LOCK-FREE, SCALABLE READ ACCESS TO SHARED DATASTRUCTURES

기술 분야

본 명세서에 기술 된 주제는 공유 구조에 대한 잠금이없고 확장 가능한 판독 액세스를 제공하는 기술에 관한 것이다 데이터베이스 커널에 포함됩니다.

배경

공유 구조의 데이터 일관성을 보장하기 위해 많은 컴퓨터 프로그램에서 공유 데이터 구조에 대한 동기화 된 액세스가 필요합니다. 많은 경우에, 그러한 공유 구조는 상대적으로 거의 수정되지 않지만 꽤 자주 읽습니다. 데이터 일관성을 보장하기 위해. 이러한 구조는 기본 데이터 수정 전용이며 읽기 작업을 위해 공유되는 읽기 / 쓰기 잠금을 사용하여 잠글 수 있습니다. 그러나 읽기 / 쓰기 잠금은 특히 저렴한 동기화 프리미티브가 아니며 읽기 액세스조차도 CPU에서 L2- 캐시 미스를 유발하여 다중 코어 컴퓨팅 시스템의 성능을 심각하게 제한 할 수 있습니다.

이러한 문제는 완화 될 수 있지만 단일 읽기 / 쓰기 잠금에 대해 메모리 사용량이 많아 질 수 있습니다. 특히, 각 CPU 코어에 대해 하나의 메모리 캐시 라인을 예약하여 해당 코어에서 공유 잠금을 수행 할 수 있습니다.

독점 잠금이 없을 때 캐시 라인을 계산할 수 있습니다

선물 요청.

그러나, 이러한 구성에서는 적어도 두 가지 문제가 여전히 남아있다. 먼저, 배타적 액세스는 해당 작업이 완료 될 때까지 공유 구조의 읽기를 제외합니다. 특히 현대의 많은 코어 아키텍처가 이제 정기적으로 100 개 이상의 CPU 코어를 초과하므로 이러한 제한으로 인해 성능 병목 현상이 발생할 수 있습니다. 인 메모리 데이터베이스의 맥락에서 쿼리 실행 시간을 지배하는 I / O 시간이 없기 때문에 문제가 더욱 두드러집니다.

둘째, CPU 코어 당 하나의 캐시 라인을 사용하여 최적화 된 읽기 / 쓰기 잠금을 사용하더라도 과도한 수정로드로 인해 독점 잠금 대기 중에 높은 비율의 L2 캐시 미스가 발생할 수 있습니다.

이상적으로 공유 독자는 절대로 modi에 의해 차단되어서는 안됩니다

내부 구조의 허구

개요

일 양상에서, 객체 그래프의 적어도 하나의 객체의 적어도 하나의 판독 동작이 개시된다. 객체 그래프 char는 노드의 적어도 일부가 연관된 객체를 가리키는 다수의 에지를 지정하는 대응하는 기준 카운트를 갖는 루트 객체를 포함하는 객체의 계층을 작동시킨다. 호환되지 않는 변경뿐만 아니라 그래프의 객체에 대해 호환 가능한 변경을 수행 할 수 있습니다. 그 후, 각각의 대응하는 판독 동작에 대해, 루트 오브젝트의 참조 카운트가 증가된다 (루트 오브젝트에 대한 참조 카운트는 초기에 루트 오브젝트를 가리키는 단일 앵커 포인터를 반영 함). 이어서, 하나 이상의 객체가 변경된다. 호환되지 않는 변경으로 인해 루트 오브젝트, 하나 이상의 변경된 오브젝트 및 오브젝트 계층 구조 내의 모든 중간 오브젝트가 발생합니다.

복제되었습니다. 그런 다음 앵커 포인터가 복제 된 루트 개체에 연결됩니다. 루트 개체의 참조 횟수가 0 일 때 루트 개체는 나중에 삭제됩니다. 또한, 객체 그래프는 루트 객체에서 시작하여 변경된 객체에서 끝나고 해당 객체의 모서리를 제거합니다.

그런 다음 참조 카운트가 0 인 각 객체가 삭제됩니다.

일부 구현들에서, 기준 카운트는 둘 이상의 CPU에 걸쳐 스트라이핑 될 수있다. 그러한 배치는

L-2 캐시 미스를 줄인다는 점에서 유리합니다. 일부 또는 전부

객체는 메모리 내 데이터베이스에 저장 될 수 있습니다. 데이터베이스는 단조 증가하는 행 식별자로 행에 데이터를 저장할 수 있습니다. 데이터베이스는 다중 버전 통화 제어를 사용할 수 있습니다.

다른 양상에서, 테이블의 적어도 하나의 페이지의 판독 동작이 개시된다. 읽기 작업은 첫 번째 테이블 헤더 개체를 가리키는 앵커 개체를 사용하여 하나 이상의 페이지에 액세스합니다. 제 1 테이블 헤더 객체는 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함하는 제 1 링크 된 객체에 대한 링크를 포함한다. 그 후, 링크 된 오브젝트 및 테이블 헤더 오브젝트는 읽기 동작과 동시에 복제되어, 복제 된 링크 된 오브젝트는 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함한다. 후속 적으로, 앵커 오브젝트는 복제 된 테이블 헤더 오브젝트를 가리 키도록 설정됩니다. 하나의 추가 페이지를 복제 된 연결된 개체에 연결할 수 있습니다. 그 후, 판독 동작이 완료된 후 제 1 테이블 헤더 객체 및 제 1 링크 된 객체가 삭제된다.

테이블은 메모리 내 데이터베이스에 저장 될 수 있습니다. 제 1 링크 된 객체는 참조 카운트를 포함하는 버전 화 된 객체 일 수있다. 읽기 작업 중에 메모리 내 데이터베이스가 첫 번째 테이블 헤더 개체와 첫 번째 연결된 개체를 유지하도록하는 참조 횟수입니다. 복제 된 테이블 헤더 오브젝트는 처음에 첫 번째 링크 된 오브젝트에 링크 될 수 있습니다.

제 1 테이블 헤더 객체는 대응하는 페이지를 특징 짓는 관련된 메타 데이터를 가질 수있다. 복제 된 테이블 헤더 객체는 또한 제 1 테이블 헤더 객체와 관련된 메타 데이터와 연관 될 수있다.

비 일시적 컴퓨터 프로그램 제품도 적어도 하나의 컴퓨터의 하나 이상의 데이터 프로세서에 의해 실행될 때, 적어도 하나의 컴퓨터로 하여금 본 명세서에서의 동작을 수행하게하는 컴퓨터 실행 가능 명령어를 저장하는 것으로 설명 된도 1이 설명된다. 유사하게, 프로세서 및 프로세서에 연결된 메모리를 포함 할 수있는 컴퓨터 시스템이 또한 설명된다. 메모리는 프로세서로 하여금 본 명세서에 설명 된 하나 이상의 동작을 수행하게하는 하나 이상의 프로그램을 일시적으로 또는 영구적으로 저장할 수있다. 또한, 방법에 의해 지정된 동작은 단일 컴퓨팅 시스템 내에서 또는 둘 이상의 컴퓨팅 시스템간에 분산 된 하나 이상의 데이터 프로세서에 의해 구현 될 수있다.

본 명세서에 기술 된 주제는 많은 고급 태그를 제공한다. 예를 들어, 현재 주제와 함께 공유 데이터 구조에 액세스하려는 읽기 작업은 차단되지 않으므로 테이블 수정로드가 심한 경우에도 훨씬 더 나은 CPU 리소스 사용이 가능합니다.

본 명세서에 기술 된 주제의 하나 이상의 변형의 세부 사항은 첨부 도면 및 이하의 설명에 기재되어있다. 본 명세서에 기술 된 주제의 다른 특징 및 장점은 상세한 설명 및 도면 및 청구 범위로부터 명백 할 것이다.

도면의 설명

FIG. 1은 데이터 저장 애플리케이션을 포함하는 시스템을 도시 한 도면이다.

FIG. 2a는 데이터 구조에 대한 락 프리 스케일 러블 액세스를 나타내는 제 1 프로세스 흐름도이다.

FIG. 2b는 데이터 구조에 대한 락 프리 스케일 러블 액세스를 나타내는 제 2 프로세스 흐름도이다.

FIG. 3은도 1의 시스템의 세부 사항을 도시 한 도면이다. FIG1;

FIG. 4A-4K는 호환되지 않는 수정 동안 공유 데이터 구조에 대한 판독 액세스를 제공하기 위해 복제 된 테이블 헤더 객체 및 복제 된 링크 된 객체의 생성 및 사용을 도시 한 도면이다.

상세 설명

FIG1 1은 배치 될 수 있고, 하나 이상의 네트워크를 통해 링크 될 수있는 하나 이상의 프로그램 가능 프로세서를 포함 할 수 있고, 하나 이상의 모듈, 소프트웨어 구성 요소를 실행하는 컴퓨팅 시스템 (102) 또는 데이터 저장 애플리케이션 (104)은 데이터베이스, 기업 자원 프로그램, 분산 저장 시스템 (예를 들어 캘리포니아 주 서니 베일의 NetApp으로부터 입수 가능한 NetApp Filer) 등 중 하나 이상을 포함 할 수있다.

하나 이상의 모듈, 소프트웨어 컴포넌트 등은 컴퓨팅 시스템 (102)의 로컬 사용자가 액세스 할 수있다. 컴퓨팅 시스템에 액세스하는 원격 사용자뿐만 아니라 네트워크를 통해 하나 이상의 클라이언트 머신 (106)으로부터 (102)하나 이상의 제 1 모듈에 의해 생성 된 하나 이상의 사용자 인터페이스 스크린은 로컬 디스플레이 또는 클라이언트 머신 (106) 중 하나와 관련된 디스플레이를 통해 사용자에게 디스플레이 될 수있다. 데이터 저장 애플리케이션 (104)의 데이터 유닛 예를 들어 입력 / 출력 컴포넌트를 통해 하나 이상의 스토리지 (114)에 스토리지 페이지 형태로 데이터를 기록 할 수있는 지속성 계층 (112) (예를 들어, 페이지 버퍼 또는 다른 유형의 임시 지속성 계층)에 일시적으로 저장 될 수있다. 하나 이상의 스토리지 (114)는 더 오래 동안 데이터를 기록하도록 구성된 하나 이상의 물리적 스토리지 매체 또는 장치 (예를 들어, 하드 디스크 드라이브, 영구 플래시 메모리, 랜덤 액세스 메모리, 광학 매체, 자기 매체 등)를 포함 할 수있다. 용어 저장. 스토리지 (114) 및 입 / 출력 컴포넌트 (116)는도 1의 컴퓨팅 시스템 (102) 외부에 도시되어 있음에도 불구하고 컴퓨팅 시스템 (102)에 포함될 수 있음에 유의해야한다. FIG. 1

장기 저장 장치 (114)에 보유 된 데이터는 페이지로 조직화 될 수 있으며, 이들 각각은 정의 된 양의 저장 공간을 할당한다. 일부 구현들에서, 각 페이지에 할당 된 저장 공간의 양은 일정하고 고정 될 수있다. 그러나, 각 페이지에 할당 된 저장 공간의 양이 변할 수있는 다른 구현들도 현재 주제의 범위 내에있다.

FIG 2a는 210A에서, 객체 그래프의 적어도 하나의 객체의 적어도 하나의 판독 동작이 시작되는 방법 (200A)을 나타내는 프로세스 흐름도이다. 객체 그래프는 노드의 적어도 일부가 연관된 객체를 가리키는 다수의 에지를 특정하는 대응하는 기준 카운트를 갖는 루트 객체를 포함하는 객체의 계층을 특징으로한다. 호환되지 않는 변경뿐만 아니라 그래프의 객체에 대해 호환 가능한 변경을 수행 할 수 있습니다. 그 후, 220A에서, 대응하는 각각의 판독 동작에 대해, 대응하는 객체의 참조 카운트가 증가된다 (루트 객체에 대한 기준 카운트는 루트 객체를 가리키는 앵커 포인터를 초기에 반영 함). 이어서, 230A에서, 하나 이상의 객체가 변경된다. 호환되지 않는 변경은 240A에서 루트 오브젝트, 하나 이상의 변경된 오브젝트 및 복제되는 오브젝트의 계층 내에서 임의의 중간 오브젝트에서 발생합니다. 그런 다음 앵커 포인터는 250A에서 복제 된 루트 객체에 연결됩니다. 루트 개체의 참조 횟수가 0 일 때 루트 개체는 나중에 260A에서 삭제됩니다. 또한, 객체 그래프는 루트 객체에서 시작하여 이러한 객체에 대한 임의의 에지를 제거하는 적어도 하나의 변경된 객체에서 끝나는 270A에서 이송된다. 그런 다음 참조 카운트가 0 인 각 객체가 삭제됩니다.

FIG 2b는 210B에서 인 메모리 데이터베이스의 메모리에 저장된 테이블의 적어도 하나의 페이지의 판독 동작이 시작되는 프로세스 흐름도 (200B)이다. 읽기 작업은 첫 번째 테이블 헤더 개체를 가리키는 앵커 개체를 사용하여 하나 이상의 페이지에 액세스합니다. 제 1 테이블 헤더 객체는 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함하는 제 1 링크 된 객체에 대한 링크를 포함한다. 그 후, 220B에서, 링크 된 객체 및 테이블 헤더 객체는 판독 동작과 동시에 복제된다. 복제 된 링크 된 객체는 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함한다. 다음으로 230B에서 앵커 오브젝트는 복제 된 테이블 헤더 오브젝트를 가리 키도록 설정되고, 240B에서, 하나 이상의 추가 페이지가 복제 된 링크 된 오브젝트에 링크됩니다. 이어서, 250B에서, 판독 동작이 완료된 후 제 1 테이블 헤더 객체 및 제 1 링크 된 객체가 삭제된다.

다르게 말하면, 읽기 조작은 오브젝트 그래프의 루트 오브젝트 (별칭 헤더 오브젝트)로 시작하고 (예 : 테이블 헤더)이 오브젝트에 하나의 (가상) 참조를 추가합니다. 이 객체 그래프 (유향 그래프)에서 다른 객체의 참조를 통해 연결된 추가 객체를 가질 수 있습니다. 일반적으로이 그래프는 트리입니다 (즉, 사이클이 없음). 객체는 그래프의 노드 / 정점이며 참조는 가장자리 (방향)입니다. 그래프의 어느 위치에서 개체에 대한 각 참조가 계산되므로 개체의 참조 수는이 특정 개체를 가리키는 가장자리 수입니다. 루트의 참조 수는 1 개 (앵커의 경우)에 독자 수를 더한 값입니다. 이 참조 카운트는 실제로 L2- 캐시충돌을 방지하기 위해 CPU 코어에 스트라이프로 저장됩니다.

호환되는 변경은 그래프 또는 참조 카운트를 변경하지 않으며 그래프의 개체 중 하나만 변경됩니다. 호환되지 않는 변경 사항은 영향을받는 개체와 그래프의 루트까지 루트 (헤더 / 헤더 개체 포함)까지의 모든 개체를 복제하여 영향을받는 개체 사이에 새 가장자리를 만들고 새로 복제 된 개체와 이전 버전의 원본에서 연결된 개체 사이에 새 가장자리를 만듭니다 각각의 객체 (프로세스에서 참조 횟수 증가). 다른 모든 객체는 동일하게 유지되고 원래 객체 그래프에서 다시 연결됩니다 (즉, 원래 루트 / 헤더 객체와 복제 된 객체의 두 루트가있는 그래프가 나타남). 앵커 참조가 복제 된 참조로 전송되면 원래 루트 객체는 참조를 잃습니다. 판독기가없는 경우 참조 카운트가 0으로 떨어 지므로이 개체가 제거됩니다. 전 이적으로이 객체에서 시작하는 가장자리도 제거되어 연결된 객체의 참조 횟수가 감소하고 결과적으로 제거됩니다. 제거 할 것이 없을 때까지 계속됩니다. 리더가있는 경우 여전히 원래 루트 / 헤더 객체에 대한 참조를 보유하므로 전체 프로세스는 마지막 리더가 완료된 후에 만 ​​발생합니다.

FIG 3은 소프트웨어 아키텍처 (300)가 현재 주제의 하나 이상의 특징. 하나 이상의 하드웨어 및 소프트웨어로 구현 될 수있는 데이터 저장 애플리케이션 (104)은 하나 이상의 데이터베이스 애플리케이션, 네트워크 접속 스토리지 시스템 등을 포함 할 수있다. 현재 주제의 적어도 일부 구현들에 따르면, 이러한 데이터 저장 애플리케이션 (104)은 예를 들어 지속성 인터페이스 (302)를 통해 지속성 계층 (112) 또는 다른 유형의 메모리 버퍼를 포함하거나 그렇지 않으면 인터페이스 할 수있다. 지속성 계층 내의 페이지 버퍼 (304) 112는 하나 이상의 논리 페이지 (306)를 저장할 수 있고, 선택적으로 새도우 페이지, 활성 페이지 등을 포함 할 수있다. 지속성 계층 (112)에 보유 된 논리 페이지 (306)는 입력 / 출력 컴포넌트 (116)를 통해 스토리지 (예를 들어, 용어 저장 등) (114)에 기록 될 수 있으며, 이는 소프트웨어 모듈, 하나 또는 그로 구현 된 서브 시스템 일 수있다. 더 많은 소프트웨어 및 하드웨어 등. 스토리지 (114)는 저장된 페이지 (312)가 물리적 메모리 블록에 할당되는 하나 이상의 데이터 볼륨 (310)을 포함 할 수있다.

일부 구현들에서, 데이터 저장 애플리케이션 (104) 페이지 관리자 (314)는 페이지 상태를 모니터링하는 자유 블록 관리자 (322)를 포함 할 수있는 지속 계층 (112)에서 페이지 관리 모듈 (320)과 통신 할 수있다. 정보 (324), 예를 들어 스토리지 (114) 내의 물리적 페이지의 상태 및 지속 레이어 (112) (및 선택적으로 페이지 버퍼 (304))의 논리 페이지. 세이브 포인트 관리자 (316)는 지속성 계층 (204)에서 세이브 포인트 코디네이터 (326)와 통신하여 세이브 포인트를 처리 할 수 ​​있으며,이 세이브 포인트는 가능한 충돌 후 재시작을 위해 데이터베이스의 일관된 지속 상태를 생성하는데 사용된다.

데이터 저장 애플리케이션 (104)의 일부 구현들에서 지속성 계층 (112)의 페이지 관리 모듈은 쉐도우 페이징을 구현할 수있다. 자유 블록 관리자 322 페이지 관리 모듈 (320) 내에서 물리적 페이지의 상태를 유지할 수있다. 페이지 버퍼 (304)는 ​​본 명세서에서 논의 된 바와 같이 동작하는 고정 페이지 상태 버퍼를 포함 할 수있다. 페이지 관리 모듈 (320)의 일부이거나 그와 통신 할 수있는 변환기 컴포넌트 (340)는 스토리지 (114)에 기록 된 논리 페이지와 물리 페이지 사이의 맵핑을 담당 할 수있다. 변환기 (340)는 논리 페이지의 현재 맵핑을 변환기 (340)는 논리 페이지 (306)가 하나 이상의 변환기 테이블 (342) 내의 대응하는 물리 페이지에 대한 논리 페이지 (306)의 현재 맵핑을 유지할 수있다. 논리 페이지 (306)가 스토리지 (114)로부터 판독 될 때, 스토리지 페이지 저장 페이지 이후에 논리 페이지가 스토리지 (114)에 기록 될 때, 새로운 자유 물리 페이지가 논리 페이지에 할당된다. 자유 블록 관리자 (322)는 새로운 물리적 페이지를 "사용됨"으로 표시하고 새로운 매핑은 더 많은 변환기 테이블 342.

지속성 계층 (112)은 데이터 저장 애플리케이션 (104)에서 이루어진 변경이 내구성이 있고 데이터 저장 애플리케이션 (104)이 재시작 후 가장 최근의 커밋 된 상태로 복원 될 수 있도록 보장 할 수있다. 스토리지 (114)에 데이터를 기록하는 것은 기록 트랜잭션의 종료와 동기화 될 필요가 없다. 따라서 커밋되지 않은 변경 사항은 디스크에 기록 될 수 있으며 쓰기 트랜잭션이 완료되면 커밋 된 변경 사항이 디스크에 아직 기록되지 않을 수 있습니다. 시스템 충돌 후 완료되지 않은 트랜잭션에 의한 변경 사항은 롤백 할 수 있습니다. 이 과정에서 이미 커밋 된 트랜잭션으로 발생하는 변경 내용을 잃어 버려서는 안됩니다. 로거 구성 요소 (344)는 또한 데이터 저장 애플리케이션의 데이터에 대한 변경을 선형 로그에 저장하기 위해 포함될 수있다. 로거 구성 요소 (344)는 모든 작업이 데이터에 적용되고 기록 된 "커밋 레코드를 갖는 트랜잭션이 복구 종료시 여전히 열린 트랜잭션을 롤백하기 전에 커밋되도록 보장하기 위해 마지막 저장 점 이후 복구 중에 조작을 재생하는 데 사용될 수있다" 방법

일부 데이터 저장 응용 프로그램의 경우 디스크에 데이터를 쓰는 것이 반드시 쓰기의 끝과 동기화되지는 않습니다 트랜잭션. 커밋되지 않은 상황이 발생할 수 있습니다. 변경 내용은 디스크에 기록되는 동시에 기록 트랜잭션이 완료 될 때 커밋 된 변경 내용은 아직 디스크에 기록되지 않습니다. 시스템 충돌 후, 완료되지 않은 트랜잭션의 변경 사항은 롤백되어야하고 커미트 된 트랜잭션의 변경 사항은 유실되지 않아야합니다.

커밋 된 변경이 손실되지 않도록하기 위해, 변경이 이루어질 때마다 로거 구성 요소 (344)에 의해 리두 로그 정보가 기록 될 수있다. 이 정보는 트랜잭션이 종료 될 때 최신 디스크에 기록 될 수 있습니다. 일반 데이터가 데이터 볼륨에 기록되는 동안 로그 항목은 별도의 로그 볼륨에 유지 될 수 있습니다. 다시 실행 로그를 사용하면 해당 데이터 페이지가 디스크에 기록되지 않은 경우에도 커밋 된 변경 내용을 복원 할 수 있습니다. 커밋되지 않은 변경을 취소하기 위해, 지속성 계층 (112)은 (하나 이상의 로그로부터의) 실행 취소 로그 엔트리와 섀도 페이징의 조합을 사용할 수있다.

지속성 인터페이스 (302)는 상점 (예를 들어, 메모리 내 상점 등)의 판독 및 기록 요청을 처리 할 수있다. 지속성 인터페이스 (302)는 또한 로깅 및 로깅없이 데이터를 기록하기위한 기록 방법을 제공 할 수있다. 기록 된 기록 동작이 사용되는 경우, 지속성 인터페이스 (302)는 로거 (344)를 호출한다. 또한, 로거 (344)는 저장소 (예를 들어, 인 메모리 저장소 등)가 로그 엔트리를 로그 큐에 직접 추가 할 수있는 인터페이스를 제공한다. 로거 인터페이스는 또한 메모리 로그 큐의 로그 항목을 디스크로 플러시하도록 요청하는 방법을 제공합니다.   
 로그 항목에는 로그 시퀀스 번호, 로그 항목 유형 및 트랜잭션 식별자가 포함됩니다. 작업 유형에 따라 추가 정보가 로거 (344)에 의해 기록됩니다. 예를 들어 "update"유형의 항목의 경우 영향을받는 레코드와 그 이후의 식별 수정 된 데이터의 이미지.   
 데이터 애플리케이션 (104)이 재시작 될 때, 로그 엔트리는 처리 될 필요가있다. 이 프로세스의 속도를 높이기 위해 재실행 로그가 항상 처음부터 처리되는 것은 아닙니다. 대신 위에서 언급 한 것처럼 마지막 저장 점 이후에 작성된 모든 디스크 변경 사항 (예 : 메모리 등)을 기록하는 저장 점을 주기적으로 수행 할 수 있습니다. 시스템을 시작할 때 로그 만 마지막 세이브 포인트를 처리 한 후 생성됩니다. 다음 백업 조작 후 저장 점 위치 이전의 이전 로그 항목을 제거 할 수 있습니다.

로거 (344)가 로그 엔트리를 기록하기 위해 호출 될 때, 로거는 즉시 디스크에 기록하지 않는다. 대신 로그 항목을 메모리의 로그 큐에 넣을 수 있습니다. 로그 큐의 항목은 해당 트랜잭션이 완료 (커밋 또는 중단) 될 때 디스크에 기록 될 수 있습니다. 커밋 된 변경 내용이 손실되지 않도록하려면 해당 로그 항목을 디스크로 플러시하기 전에 커밋 작업이 성공적으로 완료되지 않습니다. 로그 큐 페이지가 가득 찼거나 저장 점이 수행 될 때와 같이 다른 이벤트에 의해 로그 큐 항목을 디스크에 쓰는 것도 트리거 될 수 있습니다.

현재의 주제와 관련하여, 로거 (344)는 자연 순서 (예를 들어, 순차적 순서 등)로 메모리 버퍼에 순차적으로 데이터베이스 로그 (또는 간단히 "로그"로 지칭 됨)를 기록 할 수있다. 저장 장치는 로그 데이터를 저장하는 데 사용되며 여러 로그 파티션을 정의 할 수 있습니다. 그 후, 로거 (344) (상기 언급 된 바와 같이 로그 데이터를 생성 및 구성 함)는 이용 가능한 로그 파티션 전체에 대한 로그 버퍼에 대한 기록을로드 밸런스 할 수있다. 경우에 따라로드 균형 조정은 다양한 쓰기 작업이 순차적이고 연속적인 방식으로 로그 버퍼에 전달되는 라운드 로빈 배포 방식에 따릅니다. 이러한 배열에서, 다중 파티션 로그의 특정 파티션의 단일 로그 세그먼트에 기록 된 로그 버퍼는 연속적이지 않습니다. 그러나 복구 중에 모든 파티션의 로그 세그먼트에서 올바른 순서로 로그 버퍼를 다시 정렬 할 수 있습니다.

위에서 언급 한 바와 같이, 데이터 저장 애플리케이션 (104)은 쉐도우 페이징을 사용하여 세이브 포인트 관리자 (316)가 트랜잭션 적으로 일관된 세이브 포인트를 기록 할 수있다. 이러한 구성으로, 데이터 백업은 특정 저장 점에 포함 된 모든 데이터 페이지의 사본을 포함하며, 이는 데이터 백업 프로세스의 첫 단계로서 수행된다. 현재 주제는 다른 유형의 데이터 페이지 스토리지에도 적용될 수 있습니다.

데이터 저장 애플리케이션 (104)은 트랜잭션 격리 및 일관된 판독을 위해 다중 버전 동시 제어 (MVCC)를 이용할 수있다. 데이터베이스의 각 행은 고유 한 단조 증가 식별자 (RowlD)와 연관 될 수 있습니다. 레코드의 새 버전이 작성되면이 새 버전도 새 RowlD가 될 수 있습니다 (예 : MVCC 의미로 인해 이전 버전은 병렬 리더를 위해 유지되어야하며 커밋 후 가비지 콜렉션 중에 만 정리됩니다).

본 명세서에서 페이지에 대한 언급은 데이터 저장 애플리케이션 (104)의 일부를 구성하는 메모리 내 데이터베이스의 메모리에 저장된 테이블의 페이지를 지칭 할 수있다. MVCC 기반 데이터베이스 테이블 구현으로, 테이블의 모든 내부 일시적 데이터 오브젝트가 버전 화 될 수있다. 이들 데이터 객체는 테이블, 헤더 객체, 메타 데이터 객체,로드 된 페이지의 벡터, 압축 된 원주 테이블을위한 사전 해시 / 트리 등과 같은 다른 내부 상태 (들)를 포함 할 수있다.

현재의 주제로, 독자가 사용하는 모든 테이블 제어 구조의 버전을 지정할 수 있습니다. 이러한 구조에는 예를 들어 페이지 목록, 값 간접 벡터, 내부 메타 데이터 등이 포함됩니다. 독자는 데이터 구조에 대한 잠금을 얻지 않고 쿼리 또는 쿼리 계획 연산자가 끝날 때까지 현재 버전의 데이터 구조로 작업합니다. 이 배열을 사용하면 이전 버전은 짧은 시간 동안 만 유지됩니다 (예 : 서브 초). AS 버전의 객체는 일반적으로 작고 메모리 오버 헤드도 작습니다. 또한 OLTP 시스템의 경우에도 호환되지 않는 변경은 거의 없습니다 (즉, 동시 버전이 많지 않음). 또한 일부 구현에서는 이전 버전의 우선 순위가 큰 / 큰 개체가

(예를 들어, 원주 테이블의 주요 부분 등) 여전히 존재하며, 해당 객체의 새 버전을 만들 수 없습니다. 예를 들어, 컬럼 형 테이블이 버전 n-1에서 버전 n으로 병합되는 동안 시작된 컬럼 형 테이블의 주요 부분에서 스캔을 수행하는 리더가있는 경우이 스캔은 버전 n-1의 주요 부분을 사용합니다. 버전 n으로의 병합이 완료된 후에도 버전 n-1의 주요 부분에서 스캔이 실행되는 한 버전에서 버전 n + 1 로의 추가 병합이 방지됩니다 (메모리 요구가 엄청나게 증가 할 수 있음).

------------------------------------------------------------

다음은 도시화 된 다이어그램 (400A - 400K)을 설명한다. 4A-4J. 메모리에서 테이블을 나타내는 객체는 트리로 구성 될 수 있으며, 트리에서 테이블 헤더 객체 (402)에 뿌리를두고 앵커 포인터 (404)에 의해 지시된다. 앵커 포인터 (404)는 테이블의 루트 객체를 지시하는데 사용된다. 모든 버전 관리 객체와 마찬가지로 테이블 헤더 객체에는 참조 횟수가 있습니다. 도 3의 다이어그램 (400A)에서 4a에서, 참조 카운트는 1이다. 메타 데이터 (406)는 특성화 할 수있다

헤더 객체 (402)에 의해 표현 된 테이블에 대한 다양한 양상들 (즉, 테이블 열들 및 그들의 데이터 타입들, 제약 사항 등을 정의하기 위해). 테이블은 (페이지 핸들을 통해) 복수의 페이지 (410)에 링크되는 페이지 벡터 객체 (408)에 링크된다.

다음은도 1 내지도 3에 도시 된 다이어그램 (400A 400K)을 참조한다. 4A-4J. 메모리에서 테이블을 나타내는 객체는 트리로 구성 될 수 있으며, 트리에서 테이블 헤더 객체 (402)에 뿌리를두고 앵커 포인터 (404)에 의해 지시된다. 앵커 포인터 (404)는 테이블의 루트 객체를 지시하는데 사용된다. 모든 버전 관리 객체와 마찬가지로 테이블 헤더 객체에는 참조 횟수가 있습니다. 도 3의 다이어그램 (400A)에서. 4a에서, 참조 카운트는 1이다. 메타 데이터 (406)는 특성화 할 수있다

헤더 객체 (402)에 의해 표현 된 테이블에 대한 다양한 양상들 (즉, 테이블 열들 및 그들의 데이터 타입들, 제약 사항 등을 정의하기 위해) 테이블은 페이지 벡터 객체 (408)에 링크되고, 페이지 벡터 객체 (408)는 (페이지 핸들을 통해)로드 된 인 메모리 테이블의 복수의 페이지 (410) (이 예에서는 제 1 페이지 (410) 및 제 2 페이지 (410)에 링크 됨)에 링크된다. 기억. 페이지 핸들은 메모리에로드 된 페이지를 고정하는 특수한 유형의 포인터입니다. 현재 배열에서, 내부 구조에 대한 수정은 뮤텍스 잠금과 같은 일부 수단에 의해 또는 메시지 전달 서브 시스템의 단일 워커에서 모든 수정 동작을 실행함으로써보다 복잡한 구현에 의해 항상 동시 액세스에 대해 동기화 될 수있다.

현재 주제는 다양한 상이한 객체와 관련하여 이용 될 수 있고 페이지 벡터 객체는 많은 구현 중 하나로 사용된다는 것을 이해할 것이다. 예를 들어, 현재 주제는 사전 값 벡터, 사전 해시 및 / 또는 검색 트리, 다양한 메타 데이터 객체 등과 같은 객체에 적용될 수 있습니다. 또한 페이지 벡터의 경우 델타 부분에 대해 하나의 페이지 벡터가있을 수 있습니다. 병합 조작을 실행하는 동안 작성된 델타 테이블과 테이블. 주요 부분 열에 대한 여러 페이지 벡터 (또는 선형 공간에 여러 압축 벡터)가있을 수도 있습니다.

일반적으로 두 가지 유형의 수정, 즉 호환 가능한 수정과 호환되지 않는 수정이 있습니다. 호환 가능한 수정의 예는 벡터 크기를 조정할 필요가없는 한 메모리에로드 된 메모리 내 테이블의 모든 페이지를 보유하는 페이지 벡터에 새 페이지 핸들을 추가하는 것입니다.

현재의 주제는 호환되지 않는 수정을 다룹니다. 다시도 1의 다이어그램 (400A)을 참조하면. 도 4a에 도시 된 바와 같이, 페이지 벡터 (408)는 최대 4 개의 페이지에 링크되도록 크기가 정해진 다. (도 4b의 다이어그램 (400B)에 도시 된 바와 같이) 제 3 페이지 (410)를 추가하는 것은 페이지 벡터 (408)의 크기 조정을 필요로하지 않는다. 데이터 구조의 변경으로 인해 제 3 페이지 (410)가 추가된다. 이 제 3 페이지 (410)는이어서 페이지 벡터 (408)에 링크되고, 테이블 헤더 객체의 참조는 헤더 테이블 객체 (402)의 새로운 버전을 반영하기 위해 2로 증가된다. 그 후, 판독 동작의 일부로서, 다이어그램을 참조하여 도 4의 400C 도 4c에서, 하나 이상의 링크 된 페이지 (410)의 메모리에서 위치를 식별하기 위해 테이블 헤더 객체 (402)에 액세스하는 판독기 (412)가 시작된다.

도 3의 다이어그램 (400D)을 참조하면 도 4d에 도시 된 바와 같이, 데이터 구조로 변경되면 제 4 페이지 (410)가 추가된다. 이어서,이 제 4 페이지 (410)는 페이지 벡터 (408)에 링크된다. 도 4e에 도시 된 바와 같이, 데이터 구조로 변경되면 제 5 페이지 (410s)가 추가된다. 이 경우, 페이지 벡터 (408)는 제 5 페이지 (410s)에 대한 핸들을위한 공간을 갖지 않는다. 제 5 페이지 (410s)를 수용하기 위해, 페이지 벡터 (408)는 (도 4f의 다이어그램 400F에 도시 된 바와 같이) 복제 페이지 벡터 (414)로 복제 / 복사되어 처음 4 개의 페이지 (410a)에 대한 핸들을 갖는다. 또한,도 3의 다이어그램 (400G)을 참조하면 도 4g에 도시 된 바와 같이, 테이블 헤더 오브젝트 (402)는 클론 테이블 헤더 오브젝트 (416)로 복제되고,도 4의 다이어그램 (400H)을 참조한다. 도 4h에서, 복제 된 테이블 헤더 객체 (416)는 원본 페이지 벡터 (408)로부터 복제 된 페이지 벡터 객체 (414)를 가리 키도록 업데이트된다.

다음으로,도 3의 다이어그램 (400I)을 참조하여 도 4i에 도시 된 바와 같이, 앵커 포인터 객체 (404)는 복제 된 테이블 헤더 객체 (416)로 설정되고 복제 된 테이블 헤더 객체 (416)의 참조 번호는 0에서 1로 증가하고 원래 테이블 헤더 객체 (404)의 참조 번호는 2에서 1로 감소된다 (앵커에서 원본 테이블 헤더로의 참조가 삭제되고 새 헤더 객체를 가리 키도록 앵커 포인트를 업데이트하여 새 테이블 헤더에 대한 참조가 추가 되었기 때문에). 복제 된 페이지 벡터 (414)는 적어도 5 개의 페이지에 대한 핸들을 저장할 수 있고, 따라서,도 4j의 다이어그램 400J를 참조하면, 제 5 페이지 (410s)는 복제 된 페이지 벡터 객체 (414)에 링크된다. 리더 (412)는 다이어그램 400J를 참조한다. 도 1의 도 4j는 구 테이블 헤더 객체 (402)를 사용하여 처음 4 개의 페이지 (410a) 중 어느 하나에 저장된 데이터를 여전히 읽을 수있다. 도 4k에 도시 된 바와 같이, 제 6 페이지 (410)가 추가되고 복제 된 페이지 벡터 객체 (414)에 링크 될 수있다. 도 4k에서, 원본 테이블 헤더 객체 (402) 및 원본 페이지 벡터 객체 (408)는 더 이상 참조가 없기 때문에 제거 될 수있다.

위에서 알 수있는 바와 같이, 영향을받는 데이터 구조 (즉, 페이지 벡터 객체 (412))가 복제되고 앵커 객체 (404)로부터 수정 된 객체까지의 경로상의 모든 객체 (예를 들어, 테이블 헤더 객체 (402))가 복제된다 . 복제본은 객체의 얕은 사본으로 간주 될 수 있습니다. 새로운 버전의 자식 개체 (이 경우 새로운 크기 조정 된 벡터) 만 부모의 복제본에 고정되고 다른 자식 개체는 해당 참조 카운트를 증가시킵니다 (그림에는 표시되지 않음). 복제가 완료되면 앵커 포인터가 새로 복제 된 루트 개체 (이 경우 테이블 헤더 개체)를 가리 키도록 업데이트됩니다.

또한 데이터베이스는 MVCC를 사용하여 데이터베이스 테이블 데이터에 액세스하므로 이전 데이터 구조와 새 데이터 구조 모두에서 동일한 데이터 세트를 읽을 수 있습니다. 위의 예에서 새 페이지에 기록 된 데이터는 실행중인 판독기 트랜잭션에 표시되지 않으므로 판독기가 읽을 수없는 것은 아닙니다. 마찬가지로 기존 페이지에 새 데이터를 추가하는 호환 가능한 변경 사항도 이전 데이터 만 볼 수 있으므로 독자에게는 변하지 않습니다. 내부 구조의 일관된 더티 (dirty) 읽기를 보장하는 방식 (예 : 벡터에 쓰기는 먼저 새 요소를 쓴 다음 요소 수를 늘리고 리더기 OTOH가 먼저 읽은 다음 요소에 액세스) ). 라이터가 먼저 요소 수를 늘린 다음 요소를 쓰면 독자는 경쟁 조건을 갖게됩니다. 이미 새 요소 수를 읽고 아직 작성되지 않은 요소 (초기화되지 않은 메모리)에 액세스 할 수 있으며 이는 일반적으로 충돌 및 / 또는 잘못된 결과.

L2- 캐시 친 화성을 보장하기 위해 앵커 포인터 객체는 각 CPU 코어에 대해 개별적으로 루트 객체 및 루트 객체 포인터의 참조 카운트를 포함 할 수 있습니다. 따라서 리더는 원자 연산을 사용하여 코어 전용 캐시 라인 만 수정하여 참조 카운트를 늘리고 동시에 현재 앵커 포인터를 읽습니다. 호환되지 않는 구조 변경이 발생하면 모든 코어-개인 카운터의 참조 카운트를 루트 개체의 참조 카운트로 하나씩 집계 할 수 있으며 앵커 포인터 개체를 원자 적으로 업데이트하여 새로운 루트 포인터와 제로 참조 카운트를 포함합니다 (최종 반복) 리더와 충돌하는 경우 작동). 오래된 구조 (다른 앵커 포인터)를 남겨둔 독자는 코어-개인 캐시의 참조 횟수 대신 루트 버전 (이 예에서는 테이블 헤더 개체)의 이전 버전에서 공유 참조 횟수를 원자 적으로 감소시켜야합니다. 앵커 포인터의 선.

본 명세서에 기술 된 주제의 양태는 원하는 구성에 따라 시스템, 장치, 방법 및 / 또는 물품으로 구현 될 수있다. 특히, 본 명세서에 기술 된 주제의 다양한 구현은 디지털 전자 회로, 집적 회로, 특별히 설계된 주문형 집적 회로 (ASIC), 컴퓨터 하드웨어, 펌웨어, 소프트웨어 및 / 또는 이들의 조합에서 실현 될 수있다. 이러한 다양한 구현은 특수하거나 범용 일 수 있고, 데이터 및 명령을 수신하고 데이터를 전송하기 위해 결합 될 수있는 적어도 하나의 프로그램 가능 프로세서를 포함하는 프로그램 가능 시스템상에서 실행 가능하고 /하거나 해석 가능한 하나 이상의 컴퓨터 프로그램에서의 구현을 포함 할 수있다. 및 저장 시스템, 적어도 하나의 입력 장치 및 적어도 하나의 출력 장치에 대한 명령을 포함한다.

프로그램, 소프트웨어, 소프트웨어 응용 프로그램, 응용 프로그램, 구성 요소 또는 코드라고도 할 수있는 이러한 컴퓨터 프로그램에는 프로그래밍 가능 프로세서에 대한 기계 명령어가 포함되어 있으며 고급 절차 및 / 또는 객체 지향 프로그래밍 언어로 구현 될 수 있습니다. 및 / 또는 조립 / 기계 언어로. 본 명세서에서 사용되는 용어 "기계 판독 가능 매체"는 임의의 컴퓨터 프로그램 제품, 장치 및 / 또는 장치를 지칭한다. 예를 들어, 기계 판독 가능 신호로서 기계 명령을 수신하는 기계 판독 가능 매체를 포함하여 기계 명령 및 / 또는 데이터를 프로그램 가능 프로세서에 제공하는데 사용되는 자기 디스크, 광 디스크, 메모리 및 프로그램 가능 논리 장치 (PLDS)와 같은 . "기계 판독 가능 신호"라는 용어는 기계 명령 및 / 또는 데이터를 프로그램 가능 프로세서에 제공하는데 사용되는 임의의 신호를 지칭한다. 기계 판독 가능 매체는 예를 들어 비 일시적 솔리드 스테이트 메모리 또는 자기 하드 드라이브 또는 임의의 등가의 저장 매체와 같은 이러한 기계 명령을 비 일시적으로 저장할 수있다. 머신 판독 가능 매체는 예를 들어 프로세서 캐시 또는 하나 이상의 물리적 프로세서 코어와 관련된 다른 랜덤 액세스 메모리와 같은 일시적인 방식으로 이러한 머신 명령어를 대안 적으로 또는 추가적으로 저장할 수있다

사용자와의 상호 작용을 제공하기 위해, 여기에 설명 된 주제는 예를 들어 사용자에게 정보를 표시하기위한 음극선 관 (CRT) 또는 액정 디스플레이 (LCD) 모니터와 같은 디스플레이 장치를 갖는 컴퓨터에서 구현 될 수있다. 및 키보드 및 포인팅 장치. 예를 들어 사용자가 컴퓨터에 입력을 제공 할 수있는 마우스 또는 트랙볼과 같은 것. 사용자와의 상호 작용을 제공하기 위해 다른 종류의 장치가 사용될 수있다. 예를 들어, 사용자에게 제공되는 피드백은 예를 들어 시각 피드백, 청각 피드백 또는 촉각 피드백과 같은 임의의 형태의 감각 피드백 일 수 있고; 사용자로부터의 입력은 음향, 음성 또는 촉각 입력을 포함하지만 이에 제한되지 않는 임의의 형태로 수신 될 수있다. 다른 가능한 입력 장치로는 단일 또는 멀티 포인트 저항 또는 용량 성 트랙 패드, 음성 인식 하드웨어 및 소프트웨어, 광학 스캐너, 광학 포인터, 디지털 이미지 캡처 장치 및 관련 해석 소프트웨어와 같은 터치 스크린 또는 기타 터치 감지 장치가 포함됩니다. 등.

본 명세서에 기술 된 주제는 예를 들어 하나 이상의 데이터 서버와 같은 백엔드 구성 요소를 포함하거나, 예를 들어 하나 이상의 애플리케이션 서버와 같은 미들웨어 구성 요소를 포함하거나, 또는 다음을 포함하는 컴퓨팅 시스템에서 구현 될 수있다. 예를 들어, 그래픽 사용자 인터페이스를 갖는 하나 이상의 클라이언트 컴퓨터 또는 사용자가 본 명세서에 기술 된 주제의 구현과 상호 작용할 수있는 웹 브라우저와 같은 프론트 엔드 구성 요소, 또는 이러한 백엔드 미들웨어의 임의의 조합 또는 프론트 엔드 구성 요소. 클라이언트와 서버는 일반적으로 서로 배타적이지 않지만 일반적으로 통신 네트워크를 통해 상호 작용하지만 시스템의 구성 요소는 모든 형태 또는 매체의 디지털 데이터 통신으로 상호 연결될 수 있습니다. 통신 네트워크의 예는 LAN (Local Area Network), WAN (Wide Area Network) 및 인터넷을 포함하지만 이에 제한되지는 않는다. 클라이언트와 서버의 관계는 각 컴퓨터에서 실행되고 서로 클라이언트-서버 관계를 갖는 컴퓨터 프로그램으로 인해 발생합니다.

전술 한 설명에서 제시된 구현들은 본 명세서에서 설명 된 주제와 일치하는 모든 구현들을 나타내지는 않는다. 대신에, 이들은 설명 된 주제와 관련된 양태와 일치하는 일부 예일 뿐이다. 비록 몇몇 변형이 본 명세서에서 상세하게 설명되었지만, 다른 변형 또는 추가가 가능하다. 특히, 본 명세서에 개시된 것들에 추가하여 추가의 특징 및 / 또는 변형이 제공 될 수있다. 예를 들어, 위에서 설명 된 구현들은 개시된 특징들의 다양한 조합 및 하위 조합 및 / 또는 본 명세서에 개시된 것들에 대한 하나 이상의 특징의 조합 및 하위 조합에 관한 것일 수있다. 또한, 첨부 도면에 도시되고 /되거나 본원에 기술 된 논리 흐름은 바람직한 결과를 달성하기 위해 도시 된 특정 순서, 또는 순차적 순서를 반드시 요구할 필요는 없다. 다음의 청구항들의 범위는 다른 구현들 또는 실시 예들을 포함 할 수있다.

---------------------------------------------------------------------------------------

주장하는 것은 :

1 다음을 포함하는 방법 :

객체 그래프의 적어도 하나의 객체의 적어도 하나의 판독 동작을 개시하는 단계로서, 상기 객체 그래프는 루트 객체를 포함하는 인 메모리 데이터베이스 내에 저장된 객체의 계층을 특징으로하며, 상기 노드의 적어도 일부는 연관된 객체를 가리키는 에지의 수-여기서 그래프 내의 객체에 대해 호환 가능한 변경이 이루어질 수 있고 그래프 내의 객체에 대해 호환되지 않는 변경이 이루어질 수 있음-;

각각의 대응하는 판독 동작에 대해 루트 객체의 기준 카운트를 증가시키는 단계; 루트 객체에 대한 기준 카운트는 루트 객체를 가리키는 앵커 포인터를 초기에 반영하는 단계;

하나 이상의 객체를 변경하는 단계;

이러한 호환되지 않는 변경을 수용하기 위해 적어도 하나의 변경된 객체에 대해 호환되지 않는 변경이 이루어진 경우, 루트 객체, 적어도 하나의 변경된 객체 및 객체의 계층 구조 내에있는 중간에있는 객체 복제 새로 복제 된 객체와 하나 이상의 변경된 객체의 이전 버전에서 원래 연결된 객체 사이의 가장자리;

앵커 포인터를 복제 된 루트 객체에 링크하는 단계;

루트 오브젝트에 대한 참조 카운트가 0 일 때 루트 오브젝트를 삭제하는 단계;

루트 객체에서 시작하여 적어도 하나의 변경된 객체에서 끝나는 객체 그래프를 순회하여 그러한 객체에 대한 임의의 에지를 제거하는 단계;

대응하는 기준 카운트가 0이 될 때 각 객체를 드롭하는 단계;

여기서 :

데이터베이스는 단조 증가하는 행 식별자를 갖는 행에 데이터를 저장하고;

데이터베이스는 트랜잭션 격리 및 일관된 읽기를 위해 다중 버전 동시성 제어를 사용합니다. 새 버전의 레코드가 작성 될 때 새 버전의 레코드에 새 행 ID가 제공되고 이전 버전의 레코드가 병렬 리더를 위해 유지됩니다. 커밋 후 가비지 수집 프로세스 중에 청소합니다.

2 제 1 항에있어서, 상기 기준 카운트는 둘 이상의 CPU에 걸쳐 스트라이핑되는 방법.

3. 비 일시적 컴퓨터 프로그램 제품 저장

적어도 하나의 프로그램 가능 프로세서에 의해 실행될 때, 적어도 하나의 프로그램 가능 프로세서가 다음을 포함하는 동작을 수행하게하는 명령어

인 메모리 데이터베이스에 저장된 테이블의 적어도 하나의 페이지의 판독 동작을 개시하는 단계-상기 판독 동작은 적어도 하나의 페이지에 액세스하기 위해 제 1 테이블 헤더 오브젝트를 가리키는 앵커 오브젝트를 사용함-상기 제 1 테이블 헤더 오브젝트는 링크를 포함 함) 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함하는 제 1 링크 된 오브젝트;

상기 판독 된 동작과 동시에 상기 링크 된 객체 및 상기 테이블 헤더 객체를 복제하는 단계-상기 복제 된 링크 된 객체는 상기 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함 함-;

앵커 된 객체가 복제 된 테이블 헤더 객체를 가리 키도록 설정하는 단계;

적어도 하나의 추가 페이지를 복제 된 링크 된 객체에 링크하는 단계; 및 판독 동작이 완료된 후 제 1 테이블 헤더 객체 및 제 1 링크 된 객체를 드롭하는 단계를 포함한다.

여기서 :

데이터베이스는 단조 증가하는 행 식별자를 갖는 행에 데이터를 저장하고;

데이터베이스는 트랜잭션 격리 및 일관된 읽기를 위해 다중 버전 동시성 제어를 사용합니다. 새 버전의 레코드가 작성 될 때 새 버전의 레코드에 새 행 ID가 제공되고 이전 버전의 레코드가 병렬 리더를 위해 유지됩니다. 커밋 후 가비지 수집 프로세스 중에 청소합니다.

4 제 3 항에있어서, 상기 제 1 링크 된 객체는 기준 카운트를 포함하는 버전 화 된 객체이며, 상기 기준 카운트는 상기 인 메모리 데이터베이스가 상기 판독 동작 동안 상기 제 1 테이블 헤더 객체 및 상기 제 1 링크 된 객체를 유지하도록 강제하는 컴퓨터 프로그램 제품.

5 제 4 항에있어서, 상기 기준 카운트는 둘 이상의 CPU에 걸쳐 스트라이핑되는 컴퓨터 프로그램 제품.

6 제 3 항에있어서, 상기 복제 된 테이블 헤더 객체는 초기에 상기 제 1 링크 된 객체에 연결되는 컴퓨터 프로그램 제품.

7 제 3 항에있어서, 상기 제 1 테이블 헤더 객체는 대응하는 페이지를 특징 짓는 관련된 메타 데이터를 갖는 컴퓨터 프로그램 제품.

8 제 3 항에있어서, 상기 복제 된 테이블 헤더 객체는 또한 상기 제 1 테이블 헤더 객체와 관련된 메타 데이터와 관련되는 컴퓨터 프로그램 제품.

9. 비 일시적 컴퓨터 프로그램 제품 저장

하나 이상의 프로그램 가능 프로세서에 의해 실행될 때, 하나 이상의 프로그램 가능 프로세서로 하여금 다음을 포함하는 동작을 수행하게하는 명령어;

객체 그래프의 적어도 하나의 객체의 적어도 하나의 판독 동작을 개시하는 단계로서, 상기 객체 그래프는 루트 객체를 포함하는 인 메모리 데이터베이스에 저장된 객체의 계층을 특징으로하며, 상기 노드의 적어도 일부는 연관된 객체를 가리키는 에지의 수-여기서 그래프 내의 객체에 대해 호환 가능한 변경이 이루어질 수 있고 그래프 내의 객체에 대해 호환되지 않는 변경이 이루어질 수 있음-;

각각의 대응하는 판독 동작에 대해, 대응하는 객체의 기준 카운트를, 루트 객체의 기준 카운트가 루트 객체를 가리키는 앵커 포인터를 초기에 반영하는 단계;

하나 이상의 객체를 변경하는 단계;

이러한 호환되지 않는 변경을 수용하기 위해 적어도 하나의 변경된 객체에 대해 호환되지 않는 변경이 이루어진 경우, 루트 객체, 적어도 하나의 변경된 객체 및 객체의 계층 구조 내에있는 중간에있는 객체 복제 새로 복제 된 객체와 하나 이상의 변경된 객체의 이전 버전에서 원래 연결된 객체 사이의 가장자리;

앵커 포인터를 복제 된 루트 객체에 링크하는 단계;

루트 오브젝트에 대한 참조 카운트가 0 일 때 루트 오브젝트를 삭제하는 단계;

루트 객체에서 시작하여 적어도 하나의 변경된 객체에서 끝나는 객체 그래프를 순회하여 그러한 객체에 대한 임의의 에지를 제거하는 단계; 과

대응하는 참조 카운트가 0이 될 때 각 객체를 드롭하는 단계;

여기서 :

데이터베이스는 단조 증가하는 행 식별자를 갖는 행에 데이터를 저장하고;

데이터베이스는 트랜잭션 격리 및 일관된 읽기를 위해 다중 버전 동시성 제어를 사용합니다. 새 버전의 레코드가 작성 될 때 새 버전의 레코드에 새 행 ID가 제공되고 이전 버전의 레코드가 병렬 리더를 위해 유지됩니다. 커밋 후 가비지 수집 프로세스 중에 청소합니다.

10 제 9 항에있어서, 상기 기준 카운트는 둘 이상의 CPU에 걸쳐 스트라이핑되는 컴퓨터 프로그램 제품.

11 제 9 항에있어서, 상기 판독 동작은 호환되지 않는 변경을 포함하는 컴퓨터 프로그램 제품.

12. 다음을 포함하는 시스템 :

적어도 하나의 프로그램 가능 프로세서, 및 적어도 하나의 프로그램 가능 프로세서에 의해 실행될 때,

인 메모리 데이터베이스에 저장된 테이블의 적어도 하나의 페이지의 판독 동작을 개시하는 단계-상기 판독 동작은 적어도 하나의 페이지에 액세스하기 위해 제 1 테이블 헤더 오브젝트를 가리키는 앵커 오브젝트를 사용함-상기 제 1 테이블 헤더 오브젝트는 링크를 포함 함) 적어도 하나의 페이지를 포함하는 복수의 페이지에 대한 핸들을 포함하는 제 1 링크 된 객체에;

상기 판독 된 동작과 동시에 상기 링크 된 객체 및 상기 테이블 헤더 객체를 복제하는 단계-상기 복제 된 링크 된 객체는 상기 적어도 하나의 페이지를 포함하는 복수의 5 페이지에 대한 핸들을 포함 함-;

앵커 된 객체가 복제 된 테이블 헤더 객체를 가리 키도록 설정하는 단계;

적어도 하나의 추가 페이지를 복제 된 링크 된 객체에 링크하는 단계; 과

판독 동작이 완료된 후 제 1 테이블 헤더 객체 및 제 1 링크 된 객체를 드롭하는 단계

여기서 :

데이터베이스는 단조 증가하는 행 식별자를 갖는 행에 데이터를 저장하고;

데이터베이스는 트랜잭션 격리 및 일관된 읽기를 위해 다중 버전 동시성 제어를 사용합니다. 새 버전의 레코드가 작성 될 때 새 버전의 레코드에 새 행 ID가 제공되고 이전 버전의 레코드가 병렬 리더를 위해 유지됩니다. 커밋 후 가비지 수집 프로세스 중에 청소합니다.

13 제 12 항에있어서, 상기 제 1 링크 된 객체는 기준 카운트를 포함하는 버전 화 된 객체이고, 상기 기준 카운트는 상기 인 메모리 데이터베이스가 상기 판독 동작 동안 상기 제 1 테이블 헤더 객체 및 상기 제 1 링크 된 객체를 유지하도록 강제하는 시스템.

14 제 13 항에있어서, 상기 기준 카운트는 둘 이상의 CPU에 걸쳐 스트라이핑되는 시스템.

15 제 12 항에있어서, 상기 복제 된 테이블 헤더 객체는 초기에 상기 제 1 링크 된 객체에 연결되는 시스템.

16 제 12 항에있어서, 상기 제 1 테이블 헤더 객체는 대응하는 페이지를 특징 짓는 관련된 메타 데이터를 갖는 시스템.

17 제 12 항에있어서, 상기 복제 된 테이블 헤더 객체는 또한 상기 제 1 테이블 헤더 객체와 관련된 메타 데이터와 관련되는 시스템.

18 시스템으로서,

적어도 하나의 프로그램 가능 프로세서; 과

적어도 하나의 프로그램 가능 프로세서에 의해 실행될 때, 다음을 포함하는 동작을 발생시키는 메모리 저장 명령;

객체 그래프의 적어도 하나의 객체의 적어도 하나의 판독 동작을 개시하는 단계로서, 상기 객체 그래프는 루트 객체를 포함하는 인 메모리 데이터베이스에 저장된 객체의 계층을 특징으로하며, 상기 노드의 적어도 일부는 연관된 객체를 가리키는 에지의 수-여기서 그래프 내의 객체에 대해 호환 가능한 변경이 이루어질 수 있고 그래프 내의 객체에 대해 호환되지 않는 변경이 이루어질 수 있음-;

각각의 대응하는 판독 동작에 대해, 대응하는 객체의 기준 카운트를, 루트 객체의 기준 카운트가 루트 객체를 가리키는 앵커 포인터를 초기에 반영하는 단계;

하나 이상의 객체를 변경하는 단계;

이러한 호환되지 않는 변경을 수용하기 위해 적어도 하나의 변경된 객체에 대해 호환되지 않는 변경이 이루어진 경우, 루트 객체, 적어도 하나의 변경된 객체 및 객체의 계층 구조 내에있는 중간에있는 객체 복제 새로 복제 된 객체와 하나 이상의 변경된 객체의 이전 버전에서 원래 연결된 객체 사이의 가장자리;

앵커 포인터를 복제 된 루트 객체에 링크하는 단계;

루트 오브젝트에 대한 참조 카운트가 0 일 때 루트 오브젝트를 삭제하는 단계;

루트 객체에서 시작하여 적어도 하나의 변경된 객체에서 끝나는 객체 그래프를 순회하여 그러한 객체에 대한 임의의 에지를 제거하는 단계; 과

대응하는 참조 카운트가 0이 될 때 각 객체를 드롭하는 단계;

여기서 :

데이터베이스는 단조 증가하는 행 식별자로 행에 데이터를 저장합니다.

데이터베이스는 트랜잭션 격리 및 일관된 읽기를 위해 다중 버전 동시성 제어를 사용합니다. 새 버전의 레코드가 작성 될 때 새 버전의 레코드에 새 행 ID가 제공되고 이전 버전의 레코드가 병렬 리더를 위해 유지됩니다. 커밋 후 가비지 수집 프로세스 중에 청소합니다.

19 제 18 항에있어서, 상기 기준 카운트는 둘 이상의 CPU에 걸쳐 스트라이핑되는 시스템.

20 제 18 항에있어서, 상기 판독 동작은 호환되지 않는 변경을 포함하는 시스템.