 사용자 쿼리는, 움직이는 물체에 대하여, 그리드 인덱스로의 삽입에 대한 요청인 삽입 쿼리, 그리드 인덱스로부터의 삭제에 대한 요청인 삭제 쿼리, 그리드 인덱스 내에서 소정의 범위 내에 존재하는 물체 대한 검색 요청인 레인지 쿼리, 및 소정의 범위 내에서 특정 좌표와 가장 가까운 k(k는 임의의 자연수) 개의 물체에 대한 검색 요청인 k-NN 쿼리 중 어느 하나인,

그리드 인덱스 대체 방법.

### 청구항 3

제1항에 있어서,

청크는 상기 컴퓨팅 장치에 포함되는 메인 메모리 내에서 시리얼 메모리 할당(serial memory allocation)의 단위인,

그리드 인덱스 대체 방법.

### 청구항 4

제1항에 있어서,

상기 전이하는 단계와 병렬적으로 수행되는,

사용자 쿼리에 대한 보상을 획득하는 단계; 및

획득된 보상을 이용하여 놉 세팅 결정 모델을 업데이트하는 단계를 더 포함하는,

그리드 인덱스 대체 방법.

### 청구항 5

적어도 프로세서와 메모리를 포함하는 컴퓨팅 장치에 의해 수행되는 그리드 인덱스 대체 방법에 있어서,

사용자 쿼리를 처리하는 단계;

그리드 인덱스 재구축 여부를 결정하는 단계; 및

그리드 인덱스를 재구축하기로 결정된 경우, 기존의 그리드 인덱스인 제1 그리드 인덱스에 포함되는 물체들 각각을 결정된 놉 세팅에 대응하는 새로운 그리드 인덱스인 제2 그리드 인덱스로 전이하는 단계를 포함하고,

상기 전이하는 단계는, 상기 제2 그리드 인덱스로 전이하는 과정 중에 발생하는 사용자 쿼리를 처리하고,

상기 놉 세팅은,

위도 방향의 셀(cell)의 개수, 경도 방향의 셀의 개수, 하나의 셀에 저장할 수 있는 물체의 최대 개수, 하나의 청크(chunk)에 포함되는 위도 방향의 셀 개수, 및 하나의 청크에 포함되는 경도 방향의 셀 개수를 포함하는,

그리드 인덱스 대체 방법.

**청구항 6**

제5항에 있어서,

상기 결정하는 단계에서, 그리드 인덱스를 재구축하지 않기로 결정된 경우, 상기 사용자 쿼리를 처리하는 단계를 재차 수행하는,

그리드 인덱스 대체 방법.

**청구항 7**

제5항에 있어서,

상기 그리드 인덱스 재구축 여부를 결정하는 단계는,

그리드 인덱스를 정의하는 놉 세팅을 결정하는 단계; 및

결정된 놉 세팅을 기존의 놉 세팅과 비교하는 단계를 포함하는,

그리드 인덱스 대체 방법.

**청구항 8**

제7항에 있어서,

상기 그리드 인덱스 재구축 여부를 결정하는 단계는,

결정된 놉 세팅과 기존의 놉 세팅이 동일한 경우 그리드 인덱스를 재구축하지 않기로 결정하고, 결정된 놉 세팅과 기존의 놉 세팅이 동일하지 않은 경우 그리드 인덱스를 재구축하기로 결정하는,

그리드 인덱스 대체 방법.

**청구항 9**

제5항에 있어서,

상기 사용자 쿼리는, 움직이는 물체에 대하여, 그리드 인덱스로의 삽입에 대한 요청인 삽입 쿼리, 그리드 인덱스로부터의 삭제에 대한 요청인 삭제 쿼리, 그리드 인덱스 내에서 소정의 범위 내에 존재하는 물체 대한 검색 요청인 레인지 쿼리, 및 소정의 범위 내에서 특정 좌표와 가장 가까운 k(k는 임의의 자연수) 개의 물체에 대한 검색 요청인 k-NN 쿼리 중 어느 하나인,

그리드 인덱스 대체 방법.

**청구항 10**

제5항에 있어서,

청크는 상기 컴퓨팅 장치에 포함되는 메인 메모리 내에서 시리얼 메모리 할당(serial memory allocation)의 단위인,

그리드 인덱스 대체 방법.

## 발명의 설명

### 기 술 분 야

[0001] 본 발명은 자율 운영 그리드 인덱스를 위한 시스템 및 방법에 관한 것으로, 보다 구체적으로는, 움직이는 물체에 대해 인메모리 환경에 적합한 새로운 그리드 인덱스, 기존의 그리드 인덱스를 새로운 그리드 인덱스로 대체하는 장치 및 방법, 새로운 인덱스를 위한 놉 세팅을 결정하는 장치 및 방법에 관한 것이다.

### 배 경 기 술

[0002] 움직이는 물체(moving objects)를 기반으로 하는 많은 위치 기반 서비스들(Location-Based Services, LBS)이 개발 및 상용화되고 있다. 예를 들면, 승차 공유 서비스(ride-sharing services), 마이크로 모빌리티 서비스(micro-mobility service), 내비게이션 및 교통 관제 시스템(navigation and traffic management systems), 배달 서비스(delivery services), 및 자율 주행(autonomous driving)은 모두 움직이는 물체를 기반으로 한다. 따라서, 움직이는 물체에 대한 효율적인 관리는 매우 중요하다.

[0003] 움직이는 물체로부터 발생하는 많은 위치 업데이트(location-update)를 빠르게 처리하면서, 위치 기반 서비스에 필요한 정보 또한 빠르게 탐색할 수 있어야 한다. 이러한 요구사항으로부터, 움직이는 물체를 위한 새로운 인메모리 그리드 인덱스 시스템(와플(Waffle)이라 명명함)을 제안한다. 제안하는 그리드 인덱스 시스템(와플)은 지리적인 공간을 고정된 크기의 셀들로 분할한다. 효율적인 쿼리 처리를 위해, 와플은 청크들(각 청크는 이웃하는 셀들로 구성됨)을 형성한다. 이와 같은 와플 인덱스는 놉들(knobs)에 의해 정의된다. 놉 설정은 와플의 성능에 매우 큰 영향을 미치고, 적절한 설정은 물체들이 이동함에 따라 변화한다. 따라서, 본 발명에서는, 놉 값들을 결정할 뿐만 아니라 놉 값들을 변경할 시점을 결정하는 기법을 제안한다. 새롭게 결정된 놉 값들이 주어졌을 때, 기존의 인덱스를, 사용자 쿼리에 대한 블락 없이, 새로운 인덱스로 대체할 수 있다.

### 선행기술문헌

#### 특허문헌

[0004] (특허문헌 0001) 대한민국 등록특허 제2172236호 (2020.10.30. 공고)

(특허문헌 0002) 대한민국 공개특허 제2019-0069229호 (2019.06.19. 공개)

(특허문헌 0003) 대한민국 등록특허 제2325741호 (2021.11.16. 공고)

## 발명의 내용

### 해결하려는 과제

[0005] 본 발명이 이루고자 하는 기술적인 과제는, 움직이는 물체에 대해 인메모리 환경에 적합한 새로운 그리드 인덱스 구조, 사용자 질의를 계속 처리하면서 기존의 인덱스를 새로운 인덱스로 대체하는 장치 및 방법, 및 인덱스 설정을 강화 학습 기반으로 자동 튜닝하는 장치 및 방법을 제공하는 것이다.

### 과제의 해결 수단

[0006] 본 발명의 일 실시예에 따른 그리드 인덱스 대체 방법은 적어도 프로세서와 메모리를 포함하는 컴퓨팅 장치에 의해 수행되고, 사용자 쿼리를 처리하는 단계, 그리드 인덱스를 정의하는 놉 세팅(knob setting)을 결정하는 단계, 및 기존의 그리드 인덱스에 포함되는 물체들 각각을 상기 놉 세팅에 대응하는 새로운 그리드 인덱스로 전이하는 단계를 포함하고, 상기 전이하는 단계는, 상기 새로운 그리드 인덱스로 전이하는 과정 중에 발생하는 사용자 쿼리를 처리한다.

[0007] 본 발명의 다른 실시예에 따른 그리드 인덱스 대체 방법은 적어도 프로세서와 메모리를 포함하는 컴퓨팅 장치에 의해 수행되고, 사용자 쿼리를 처리하는 단계, 그리드 인덱스 재구축 여부를 결정하는 단계, 및 그리드 인덱스를 재구축하기로 결정된 경우, 기존의 그리드 인덱스에 포함되는 물체들 각각을 결정된 놉 세팅에 대응하는 새로운 그리드 인덱스로 전이하는 단계를 포함하고, 상기 전이하는 단계는, 상기 새로운 그리드 인덱스로 전이하

는 과정 중에 발생하는 사용자 쿼리를 처리한다.

## 발명의 효과

[0008] 본 발명의 실시예들에 의할 경우, 움직이는 물체에 대해 인메모리 환경에 적합한 새로운 그리드 인덱스 구조를 제공할 수 있다.

[0009] 또한, 사용자의 질의에 대한 지속적인 처리를 수행하는 도중에, 기존의 인덱스를 새로운 인덱스로 대체할 수 있다.

[0010] 또한, 인덱스의 설정 및 변경 여부에 대하여 강화 학습을 기반으로 자동으로 결정할 수 있는 효과가 있다.

## 도면의 간단한 설명

[0011] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1은 제안하는 그리드 인덱스로써, (a)는 셀과 청크를 도시하고, (b)는 메모리 표현을 도시한다.

도 2는 삽입 쿼리 처리를 설명하기 위한 예시를 도시하는 도면이다.

도 3은 삭제 쿼리 처리를 설명하기 위한 예시를 도시하는 도면이다.

도 4는 k-NN 쿼리 처리를 설명하기 위한 예시를 도시하는 도면이다.

도 5는 $nCell^{space}$의 예시적인 효과를 설명하기 위한 도면이다.

도 6은 $nCell^{space}$의 다른 예시적인 효과를 설명하기 위한 도면이다.

도 7은 $nCell^{chunk}$의 예시적인 효과를 설명하기 위한 도면이다.

도 8은 MOPC의 예시적인 효과를 설명하기 위한 도면이다.

도 9는 리그리드(regrid)를 위한 스케줄의 예시를 도시한다.

도 10은 모델 수렴 전의 와플메이커의 워크플로우를 도시한다.

도 11은 모델 수렴 후의 와플메이커의 워크플로우를 도시한다.

도 12는 본 발명의 일 실시예에 따른 그리드 인덱스 대체 방법을 설명하기 위한 흐름도이다.

도 13은 본 발명의 다른 실시예에 따른 그리드 인덱스 대체 방법을 설명하기 위한 흐름도이다.

## 발명을 실시하기 위한 구체적인 내용

[0012] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시예들에 대해서 특정한 구조적 또는 기능적 설명들은 단지 본 발명의 개념에 따른 실시예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시예들은 다양한 형태들로 실시될 수 있으며 본 명세서에 설명된 실시예들에 한정되지 않는다.

[0013] 본 발명의 개념에 따른 실시예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시예들을 도면에 예시하고 본 명세서에서 상세하게 설명하고자 한다. 그러나, 이는 본 발명의 개념에 따른 실시예들을 특정한 개시 형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물, 또는 대체물을 포함한다.

[0014] 제1 또는 제2 등의 용어는 다양한 구성 요소들을 설명하는데 사용될 수 있지만, 상기 구성 요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만, 예컨대 본 발명의 개념에 따른 권리 범위로부터 벗어나지 않은 채, 제1 구성 요소는 제2 구성 요소로 명명될 수 있고 유사하게 제2 구성 요소는 제1 구성 요소로도 명명될 수 있다.

[0015] 어떤 구성 요소가 다른 구성 요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성 요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성 요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성 요소가 다른 구성 요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는 중간에 다른 구성 요소가 존재하지 않는 것으로 이해되어야 할 것이다. 구성 요소들 간의

관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0016] 본 명세서에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로서, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 본 명세서에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0017] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0018] 이하, 본 명세서에 첨부된 도면들을 참조하여 본 발명의 실시예들을 상세히 설명한다. 그러나, 특허출원의 범위가 이러한 실시예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한 부재를 나타낸다.

[0019] 움직이는 물체를 기반으로 하는 많은 위치 기반 서비스가 개발 및 상용화되고 있다. 예를 들면, 승차 공유 서비스(ride-sharing services, 예컨대 Uber and Lyft), 마이크로 모빌리티 서비스(micro-mobility service, 예컨대 Lime and Bird), 내비게이션 및 교통 관제 시스템(navigation and traffic management systems, 예컨대 Google Maps and Waze), 배달 서비스(delivery services, 예컨대 Grubhub, DoorDash, Uber eats, and Postmates), 및 자율 주행(autonomous driving, 예컨대 Tesla)은 모두 움직이는 물체를 기반으로 한다. 따라서, 움직이는 물체에 대한 효율적인 관리는 매우 중요하다.

[0020] 움직이는 물체의 주된 특징은 공간 상의 위치가 계속 변한다는 것이다. 따라서, 빠른 위치 업데이트를 지원하는 것이 필요하다. 또한, 레인지 쿼리(range query)와 k-NN 쿼리를 포함하는 스캔 쿼리(scan queries)에 대한 빠른 처리도 중요하다.

[0021] 본 발명에서는, 와플(Waffle)이라 명명되는 신규한 인메모리 그리드 인덱스 시스템(in-memory grid indexing system)을 제안한다. 물체의 그리드 좌표(grid coordinate)는 그리드 정의(grid definition)에 기초하여 쉽게 계산될 수 있고, 이는 해시 데이터 구조(hash data structure)에서 타겟 버킷(target bucket)을 찾는 것과 유사하다. 이 특징(feature)은 효율적인 위치 업데이트 쿼리 처리의 근간이 된다. 또한 빠른 응답 시간을 제공하기 위해 인메모리 인덱스를 구성한다.

[0022] 와플 인덱스(Waffle index)는 와플(Waffle)의 성능에 매우 큰 영향을 미치는 복수의 놉들(knobs)로 정의된다. 물체 분포(object distribution), 쿼리 워크로드(query workload), 하드웨어 구성(hardware configuration) 등에 영향을 받기 때문에 적절한 놉 값(knob values)을 설정(configuration)하는 것은 어렵다.

[0023] 본 발명에서는, 적절한 놉 값을 자동으로 설정할 수 있는 튜닝 시스템(WaffleMaker로 명명될 수 있음. 실시예에 따라, WaffleMaker는 Waffle의 일부 구성으로 이해될 수도 있음)을 제안한다. 와플메이커(WaffleMaker)는 강화학습(reinforcement learning)에 기반하고, 사전학습(pre-training)이 요구되지 않는다. 와플메이커(WaffleMaker)는 다음 과정을 모델링한다. 와플메이커(WaffleMaker)는 현재 물체 분포를 관찰하고, 놉 세팅(knob setting, 놉 설정이라 명명될 수도 있음)을 결정하며, 놉 세팅에 의해 정의된 와플 인덱스에 대한 성능 결과(performance results)를 획득한다. 이러한 과정을 반복함으로써, 와플메이커는 더 나은 성능을 갖는 놉 세팅을 출력한다.

[0024] 와플메이커에 의해 자동적으로 결정된 새로운 놉 세팅이 주어질 때마다, 와플은 와플 인덱스를 재정의하며, 이는 리그리드(regrid)라 명명될 수 있다. 와플 인덱스를 재구축하는 동안, 사용자 쿼리를 처리하는 메커니즘이 없다면, 쿼리 처리는 블락되고(blocked), 물체의 개수가 증가한다면 블락된 시간 또한 증가한다. 와플은 동시성 제어 기법(concurrency control scheme)을 이용하여 이러한 문제점을 해결한다.

[0025] 제안하는 발명의 효과는 다음과 같다. 첫째, 움직이는 물체에 대하여 와플(Waffle)이라 불리는 새로운 인메모리 그리드 인덱스 시스템을 제안한다. 셀(cells)과 청크(chunks)의 개념에 기초하여 메인 메모리에 조직화하는 구체적인 방법은 후술한다. 둘째, 와플메이커(WaffleMaker)라 불리는 신규한 강화 학습 기반의 설정 튜닝 시스템

을 제안한다. 셋째, 사용자 쿼리를 블락(blocking)하지 않고 그리드 인덱스를 재정의하기 위한 신규한 워크 플로우 모델을 제안한다.

[0026] **기본 인덱스 구조(Basic Index Structure)**

[0027] 셀과 청크의 개념에 기초하여, 와플 그리드 인덱스의 기본 구조에 대해 설명한다. 공간, 즉 지리학적 공간(geographical space)은 ($[min_{lat}, max_{lat}), [min_{lon}, max_{lon})$)으로 정의된다. 여기서, $min_{lat}$와 $max_{lat}$는 각각 최소 및 최대 위도 값을 나타내고, $min_{lon}$과 $max_{lon}$은 각각 최소 및 최대 경도 값을 나타낸다. 공간 내의 물체 O의 좌표는 ($O_{lat}$, $O_{lon}$)으로 표현된다.

[0028] **Definition 2.1 (Cell).** (1) 공간 내에서 위도 방향의 셀의 개수는 $nCell_{lat}^{space}$ 이고, (2) 공간 내에서 경도 방향의 셀의 개수는 $nCell_{lon}^{space}$ 이라 가정하자. 물체 O의 좌표는 셀 좌표(cell coordinate) ($cell_{lat}$, $cell_{lon}$)에 맵핑된다. 여기서, $cell_{lat}$, $cell_{lon}$는 수학식 1과 같다.

[0029] [수학식 1]

$$cell_{lat} = \lfloor \frac{O_{lat} - min_{lat}}{max_{lat} - min_{lat}} \times nCell_{lat}^{space} \rfloor$$

$$cell_{lon} = \lfloor \frac{O_{lon} - min_{lon}}{max_{lon} - min_{lon}} \times nCell_{lon}^{space} \rfloor$$

[0030]

[0031] 와플(Waffle)의 셀 cell($cell_{lat}$, $cell_{lon}$)은 동일한 셀 좌표를 갖는 물체들의 집합이고, 셀 내의 물체들의 개수는 MOPC(Maximum Objects Per Cell)에 의해 제한된다.

[0032] 물체를 갖지 않는 셀은 엠티 셀(empty cell)이라 불린다. $nCell_{lat}^{space}$, $nCell_{lon}^{space}$, 및 MOPC는 셀들을 구성하는데 이용되는 와플(Waffle)의 놉들(knobs)이다. MOPC를 초과하는 개수의 물체가 동일한 셀 좌표를 갖는다면, 와플(Waffle)은 하나 이상의 셀을 생성하고, 이에 대한 구체적인 쿼리 처리는 후술한다.

[0033] **Definition 2.2 (Chunk).** (1) 단일의 청크(single chunk) 내에서 위도 방향의 셀의 개수를 $nCell_{lat}^{chunk}$ 이라 하고, (2) 단일의 청크 내에서 경도 방향의 셀의 개수를 $nCell_{lon}^{chunk}$ 라 하자. 셀의 셀 좌표 ($cell_{lat}$, $cell_{lon}$)는 정수의 청크 좌표(integer chunk coordinate) ($chunk_{lat}$, $chunk_{lon}$)로 매핑된다. 여기서, $chunk_{lat}$와 $chunk_{lon}$는 수학식 2와 같다.

[0034] [수학식 2]

$$chunk_{lat} = \lfloor \frac{cell_{lat}}{nCell_{lat}^{chunk}} \rfloor$$

$$chunk_{lon} = \lfloor \frac{cell_{lon}}{nCell_{lon}^{chunk}} \rfloor$$

[0035]

[0036] 청크 chunk($chunk_{lat}$, $chunk_{lon}$)는 (1) 청크 좌표가 ($chunk_{lat}$, $chunk_{lon}$)이고 (2) 집합 내에서 셀 좌표가 고유(unique)한 모든 가능한 셀들의 집합이다. 또한, 청크는 엠티 셀을 포함할 수 있다.

[0037] 와플(Waffle)에서, 청크는 메인 메모리 내에서 시리얼 메모리 할당(serial memory allocation)의 단위이다. 쿼리를 처리하는 동안에, 청크를 구성하는 이웃 셀들(neighboring cells)은 연속하여(consecutively) 액세스되는 경향이 있고, 이러한 액세스는 시리얼 메모리 할당을 통해 캐시 친화적(cache-friendly)이 될 수 있다. 와플은, 청크 내의 셀들은 셀 좌표의 위도-우선 순서(latitude-major order)로 저장되어 있음을 가정할 수 있다. 엠티 셀만으로 구성되는 청크는 엠티 청크(empty chunk)로 불린다. 물체를 저장하기 위해 동일한 셀 좌표를 갖는 하나 이상의 셀이 요구될 때, 청크 내에서 셀 좌표는 고유(unique)하기 때문에 와플(Waffle)은 하나 이상의 청크를 생성한다. 구체적인 쿼리 처리는 후술한다. $nCell_{lat}^{chunk}$와 $nCell_{lon}^{chunk}$는 청크를 정의하기 위해 이용되는 놉들(knobs)이다.

[0038] 다섯 가지의 놉 값들(knob values) {$nCell_{lat}^{space}$, $nCell_{lon}^{space}$, MOPC, $nCell_{lat}^{chunk}$, $nCell_{lon}^{chunk}$}을 놉 세팅(knob setting, 놉 설정)이라 명명한다. 놉 세팅의 결정은 와플(Waffle)의 전반적인 성능에 매우 큰 영향을 미친다. 놉 세팅을 결정하기 위한 구체적인 방법은 후술한다. 만약 $nCell_{lat}^{\{space/chunk\}} = nCell_{lon}^{\{space/chunk\}}$ 이면, 두 개의 놉을 간략히 $nCell^{\{space/chunk\}}$로 표기한다.

[0039] 도 1은 제안하는 그리드 인덱스(와플 그리드 인덱스)로써, (a)는 셀과 청크를 도시하고, (b)는 메모리 표현을 도시한다. 도 1에서, 공간은 ([0.3, 0.9), [0.3, 1.5))이고, 놉 세팅은 {2, 4, 2, 2, 2}이다. 각 물체는 셀에 맵핑되고, 각 셀은 청크에 맵핑된다. 도 1의 (b)는, 메인 메모리 내에서 각 청크의 셀들이 시리얼하게 위치하고 있으나, 두 개의 청크는 흩어져 있음(scattered)을 보여준다.

[0040] **쿼리 처리(Query Processing)**

[0041] 와플(Waffle)은 복수의(예컨대, 4 개의) 쿼리 유형, 삽입(insertion), 삭제(deletion), 레인지(range), 및 k-NN 쿼리을 지원할 수 있다. 이하에서는, 셀과 청크 구조에 기초하여 쿼리 처리를 설명한다.

[0042] **(1) 삽입 쿼리(Insertion query):** 물체의 키(key)와 좌표에 대하여, 삽입 쿼리는 물체를 와플 인덱스에 추가한다. 여기서, 키는 물체를 식별하기 위한 식별 정보(식별자, 아이디 등)를 의미할 수 있다. 도 2의 예제에 기초하여 쿼리 처리를 설명한다. 도 2의 물체들이 키의 오름차순(실시예에 따라, 내림차순이나 임의의 순서로 구현될 수도 있음)으로 도 1과 동일한 놉 세팅을 갖는 와플 인덱스로 삽입된다. Object 10의 셀 좌표와 청크 좌표는 모두 (0, 0)이고, 물체 10은 chunk(0, 0)$_1$ 내의 cell(0, 0)으로 삽입된다. Object 11의 셀 좌표와 청크 좌표 또한 모두 (0, 0)이다. 와플은, 우선, chunk(0, 0)$_1$ 내의 cell(0, 0)이 MOPC(=2) 보다 작은 물체들을 포함하는지 여부를 체크한다. 그러나, cell(0, 0)은 이미 MOPC 만큼의 물체들을 포함하고 있기 때문에, 와플은 동일한 청크 좌표 chunk(0, 0)$_2$를 갖는 새로운 청크를 생성하고, Object 11을 생성된 청크로 삽입한다. Object 12의 셀 좌표와 청크 좌표는 각각 (1, 0)과 (0, 0)이다. 와플은, chunk(0, 0)$_1$ 내의 cell(1, 0)이 이미 MOPC 만큼의 물체들을 포함하고 있는지 체크한다. 그런 다음, 와플은 Object 11에 대한 삽입 쿼리 동안에 생성된 다음 청크 chunk(0, 0)$_2$로 이동하고, Object 12를 청크에 삽입한다. Object 13은 동일한 방식으로 삽입된다.

[0043] **(2) 삭제 쿼리(Deletion query, 또는 제거 쿼리):** 물체의 키에 대하여, 삭제 쿼리는 와플 인덱스로부터 물체를 제거(또는 삭제)한다. 도 3의 예제에 기초하여 쿼리 처리를 설명한다. 우선, 와플은 Object 1를 삭제한다. Object 1의 셀 좌표와 청크 좌표는 모두 (0, 0)이다. 와플은 chunk(0, 0)$_1$을 체크하고, Object 1을 찾는다. 그런 다음, 와플은 셀 좌표의 마지막 물체 Object 11을 찾고, Object 1을 Object 11과 스왑(swaps)하고, Object 1이 삭제된 것으로 마크한다(즉, 삭제한다). 다음으로, 와플은 Object 12을 삭제한다. Object 12이 셀 좌표와 청크 좌표는 각각 (1, 0)과 (0, 0)이다. 와플은 우선 chunk(0, 0)$_1$ 내의 Object 5를 체크하고, 그 다음으로 chunk(0, 0)$_1$ 내의 Object 6을 체크하고, Object 12를 찾는데 실패한다. 와플은 다음 청크 chunk(0, 0)$_2$로 이동하고, Object 12를 Object 13과 스왑하고, Object 12이 삭제된 것으로 마크한다(즉, 삭제한다). Object 13 또한 삭제된다면, 와플은 chunk(0, 0)$_2$를 삭제한다. 엠티 청크이기 때문이다.

[0044] **(3) 레인지 쿼리(Range query, 또는 범위 쿼리):** 레인지 쿼리는 시작 좌표(start coordinate)와 끝 좌표(end coordinate)에 의해 식별되는 직사각형 범위(rectangular range) 내의 모든 물체들을 반환한다. 와플은 레인지 쿼리를 다음과 같이 처리한다. 청크가 직사각형의 범위 경계(range boundary) 내에 완전히 포함된다면, 청크 내의 모든 물체들은 응답에 추가된다. 청크가 범위 경계와 중첩(overlaps)된다면, 청크 내의 셀은 범위 경계에 완전히 포함되거나 중첩될 것이다. 전자의 경우, 셀 내의 모든 물체들이 응답에 추가된다. 후자의 경우, 와플은 셀 내의 각 물체가 범위 경계에 포함되는지 여부를 체크한다.

[0045] **(4) k-NN 쿼리(k-NN auery):** 타겟 좌표, k, 및 직사각형 범위(시작 좌표와 끝 좌표로 식별됨)에 대하여, k-NN 쿼리는 직사각형 범위 내에서 타겟 좌표와 가장 근접한 k개의 물체들을 반환한다. 이때, Wei Zhang 등이 제안한, 그리드 인덱스에서의 k-NN 쿼리 프로세싱(Wei Zhang, Jianzhong Li, and Haiwei Pan. 2006. Processing Continuous k-nearest Neighbor Queries in Location-dependent Application. International Journal of Computer Science and Network Security 6, 3 (2006), 1-9.)이 이용될 수 있다. 우선, 셀 레벨(cell level)을 정의한다. 타겟 좌표를 포함하는 셀의 좌표는 ($t_{lat}$, $t_{lon}$)이라고 가정한다. 셀 레벨 0은

{cell($t_{lat}$, $t_{lon}$)}으로 정의된다. 셀 레벨 i(≥1)는 $\{cell(cell_{lat}, cell_{lon}) | t_{lat} - i \le cell_{lat} \le t_{lat} + i, t_{lon} - i \le cell_{lon} \le t_{lon} + i\} - \bigcup_{j=0}^{i-1}$Cell level $j$ 로 정의된다. 예시적인 셀 레벨은 도 4의 (b)에 도시되어 있다.

[0046] 와플은 세가지 타입의 요소, 즉 물체, 논-엠티 셀(non-empty cell), 및 셀 레벨(cell level)을 포함하는 우선순위 큐(priority queue)를 유지한다. 요소를 위하여, 타겟 좌표로부터의 유클리드 거리(Euclidean distance)가 계산될 수 있다. 큐에서 타겟 좌표와 보다 근접한 요소는 더 높은 우선순위를 갖는다. 상세하게, 셀에 대하여, 셀 경계와 타겟 좌표 사이의 최단거리(shortest distance)가 계산된다. 셀 레벨에 대하여, 해당 레벨에 있는 가장 가까운 셀로부터의 거리가 계산된다.

[0047] 와플은 범위 경계와 중첩되거나 범위 경계에 포함되는 요소들만을 고려한다. 우선, 와플은 (1) Cell level 0의 모든 물체들과 (2) Cell level 1을 우선순위 큐에 삽입한다. 큐의 탑(top)이 물체이면, 이는 새로운 응답이 된다. 큐의 탑이 셀이면, 셀 내의 모든 물체들이 큐에 삽입된다. 큐의 탑이 셀 레벨이면, (1) 해당 셀 레벨의 모든 논-엠티 셀들과 (2) 다음 셀 레벨이 큐에 삽입된다. k 개의 물체들이 검색되거나 큐가 비면(empty), k-NN 쿼리에 대한 처리는 완료된다.

[0048] **놉 설정의 효과(Effects of Knob Configuration)**

[0049] 이하에서는, 구체적인 놉의 설정 효과를 예제를 이용하여 설명한다. 예제 내의 물체들은 키의 오름차순으로 삽입되고 내림차순으로 삭제되는 것으로 가정한다. 그리고, 레인지 쿼리는 모든 물체들이 삽입된 후에 수행되는 것으로 가정한다.

[0050] **(1) 공간 내 셀의 개수에 따른 효과(Effects of the Number of Cells in the Space)**

[0051] 도 5는 MOPC=1 그리고 $nCell^{chunk}$=2일 때의 예시를 도시한다. 도 5의 (a)에서, Object 5가 삽입될 때, 와플은 청크 내의 cell(1, 1)이 이미 MOPC의 물체를 포함하고 있는지 여부를 체크하기 위하여, chunk(0, 0)$_1$을 체크한다. 그런 다음, 와플은 물체를 다음 청크 chunk(0, 0)$_2$에 삽입한다. 도 5의 (b)에서, Object 5는, 추가적인 청크들에 대한 체크 없이, chunk(1, 1)에 삽입된다. 도 5의 (a)에서, Object 6이 삭제될 때, 와플은 chunk(0, 0)$_1$ 내의 Object 4와 chunk(0, 0)$_2$ 내의 Object 5를 체크한 다음, chunk(0, 0)$_3$ 내의 Object 6을 찾는다. 도 5의 (b)에서, Object 6은, 추가적인 물체들에 대한 체크 없이, chunk(1, 1)로부터 삭제된다. 도 5의 (a)에서, 레인지 쿼리를 위해서, 범위 경계가 모든 셀들과 중첩되기 때문에, 와플은 모든 물체들을 체크한다. 도 5의 (b)에서, 와플은 단일의 물체만을 체크한다. 도 5의 (a)는 도 5의 (b)보다 캐시 미스(cache miss)를 발생시킬 확률이 높다. 전자가 더 많은 청크들을 체크하기 때문이다. 메모리 사용의 관점에서, 도 5의 (b)는 도 5의 (a)에 비하여 메모리 사용량이 적다.

[0052] 도 6은 MOPC=1 그리고 $nCell^{chunk}$=2일 때의 다른 예시를 도시한다. Object 2가 삽입될 때, 도 6의 (a)에서, 와플은 우선 chunk(0, 0)$_1$을 체크하고 물체를 다음 청크 chunk(0, 0)$_2$에 삽입한다. 도 6의 (b)에서, Object 6은, 추가적인 청크들에 대한 체크 없이, chunk(3, 2)에 삽입된다. Object 4가 삭제될 때, 도 6의 (a)에서, 와플은 chunk(0, 0)$_1$ 내의 Object 3을 체크하고, chunk(0, 0)$_2$ 내의 Object 4를 찾는다. 도 6의 (b)에서, Object 4는, 추가적인 물체들에 대한 체크 없이, chunk(1, 5)로부터 삭제된다. 레인지 쿼리를 위해서, 도 6의 (a)에서, 와플은 모든 물체들을 체크한다. 범위 경계가 4개의 셀들 모두와 중첩되기 때문이다. 도 6의 (b)에서, 와플은 오직 3 개의 물체들만을 체크하지만, 이들은 메인 메모리 상에서 흩어져 있다. 따라서, 도 6의 (b)는 도 6의 (a)에 비해 캐시 미스를 일으킬 확률이 높다. 즉, 메모리 사용의 관점에서, 도 6이 (a)는 도 6의 (b)에 비하여 메모리 사용량이 적다.

[0053] **(2) 청크 내 셀의 개수에 따른 효과(Effects of the Number of Cells in a Chunk)**

[0054] 도 7은 $nCell^{space}$=4 그리고 MOPC=1일 때의 예시를 도시한다. 삽입 쿼리의 경우, 도 7의 (a)와 (b)에서, 모든 물체들은, 추가적인 청크들에 대한 체크 없이, 삽입된다. 삭제 쿼리의 경우, 도 7의 (a)와 (b)에서, 모든 물체들은, 추가적인 물체들에 대한 체크 없이, 삭제된다. 레인지 쿼리의 경우, 쿼리 처리 과정 중에 체크된 물체들의 개수는, 도 7의 (a)와 (b)에서, 동일하다. 대신, 삽입/삭제/레인지/k-NN 쿼리의 경우, 도 7의 (b)는, 시리얼 메모리 할당 때문에, 도 7의 (a)에 비해 더 많은 캐시 미스를 일으킬 수 있다. 메모리 사용의 관점에서, 도 7의

(a)는 도 7의 (b)에 비하여 더 적은 메모리를 사용한다.

[0055] **(3) 셀당 최대 물체 개수에 따른 효과(Effects of the Maximum Objects per Cell)**

[0056] 도 8은 $nCell^{space}$=$nCell^{chunk}$=2일 때의 예시이다. Object 3이 삽입될 때, 도 8의 (a)에서, 와플은 우선 chunk(0, 0)$_1$과 chunk(0, 0)$_2$를 체크한 다음, 물체를 chunk(0, 0)$_3$에 삽입한다. 도 8의 (b)에서, Object 3은, 추가적인 청크들에 대한 체크 없이, chunk(0, 0)로 삽입된다. Object 3이 삭제될 때, 도 8이 (a)에서, 와플은 우선 Object 1과 Object 2를 체크한 다음 chunk(0, 0)$_3$ 내의 Object 3을 찾는다. 도 8의 (b)에서, 와플은 또한 Object 1과 Object 2를 체크한 다음 chunk(0, 0) 내의 Object 3을 찾는다. 레인지 쿼리의 경우, 도 8의 (a)는, 각 청크 내의 이웃 셀들을 독출(reading)할 때, 캐시-친화적이다. 도 8의 (b)는, 동일한 셀 좌표를 갖는 물체들을 독출할 때, 캐시-친화적이다. 메모리 사용 관점에서, 도 8의 (a)는 도 8의 (b)에 비해 메모리 사용량이 적다.

[0057] 상술한 바와 같이, 삽입/삭제 쿼리의 경우, 체크된 청크들/물체들의 개수는 계산 비용(computational cost)과 캐시 미스에 영향을 미친다. 삽입/삭제 쿼리가 처리된 후에, 캐시 미스를 최소화하면서 와플이 다음 삽입/삭제 쿼리를 수행한다면, 효율적인 쿼리 처리가 가능하다. 레인지/k-NN 쿼리의 경우, 체크된 물체들이 메인 메모리에서 시리얼하게 위치한다면, 쿼리 처리는 더욱 효율적일 것이다. 놉 값이 작으면, 와플은 보다 적은 메모리를 사용하는 경향이 있다.

[0058] **새로운 인덱스 구축 기법(Regrid: A Mechanism for Redefining a Waffle Index)**

[0059] 예제들을 통하여 놉 구성의 여러 효과들에 관해 살펴보았다. 이하에서는, 놉 세팅을 자동적으로 결정하는 방법을 설명하기에 앞서, 새로운 인덱스(Waffle index)를 재정의(redefining)하는 방법(또는 그리드 인덱스 교체 방법)을 제안한다.

[0060] x(x는 임의의 자연수) 개의 쿼리들이 처리될 때마다, 새로운 놉 세팅에 대하여, 와플은 그리드 인덱스를 재조직(reorganizes)한다. 이는 리그리드(regrid)라 불린다. 구체적으로, 오리지널 그리드 인덱스 내의 각 물체에 대하여, 와플은 오리지널 인덱스로부터 물체를 삭제하고 새로운 그리드 인덱스로 삽입한다. 오리지널 인덱스 내에 물체가 남아있지 않을 때 새로운 인덱스는 오리지널 인덱스를 대체하고, 이는 리그리드(regrid)가 완료되었음을 의미한다.

[0061] 와플 인덱스를 재구축하는 동안, 사용자 쿼리들을 처리하는 메커니즘이 없다면, 쿼리 처리는 블락되고, 블락된 시간은 물체들의 개수에 따라 증가한다. 이러한 문제점을 해결하기 위해서, 와플은 2 페이즈 락킹 프로토콜(two-phase locking protocol) 기반의 동시성 제어 기법(concurrency control scheme, 또는 동시성 제어 방법)을 포함한다. 와플은 잠금 단위(lock granularity)를 동일한 청크 좌표를 갖는 청크들의 집합으로 설정한다. 청크는 시리얼 메모리 할당의 단위이고 동일한 청크 좌표를 갖는 청크들은 쿼리 처리 동안에 함께 액세스되는 경향이 있기 때문이다. 삽입/삭제 쿼리는 배타적 잠금(exclusive lock, X-lock)을 요하고, 레인지/k-NN 쿼리는 공유 잠금(shared locks, S-lock)의 집합을 요한다. 트랜잭션(transaction)이 잠금(lock)의 집합을 요할 때, 데드락 문제(deadlock problem)는 다음과 같이 잠금 획득 순서(partial order)를 설정함으로써 예방될 수 있다. (1) 오리지널 인덱스 내의 잠금들(locks)은 새로운 인덱스 내의 잠금들 보다 먼저 획득된다(obtained). (2) 인덱스에서, $chunk_1$=chunk($lat_1$, $lon_1$) 및 $chunk_2$=chunk($lat_2$, $lon_2$)에 대하여, $lat_1 < lat_2$이면, $chunk_1$에 대한 잠금은 $chunk_2$에 대한 잠금보다 먼저 획득된다. $lat_1$=$lat_2$ 및 $lon_1 < lon_2$이면, $chunk_1$에 대한 잠금은 $chunk_2$에 대한 잠금보다 먼저 획득된다.

[0062] 잠금 프로토콜(locking protocol)에 기초하여, 와플은 리그리드(regrid)를 전이 트랜잭션들(transfer transactions)의 집합으로 분할하고(splits), 이들 각각은 어떤 물체에 대해 오리지널 인덱스로부터 하나의 삭제 쿼리와 새로운 인덱스로의 하나의 삽입 쿼리로 구성된다. 와플은 동시에 발생하는 사용자 쿼리들을, 리그리드(regrid)로부터의 모든 전이 트랜잭션들이 완료될 때까지 기다리지 않고, 처리한다. 도 9는 리그리드(regrid)를 위한 예시적인 스케줄을 도시한다. 사용자로부터의 레인지 쿼리는, 분할하지 않는다면, 리그리드(regrid)가 완료될 때까지 처리될 수 없지만(도 9의 (a)), 와플은 하나의 전이 트랜잭션 이후에 레인지 쿼리를 처리할 수 있다(도 9의 (b)).

[0063] 동시성 제어 기법(concurrency control scheme)에 기반하여 리그리드를 분할하는 것은, 리그리드(regreid) 동안에, 쿼리 처리의 수정을 요한다. 본 발명에서는, (1) 사용자는 고유한 물체에 대응한다, (2) 사용자가 쿼리를 발행(issue)하면 쿼리 처리가 완료되기 전까지 다른 쿼리를 발행할 수 없다, 및 (3) 다중 사용자로부터의 사용

자 쿼리들은 단일의 쓰레드(single thread) 기반으로 처리되고 리그리드(regrid)는 다른 쓰레드에서 처리된다는 것을 가정한다. 구체적인 쿼리 처리는 다음과 같다.

[0064] **(1) 사용자 삽입 쿼리(User insertion query):** 주어진 물체가 오리지널 인덱스와 새로운 인덱스 모두에 존재하지 않는다면, 와플은, 오리지널 인덱스를 고려하지 않고, 새로운 인덱스로의 삽입 쿼리를 위한 트랜잭션을 수행한다. 주어진 물체가 오리지널 인덱스 또는 새로운 인덱스에 이미 존재한다면, 와플은, 후술하는 바와 같이, 요청을 이동 쿼리(movement query)로 처리한다.

[0065] **(2) 사용자 삭제 쿼리(User deletion query):** 물체가 오리지널 인덱스에 여전히 존재한다면, 와플은 오리지널 인덱스에 대한 삭제 쿼리를 위한 트랜잭션을 수행한다. 물체가 새로운 인덱스에 존재한다면, 와플은 새로운 인덱스에 대한 삭제 쿼리를 위한 트랜잭션을 처리한다.

[0066] **(3) 사용자 이동 쿼리(User movement query):** 동시성에 대한 고려가 없다면 이동 쿼리는 삭제 쿼리와 연이은 삽입 쿼리로 단순히 대체할 수 있다. 하지만 리그리드(regrid)가 진행되는 도중에는 이동 쿼리를 개별적으로 고려하여야 한다. 즉, 와플은 (1) 삭제 쿼리와 (2) 연이은 삽입 쿼리를 위한 트랜잭션을 처리한다. 물체에 대한 삭제 쿼리와 삽입 쿼리가 트랜잭션으로써 처리되지 않는다면, 삭제 쿼리가 처리된 이후에, 동시간에 진행되는 다른 트랜잭션은 물체가 삭제되었는지 또는 이동 쿼리가 처리되는 도중인지를 결정할 수 없다. 타겟 물체가 여전히 오리지널 인덱스에 존재한다면, 와플은 (1) 오리지널 인덱스로부터의 삭제 쿼리와 (2) 연이은 새로운 인덱스로의 삽입 쿼리를 위한 트랜잭션을 처리한다. 타겟 물체가 새로운 인덱스에 존재한다면, 와플은 (1) 새로운 인덱스로부터의 삭제 쿼리와 (2) 연이은 새로운 인덱스로의 삽입 쿼리를 위한 트랜잭션을 처리한다.

[0067] **(4) 전이 트랜잭션(Transfer transaction):** 물체에 대한 전이 트랜잭션을 처리하기 전에, 동일한 물체를 타겟으로 하는 사용자 삽입/삭제/이동 쿼리가 처리되는 중이라면, 와플은 해당 쿼리 처리가 완료될 때까지 대기한다. 사용자 쿼리의 처리가 완료된 이후에, 와플은 전이 트랜잭션을 시작하고 동일한 물체를 타겟으로 하는 사용자 삽입/삭제/이동 쿼리가 대기하도록 한다.

[0068] 여전히 오리지널 인덱스에 존재하는 각 물체에 대하여, 와플은 (1) 오리지널 인덱스로부터의 삭제 쿼리와 (2) 연이은 새로운 인덱스로의 삽입 쿼리로 구성되는 전이 트랜잭션을 처리한다. 와플은 오리지널 인덱스로부터 이미 삭제된 물체에 대한 전이를 수행할 필요가 없다. 이전의 사용자 삭제 또는 이동 쿼리가 공간으로부터 물체를 삭제하였거나 새로운 인덱스에서 물체의 좌표를 업데이트했기 때문이다.

[0069] **(5) 사용자 레인지/k-NN 쿼리(User range/k-NN query):** 와플은 두 개의 레인지/k-NN 쿼리들을 위한 트랜잭션(오리지널 인덱스를 위한 레인지/k-NN 쿼리와 새로운 인덱스를 위한 연이은 레인지/k-NN 쿼리)을 처리한다. 각 쿼리를 처리하기 전에, 트랜잭션은 타겟 인덱스에 대한 모든 요구되는 공유 잠금들(shared locks)을 획득한다. 올바른 응답들(correct answers)은 두 개의 레인지/k-NN 응답들의 결합으로부터 구축된다. 그 이유는 다음과 같다. (1) 다른 트랜잭션 A가, 레인지/k-NN 트랜잭션에 의해 요구되는, 청크를 업데이트 하기 위한 배타적 잠금(exclusive lock)을 유지(holds)할 때, 레인지/k-NN 트랜잭션은 A가 잠금을 해제(releases)할 때까지 대기하여야 하고, 추후에 레인지/k-NN 트랜잭션은 공유 잠금(shared lock)을 획득한다. (2) 레인지/k-NN 트랜잭션이 청크에 대한 공유 잠금(shared lock)을 유지(holds)할 때, 동일한 청크를 업데이트하는 다른 트랜잭션 B는 레인지/k-NN 트랜잭션이 잠금을 해제할 때까지 대기하여야 하고, 추후에 B는 배타적 잠금(exclusive lock)을 획득한다. (3) 이동/전이 처리는, 공간 내의 물체는 오직 오리지널 인덱스 또는 새로운 인덱스에만 존재하여야 함을 보장한다.

[0070] **와플메이커(WaffleMaker): 강화 학습 기반의 설정 튜닝 시스템 (Reinforcement Learning-Based Configuration Tuning System)**

[0071] 셀과 청크를 정의하는 5개의 놉들, 다양한 설정 효과, 및 새로운 놉 세팅이 주어졌을 때 리그리드 처리에 대하여 설명하였다. 이하에서는, 리그리드(regrid)를 위한 놉 설정을 결정하기 위한, 온라인 설정 튜닝 시스템(와플메이커(WaffleMaker)라 명명될 수 있음)을 설명한다. 와플메이커는 새로운 형태의 탐색(exploration)과 활용(exploitation)을 수행하는 강화 학습에 기반하고, 사전 학습(pre-training)을 요하지 않는다. 와플메이커의 기본적인 구성 요소, 모델 정의, 및 와플메이커의 워크플로우에 대해 설명한다.

[0072] **기본적인 구성 요소(Basic Components)**

[0073] 우선, 와플메이커(WaffleMaker)의 기본적인 구성 요소들, 즉 상태(state), 액션(action), 보상(reward), 및 경험(experience)을 정의한다.

[0074] **(1) 상태(State (S))**: 와플메이커 상태(state)는 공간 내의 현재 물체들을 요약하는 그리드의 형태로 정의된다. 특히, 주어진 (1) 지리학적 공간(geographical space), (2) 위도방향으로 상태 셀들(state cells)의 개수 $nCell_{lat}^{state}$, 및 (3) 경도방향으로 상태 셀들의 개수 $nCell_{lon}^{state}$에 대하여, 와플메이커 상태는 $nCell_{lat}^{state} \times nCell_{lon}^{state}$-크기의 그리드이고, 이의 각 셀은 범위에 해당하는 물체들의 개수를 유지한다. 이때, 상태 그리드는 와플 그리드 인덱스와 연관되지 않는다(즉, 와플 그리드와는 별도로 정의되는 그리드이다). $nCell_{lat}^{state}$과 $nCell_{lon}^{state}$은 와플메이커의 하이퍼파라미터들이고, 이는 와플의 런타임 동안 변경되지 않음을 의미한다.

[0075] **(2) 액션(Action)**: 와플메이커 액션은 놉 세팅 KS를 결정하는 것이다. 특히, 각 놉은 [0, 1] 내의 값을 나타내는 단일의 실수 변수(single real-number variable)로 표현된다. 즉, 액션 공간은 연속적(continuous)임을 의미한다. 각 놉의 최소, 최대 값이 주어지면, 변수의 값은 최종 놉 값에 맵핑된다. 연속적인 액션 공간은 와플메이커가 다양한 범위의 놉을 하나의 변수로 표현하는 것을 가능케 한다.

[0076] **(3) 보상(Reward(R))**: 와플메이커 보상은 액션의 퀄리티(quality)를 나타낸다. 보상은 5 가지의 요소, 즉 삽입, 삭제, 레인지, k-NN, 및 메모리 보상으로 구성된다.

[0077] **{Insertion/Deletion/Range/k-NN} 보상($R_{\{i/d/r/k\}}$)**: $x_{\{i/d/r/k\}}(\neq 0)$ {insertion/deletion/range/k-NN} 쿼리들이 KS를 이용한 리그리드가 완료된 이후에 처리된다고 가정하자. 액션에 대한 {insertion/deletion/range/k-NN} 리워드는 $x_{\{i/d/r/k\}}$ 쿼리들에 대한 평균 {insertion/deletion/range/k-NN) 시간의 음의 값(negative value)으로 정의된다. 이는 수학식 3과 같다.

[0078] [수학식 3]

[0079] $$R_{\{i/d/r/k\}} = -\frac{total\ execution\ time\ for\ x_{\{i/d/r/k\}}\ queries}{x_{\{i/d/r/k\}}}$$

[0080] 더 짧은 실행 시간에 대하여 더 높은 보상을 제공하기 위하여, 음의 부호를 사용한다.

[0081] **메모리 리워드(Memory reward ($R_m$))**: 청크들의 전체 개수를 |chunks|라 하고, 단일의 청크의 메모리 사용량(memory usage)을 mChunk라 하자. 메모리 보상은 (대략적인) 메모리 사용량의 음의 값이다. 이는 수학식 4와 같다.

[0082] [수학식 4]

[0083] $$R_m = -|chunks| \times mChunk$$

[0084] |chunks|×mChunk는 특정 순간(specific moment)에서 와플 인덱스의 전체 메모리 사용량과 근사한 값을 갖는다. 단일 청크가 메모리 할당 단위이고 단일 청크의 메모리 사용량은, 놉 세팅이 변경되지 않는다면, 청크 내의 물체들의 존재와 무관하게 일정하기 때문이다. |chunks|는 삽입 쿼리나 삭제 쿼리에 따라 변화할 수 있다. 따라서, 와플메이커는 메모리 보상이 측정되는 시점의 값들을 획득할 수 있다.

[0085] 최종 보상을 계산하기에 전에, 각 요소는, 특정 요소가 최종 보상에 과도하게 반영되는 것을 예방하기 위하여, 정규화(normalized)된다. (1) 삽입, 삭제, 레인지, k-NN, 및 메모리의 정규화된 보상들은 각각 $R_i', R_d', R_r', R_k'$, 및 $R_m'$이고, (2) 쿼리 처리 시간(query processing time)과 메모리 사용량에 대한 가중치(weights)를 각각 $w_{time}$와 $w_{memory}$라 하자. 이때, $w_{time} + w_{memory} = 1$이다. 최종 보상 R은 수학식 5와 같이 정의된다.

[0086] [수학식 5]

[0087] $$R = w_{time}\{\frac{R_i' + R_d' + R_r' + R_k'}{4}\} + w_{memory}R_m'$$

[0088] 가중치 값들은 와플메이커의 하이퍼파라미터들이다.

[0089] **(4) 경험(Experience)**: 현재 상태(current state)는 S이고, 와플메이커는 새로운 놉 세팅 KS를 결정하고 액션에 대한 보상 R을 획득한다고 가정하자. 경험은 (S, KS, R)로 정의된다.

[0090] **WaffleMaker Model**

[0091] 상태 S와 놉 세팅 KS에 대하여, 와플메이커(WaffleMaker) 모델은 예상되는 보상을 출력하고, 이는 모델이 액션-값 함수(action-value function) Q(S, KS)=R임을 의미한다. 모델은 합성곱 레이어들(convolutional layers), 풀링 레이어들(pooling layers), 및 완전 연결 레이어들(fully connected layers)로 구성된 합성곱 신경망(convolutional neural network)이다. 모델은 추가적인 정보를 요구하지 않고 상태로부터 학습을 진행한다. 이는 기존의 연구들(Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. CoRR abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602, David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 529, 7587 (2016), 484-489. https://doi.org/10.1038/nature16961)과 유사한 면이 있다. 예컨대, Volodymyr Mnih 등의 연구는, 픽셀 값들로부터 Atari 게임을 플레이하는 방법을 학습하는 합성곱 신경망을 갖는 심층 강화 학습 모델을 제안하였다.

[0092] 모델은 오프-폴리시 방식(off-policy manner)으로 업데이트된다. 주어진 경험들의 집합 (S, KS, R)에 대하여, 와플메이커는 경험 집합에서 |batch|개 경험들을 샘플링하여 미니-배치(mini-batch)를 구성하고, 수학식 6의 MSR(Mean Squared Error)에 따라 모델을 업데이트한다.

[0093] [수학식 6]

[0094]
$$loss = \frac{1}{|batch|} \sum_{i=1}^{|batch|} (Q(S_i, KS_i)) - R_i)^2$$

[0095] 상태 S에 대하여, 와플메이커가 놉 세팅 KS를 결정하는 방식은 다음과 같다. 탐색(exploration)방식을 먼저 소개한다. 와플메이커는 임의로 |candidates|개 놉 세팅들을 선택한다. 그런 다음, 와플메이커는 놉 세팅들을 단일의 미니-배치로 구성하여 모델에 제공하고, 예상되는 보상들을 획득하며, 예상되는 보상들의 내림차순(descending order)으로 놉 세팅들을 정렬한다. 예상되는 보상들과 관련하여 상위 T%개의 놉 세팅들에 대하여, 와플메이커는 예상되는 보상을 샘플링 가중치(sampling weights)로 활용하여 최종 놉 세팅을 샘플링한다. 모델의 정확도가 높아질수록, 탐색으로 결정한 놉 세팅의 실제 최종 보상은 더 높아질 것으로 예상할 수 있다.

[0096] 다음으로 활용(exploitation)방식을 소개한다. 와플메이커는 최근 탐색들(recent explorations)에 의해 결정된 |recent|개 놉 세팅들을 이용한다. 구체적으로, 와플메이커는 최근 놉 세팅들을 미니-배치로 구성하여 모델에 제공하고, 예상되는 보상이 가장 높은 상위 하나의 놉 세팅을 결정한다. 최근 놉 세팅들로부터 놉을 결정하는 것은 연속적인 액션 공간(continuous action space)이 이산적인 액션 공간(discrete action space)으로 대체되는 것을 의미하며, 여기서 이산적인 액션들은 고정되지 않고 변화한다. 모델의 정확도가 높아질수록, 최근 놉 세팅들의 현재 집합은 이전 집합에 비해 더 높은 최종 보상으로 이어질 것으로 예상할 수 있다. 신규한 활용 방법(exploitation)은 와플이 리그리드(regrid)를 수행하는 시점을 결정할 수 있도록 한다.

[0097] |batch|, |candidates|, T, 및 |recent|는 와플메이커의 하이퍼파라이터들이다. 모델의 정확도가 향상됨에 따라, 점진적으로 T를 감소시키는 것도 가능하다.

[0098] **와플메이커의 워크플로우(Workflow of WaffleMaker)**

[0099] 이하에서는, 도 10과 도 11에 기초하여 와플메이커의 워크플로우에 대해 설명한다. 와플은, 이전 리그리드(regrid)가 완료된 이후에, x(x는 임의의 자연수로써 미리 정해진 값을 가질 수 있음) 개의 쿼리들이 처리될 때마다 새로운 리그리드를 시작한다. 새로운 리그리드(regrid)를 시도하는 기준은 상황에 따라 다르게 정의될 수 있다. 예를 들어, x 대신 일정 시간 마다 리그리드를 시도할 수 있다. 워크플로우는 모델의 수렴(convergence) 여부에 따라 가변적이다.

[0100] 우선, 도 10을 참조하여, 모델이 수렴하기 전의 워크플로우를 설명한다. 와플은 x 개의 쿼리들을 처리하고 새로운 리그리드를 시작한다(Steps 1-2). 와플메이커는 현재 상태 S로부터 새로운 놉 세팅 KS를 결정한다(Step 3). 와플메이커는 확률 $P_{exploration}$의 탐색과 확률 1-$P_{exploration}$의 활용을 통해 놉 세팅을 획득한다. $P_{exploration}$는 와플메

이커의 하이퍼파라미터이다.

[0101] 와플메이커가 새로운 놉 세팅을 결정한 이후에, 와플은 와플메이커로부터 놉 세팅을 수신한다(Step 4). 그러면, 와플은, 동시간 사용자 쿼리들을 처리하면서, 리그리드를 수행한다(Step 5).

[0102] 와플이 리그리드를 수행하는 동안, 와플메이커는 수학식 5에 기초하여 x 개의 쿼리들에 대한 보상 R을 계산하고(Step 6), R과 관련된 새로운 경험을 경험 버퍼(experience buffer)에 저장하고(Step 7), 버퍼를 활용하여 모델을 업데이트한다(Step 9).

[0103] 와플은 리그리드를 완료하고(Step 8) 다음 x 개의 쿼리들을 처리한다(Step 1). x 개의 쿼리들은 이전 리그리드 동안의 사용자 쿼리들을 포함하지 않는다. 리그리드 동안의 쿼리 처리는 두 개의 상이한 놉 세팅들에 의해 영향을 받기 때문이다. 와플은 새로운 리그리드를 시작하고(Step 2), 위 과정을 반복한다.

[0104] 모델이 수렴한 이후의 워크플로우는 도 11에 도시되어 있다. 와플은 x 개의 쿼리들을 처리하고 새로운 리그리드를 시작한다(Step 1-2). 이때, 와플메이커는 리그리드를 수행할지 여부를 결정한다(Step 3). 특히, 와플메이커는 활용(exploitation)을 통해서 현재 상태로부터 후보 놉 세팅을 획득한다. 후보 놉 세팅이 현재 놉 세팅과 동일하면, 현재 상태에 대하여 |recent| 놉 세팅들 중에서 현재 놉 세팅이 여전히 가장 높은 기대 보상을 갖고, 와플메이커는 리그리드를 수행하지 않기로 결정한다(Case 1). 그렇지 않으면, 현재 놉 세팅은 더 이상 현재 상태에 대하여 |recent| 놉 세팅들 중에서 가장 높은 기대 보상을 갖지 않고, 와플메이커는 리그리드를 수행할 것을 결정한다(Case 2).

[0105] 와플메이커에 대한 주요 동기는 (1) 놉 세팅을 어떻게 결정할 것인가와 (2) 리그리드를 언제 수행할 것인가이다. 첫번째 동기와 관련하여, 놉 세팅을 결정하는 과정은 강화 학습 문제로써 모델링된다. 여기서, 상태는 물체 분포(object distribution)에 의해 정의되고, 액션은 놉 세팅을 결정하는 것이고, 보상은 인덱스 성능(index performance)으로부터 계산된다. 두번째 동기와 관련하여, 추가적인 모델이나 규칙(rules)을 생성하는 것 대신에, 와플메이커는, 연속적인 액션 공간을 이산적인 액션 공간으로 대체함으로써, 자연스럽게 리그리드를 수행할 시점을 결정한다. 수렴 이후에, 와플메이커는 주기적으로(또는 비주기적으로) 현재 상태로부터 후보 놉 세팅을 결정하고, 후보가 기존의 놉 세팅과 상이한지 체크한다. 두 개의 놉 세팅들이 동일하다면, 와플메이커는, 리그리드를 수행할만큼 물체 분포가 충분히 변하지 않았다고 인식한다. 그렇지 않으면, 와플메이커는, 리그리드를 수행할만큼 물체 분포가 충분히 변했다고 인식한다. 요컨대, 와플메이커는 두 개의 동기들을 모두 고려한 해결책이다.

[0106] 이상에서 설명한 방법들 그리고 이하에서 설명될 방법들, 즉 그리드 인덱스 구축 방법, 놉 세팅 값 결정 방법 등은 적어도 프로세서(processor) 및/또는 메모리(memory)를 포함하는 포함하는 컴퓨팅 장치에 의해 수행될 수 있다. 즉, 방법들에 포함되는 각 단계를 중 적어도 일부는 컴퓨팅 장치에 포함되는 프로세서의 동작으로 이해될 수도 있다. 컴퓨팅 장치는 서버(server), PC(Personal Computer) 등을 포함한다. 또한, 컴퓨팅 장치는 유무선 통신망을 통하여 사용자 쿼리를 수신하기 위한 통신 인터페이스(통신부라 명명될 수도 있음)를 포함할 수 있다.

[0107] 이하에서는, 앞서 상술한 내용에 관하여는 그 구체적인 설명을 생략하기로 한다.

[0108] 도 12는 본 발명의 일 실시예에 따른 그리드 인덱스 대체 방법을 설명하기 위한 흐름도이다. 도 12를 통해 설명되는 그리드 인덱스 대체 방법은, 모델(즉, 제2 모델로 명명될 수도 있는 와플메이커 모델)의 수렴 전에 수행되는 방법을 의미할 수 있다.

[0109] 우선, 사용자 쿼리가 처리된다(S110). 처리되는 사용자 쿼리는 미리 정해진 개수의 쿼리들(x 개의 쿼리들)을 의미할 수 있고, 쿼리 처리는 제1 모델(예컨대, 와플)을 이용하여 수행될 수 있다. 또한, 사용자의 쿼리는 삽입 쿼리, 삭제 쿼리, 레인지 쿼리, k-NN 쿼리, 및 이동 쿼리 중 어느 하나를 의미할 수 있다. 또한, 사용자 쿼리는 소정의 유무선 통신을 통하여 사용자의 단말로부터 수신될 수 있고, 처리 결과는 다시 사용자의 단말로 송신될 수 있다.

[0110] 사용자의 쿼리에 대한 처리가 완료된 후, 새로운 그리드 인덱스를 위한 놉 세팅이 결정될 수 있다(S120). 새로운 놉 세팅은 새로운 그리드 인덱스를 정의하기 위한 설정 값을 의미하는 것으로, 제2 모델(예컨대, 와플메이커)을 이용하여 도출될 수 있다.

[0111] 새로운 놉 세팅이 결정되면, 새로운 그리드 인덱스가 구축될 수 있다(S130). 즉, 각 물체에 대하여 기존의 그리드 인덱스로부터 새로운 그리드 인덱스로의 전이 트랜잭션이 수행될 수 있다. 새로운 그리드 인덱스에 대한 구축은 제1 모델(예컨대, 와플)을 이용하여 수행될 수 있다. 이 과정에서, 사용자 쿼리는 블락되지 않고 처리될

수 있고, 이에 대한 상세한 설명은 생략하기로 한다.

[0112] 또한, 새로운 놉 세팅에 대하여 쿼리들에 대한 보상이 획득될 수 있다(S140). 보상은 수학식 5를 통하여 산출될 수 있다. 획득된 보상에 대한 경험은 경험 버퍼에 저장되고 모델(제2 모델로써 와플메이커를 의미할 수 있음)은 업데이트된다(S150). S140 단계와 S150 단계는 제2 모델(와플메이커)을 이용하여 수행될 수 있다.

[0113] 이때, S130 과정과, S140 과정 및 S150 과정은 병렬적으로 수행될 수 있다. 또한, 상술한 과정은 반복적으로 수행될 수 있고, 이에 따라 모델(제2 모델)에 대한 학습이 진행되어 수렴된 모델을 획득할 수 있다.

[0114] 도 13은 본 발명의 다른 실시예에 따른 그리드 인덱스 대체 방법을 설명하기 위한 흐름도이다. 도 13을 통해 설명되는 그리드 인덱스 대체 방법은, 모델(즉, 제2 모델로서, 와플메이커를 의미할 수 있음)의 수렴 후에 수행되는 방법을 의미할 수 있다.

[0115] 우선, 사용자 쿼리가 처리된다(S210). 처리되는 사용자 쿼리는 미리 정해진 개수의 쿼리들(x 개의 쿼리들)을 의미할 수 있고, 쿼리 처리는 제1 모델(와플)을 이용하여 수행될 수 있다. 또한, 사용자의 쿼리는 삽입 쿼리, 삭제 쿼리, 레인지 쿼리, k-NN 쿼리, 및 이동 쿼리 중 어느 하나를 의미할 수 있다. 또한, 사용자 쿼리는 소정의 유무선 통신을 통하여 사용자의 단말로부터 수신될 수 있고, 처리 결과는 다시 사용자의 단말로 송신될 수 있다.

[0116] 사용자의 쿼리에 대한 처리가 완료된 후, 새로운 그리드 인덱스를 구축할지 여부를 결정하는 단계가 수행된다(S220). 구체적으로, 제2 모델(와플메이커)에 의하여 도출된 놉 세팅이 이전의 놉 세팅(현재의 놉 세팅)과 같은지 여부에 대한 비교가 수행된다. 비교 결과, 동일하다면 S210 단계가 재차 수행되고, 동일하지 않다면 S230 단계가 수행된다.

[0117] S230 단계에서는 결정된 새로운 놉 세팅에 대하여, 새로운 그리드 인덱스가 구축될 수 있다(S230). 즉, 각 물체에 대하여 기존의 그리드 인덱스로부터 새로운 그리드 인덱스로의 전이 트랜잭션이 수행될 수 있다. 새로운 그리드 인덱스에 대한 구축은 제1 모델(예컨대, 와플)을 이용하여 수행될 수 있다. 이 과정에서, 사용자 쿼리는 블락되지 않고 처리될 수 있고, 이에 대한 상세한 설명은 생략하기로 한다.

[0118] S230 단계가 완료되면, S210 단계가 재차 수행됨으로써, 상술한 과정이 반복적으로 수행될 수 있다.

[0119] 이상에서 설명된 장치는 하드웨어 구성 요소, 소프트웨어 구성 요소, 및/또는 하드웨어 구성 요소 및 소프트웨어 구성 요소의 집합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성 요소는, 예를 들어, 프로세서, 콘트롤러, ALU(Arithmetic Logic Unit), 디지털 신호 프로세서(Digital Signal Processor), 마이크로컴퓨터, FPA(Field Programmable array), PLU(Programmable Logic Unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(Operation System, OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술 분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(Processing Element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(Parallel Processor)와 같은, 다른 처리 구성(Processing Configuration)도 가능하다.

[0120] 소프트웨어는 컴퓨터 프로그램(Computer Program), 코드(Code), 명령(Instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(Collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성 요소(Component), 물리적 장치, 가상 장치(Virtual Equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(Signal Wave)에 영구적으로, 또는 일시적으로 구체화(Embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

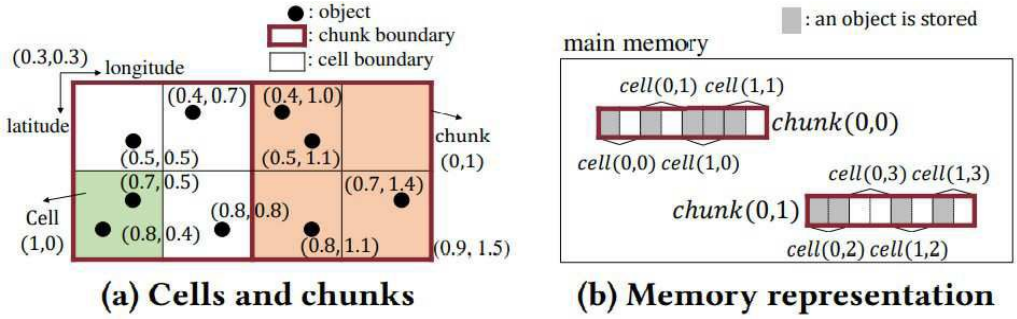
[0121] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가

능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic Media), CD-ROM, DVD와 같은 광기록 매체(Optical Media), 플롭티컬 디스크(Floptical Disk)와 같은 자기-광 매체(Magneto-optical Media), 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

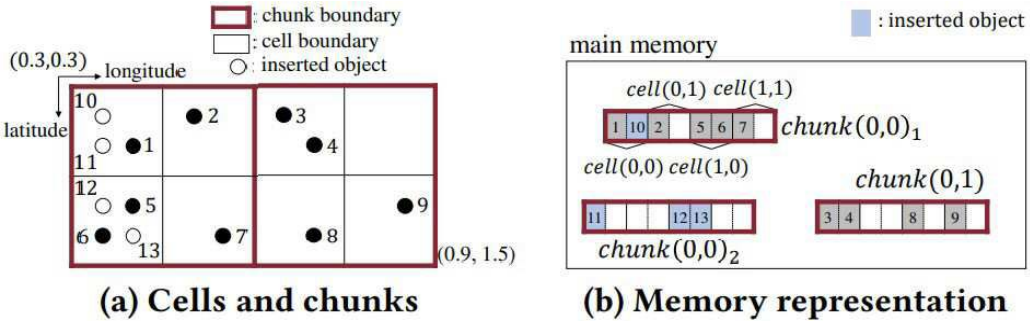
[0122] 본 발명은 도면에 도시된 실시예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시예가 가능하다는 점을 이해할 것이다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성 요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다. 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 등록청구범위의 기술적 사상에 의해 정해져야 할 것이다.

## 도면

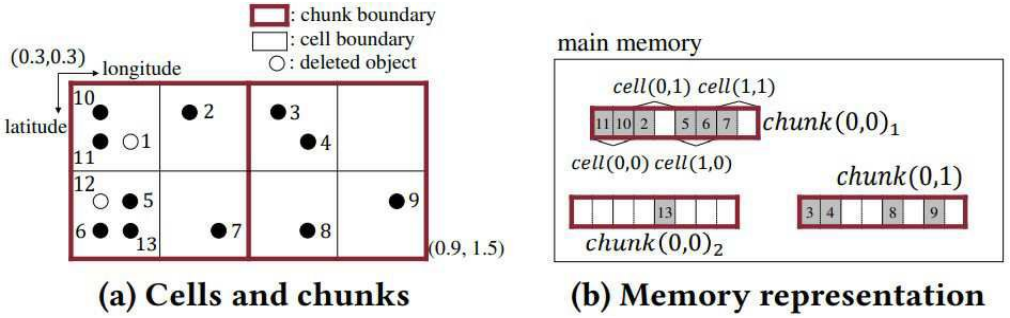
### 도면1



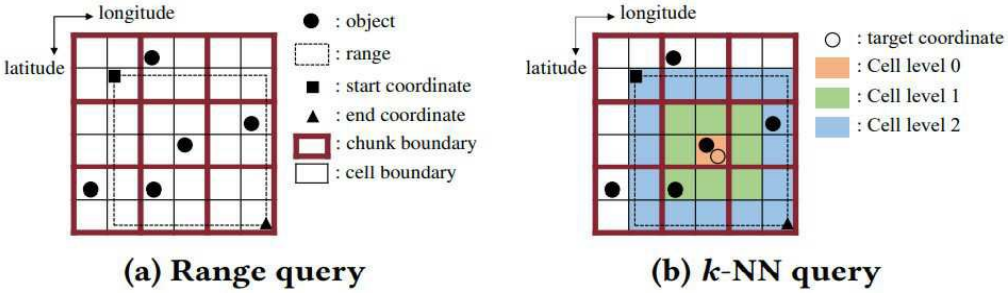

### 도면2




### 도면3

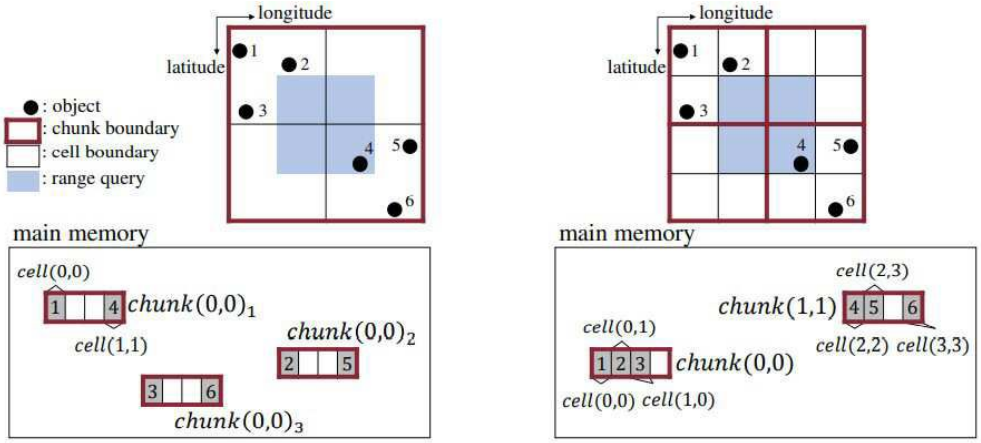


도면4



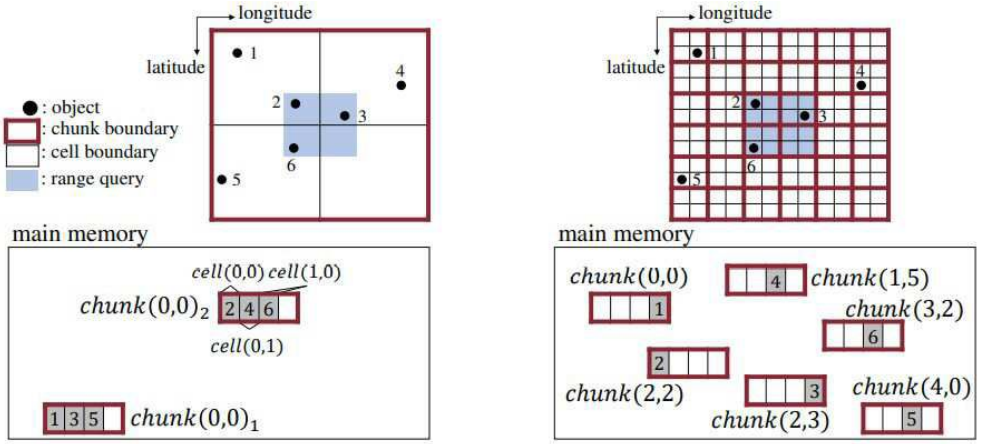

(a) Range query (b) *k*-NN query

도면5



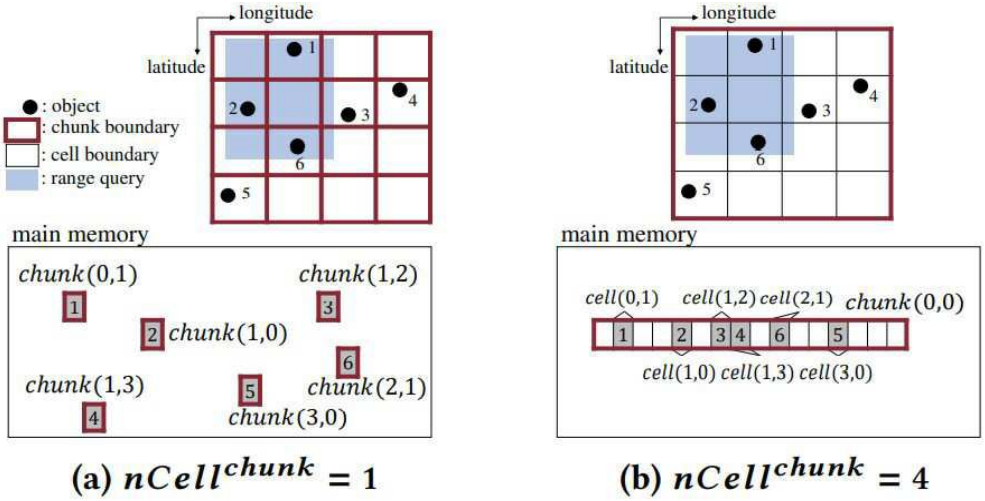

(a) $nCell^{space} = 2$ (b) $nCell^{space} = 4$

도면6



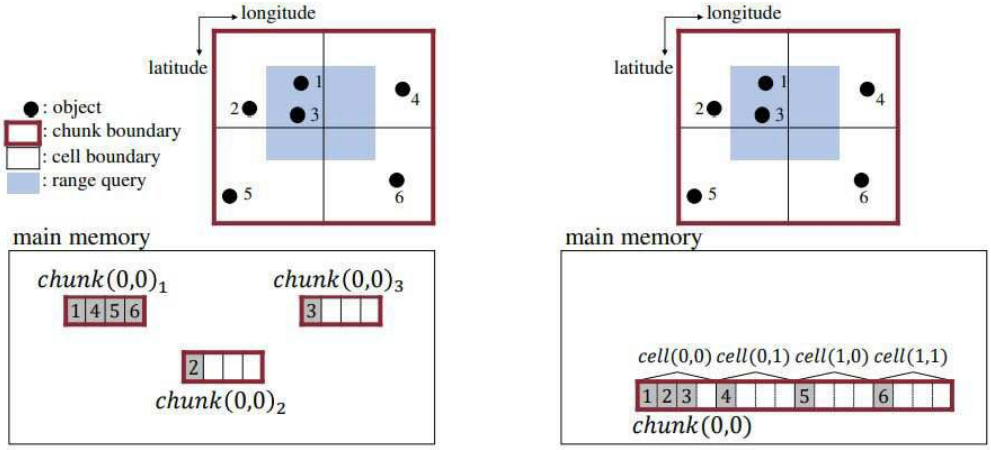

(a) $nCell^{space} = 2$ (b) $nCell^{space} = 12$

도면7



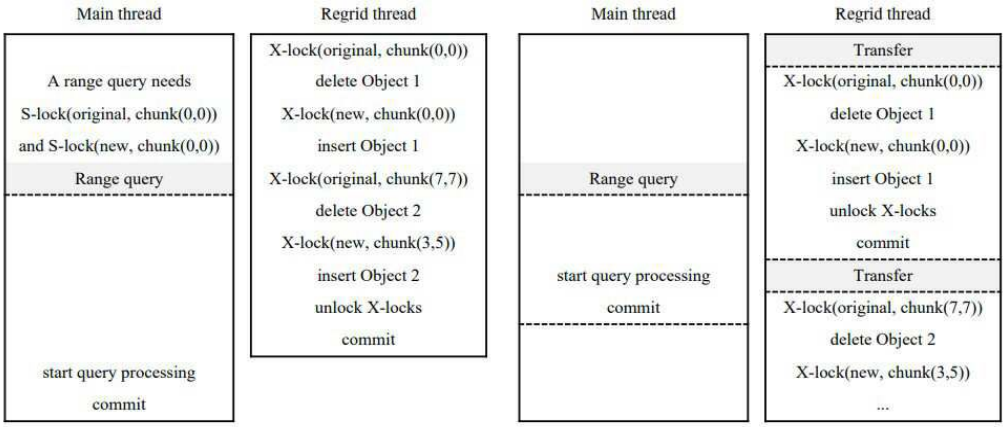

(a) $nCell^{chunk} = 1$ (b) $nCell^{chunk} = 4$

도면8



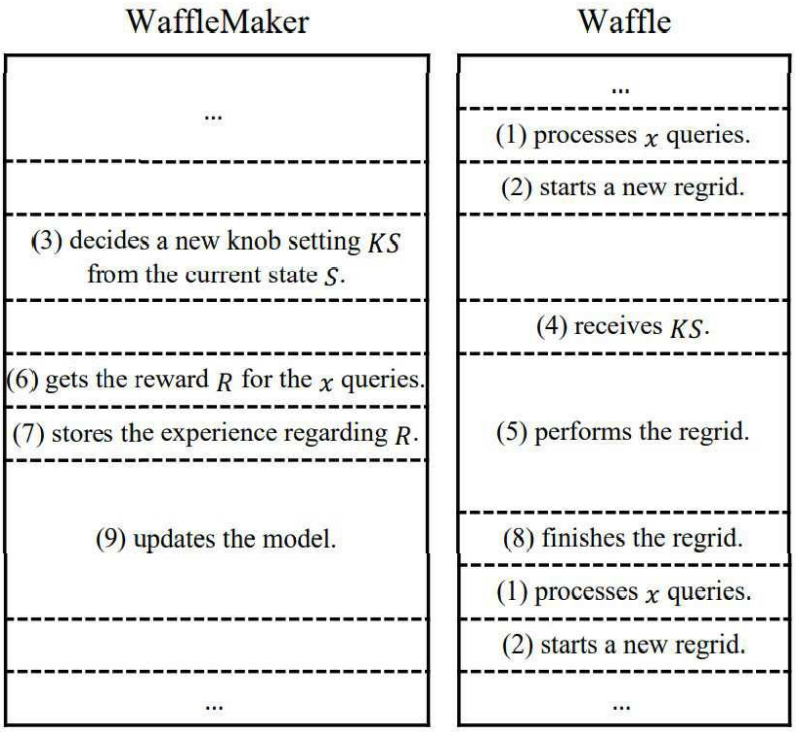

(a) $MOPC = 1$ (b) $MOPC = 4$

도면9

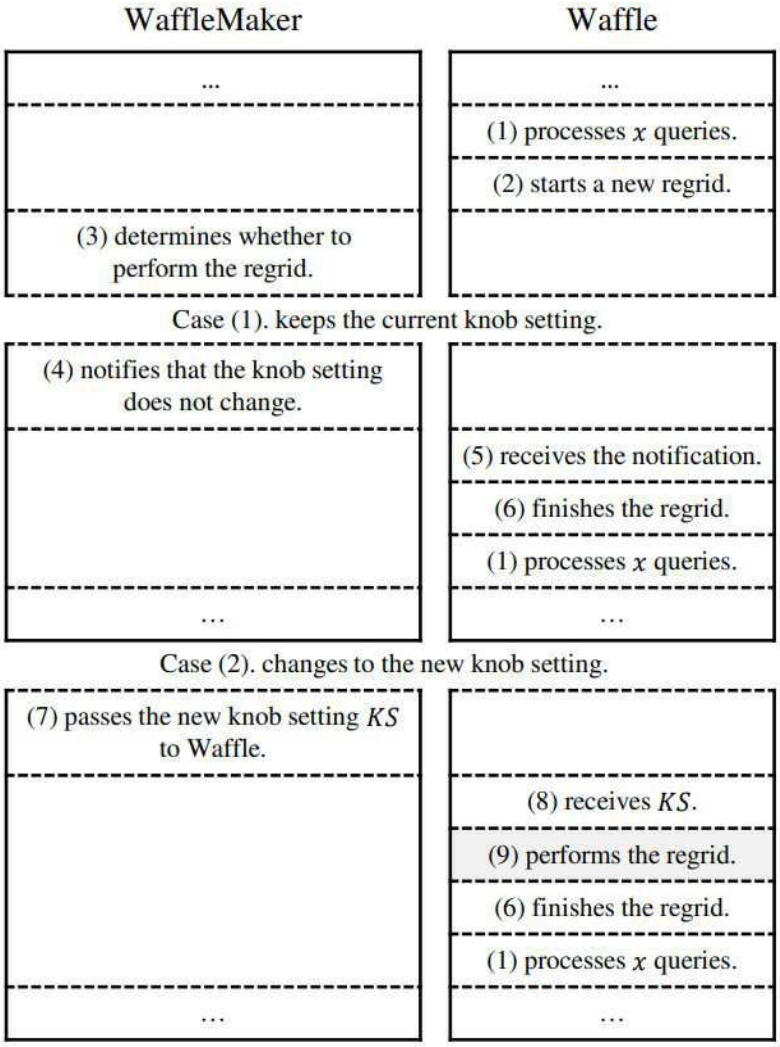



(a) Original regrid (b) Split regrid

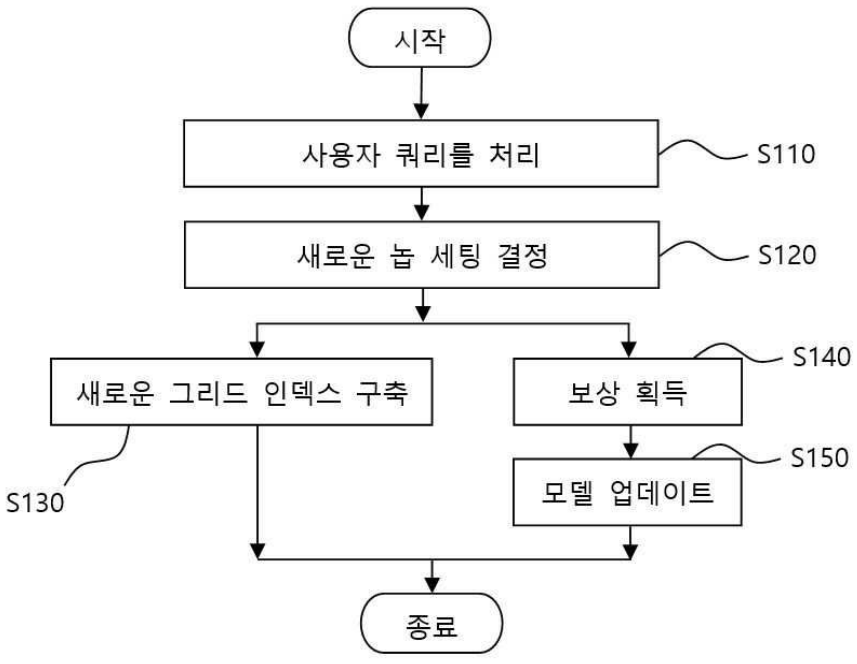
도면10


WaffleMaker
Waffle
...
(1) processes $x$ queries.
(2) starts a new regrid.
(3) decides a new knob setting $KS$ from the current state $S$.
(4) receives $KS$.
(6) gets the reward $R$ for the $x$ queries.
(5) performs the regrid.
(7) stores the experience regarding $R$.
(9) updates the model.
(8) finishes the regrid.
(1) processes $x$ queries.
(2) starts a new regrid.
...
...

도면11


WaffleMaker
Waffle
...
...
(1) processes $x$ queries.
(2) starts a new regrid.
(3) determines whether to perform the regrid.
Case (1). keeps the current knob setting.
(4) notifies that the knob setting does not change.
(5) receives the notification.
(6) finishes the regrid.
(1) processes $x$ queries.
...
...
Case (2). changes to the new knob setting.
(7) passes the new knob setting $KS$ to Waffle.
(8) receives $KS$.
(9) performs the regrid.
(6) finishes the regrid.
(1) processes $x$ queries.
...
...

도면12


시작
사용자 쿼리를 처리
S110
새로운 놉 세팅 결정
S120
새로운 그리드 인덱스 구축
S130
보상 획득
S140
모델 업데이트
S150
종료

도면13

