2020-09-25

【서지사항】

【서류명】 특허출원서

【참조번호】 0516

【출원구분】 특허출원

【출원인】

【명칭】 연세대학교 산학협력단

【특허고객번호】 2-2005-009509-9

【대리인】

【명칭】 특허법인 우인

【대리인번호】 9-2006-100082-1

【지정된변리사】 양승식, 지현수

【포괄위임등록번호】 2007-057789-6

【발명의 국문명칭】 쓰기 최적화 데이터베이스에서의 디스크 스캔 연산 GPU 오프로딩 방법

【발명의 영문명칭】 METHOD FOR OFFLOADING DISK SCAN DIRECTLY TO GPU IN WRITE-OPTIMIZED DATABASE SYSTEM

【발명자】

【성명】 박상현

【성명의 영문표기】 PARK, Sang Hyun

【주민등록번호】 670101-1XXXXXX

【우편번호】 08004

【주소】 서울특별시 양천구 오목로 300, 204동 3701호(목동, 현대하이페리온2)

【발명자】

【성명】 최원기

【성명의 영문표기】 CHOI, Won Gi

【주민등록번호】 910204-1XXXXXX

【우편번호】 07216

【주소】 서울특별시 영등포구 당산로42길 16, 507동 202호(당산동4가, 당산현대5차아파트)

【출원언어】 국어

【심사청구】 청구

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 1711103018

【과제번호】 IITP-2017-0-00477-004

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 정보통신기획평가원

【연구사업명】 SW컴퓨팅산업원천기술개발사업(SW스타랩)

【연구과제명】 (SW스타랩) IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

【기여율】 1/1

【과제수행기관명】 연세대학교 산학협력단

【연구기간】 2020.01.01 ~ 2020.12.31

【취지】 위와 같이 특허청장에게 제출합니다.

대리인 특허법인 우인 (서명 또는 인)

【수수료】

【출원료】	0	면	46,000 원
【가산출원료】	31	면	0 원
【우선권주장료】	0	건	0 원
【심사청구료】	5	항	363,000 원
【합계】	409,000 원		
【감면사유】	전담조직(50%감면)[1]		
【감면후 수수료】	204,500 원		

【수수료 자동납부번호】 05801106348

【발명의 설명】

【발명의 명칭】

쓰기 최적화 데이터베이스에서의 디스크 스캔 연산 GPU 오프로딩 방법 {METHOD FOR OFFLOADING DISK SCAN DIRECTLY TO GPU IN WRITE-OPTIMIZED DATABASE SYSTEM}

【기술분야】

【0001】본 발명이 속하는 기술 분야는 로그 구조 병합 트리 기반의 데이터베이스의 스캔 연산 방법에 관한 것이다.

【발명의 배경이 되는 기술】

【0002】이 부분에 기술된 내용은 단순히 본 실시예에 대한 배경 정보를 제공할 뿐 종래기술을 구성하는 것은 아니다.

【0003】키-값 기반의 데이터베이스는 센서 데이터, 소셜 네트워크 데이터 등과 같이 비정형 데이터를 다루는데 유용하다. 키-값 기반의 데이터베이스는 로그 구조 병합 트리(Log Structured Merge Tree)를 주로 사용한다.

【0004】로그 구조 병합 트리(Log Structured Merge Tree, LSM-Tree)는 연속적인 쓰기 연산을 수행하는 워크로드를 위해 설계되었다. LSM-트리 구조는 하나의 인메모리 데이터 구조와 여러 개의 블록(ex. 디스크 등)에 저장을 위한 이어쓰기(Append) 방식의 데이터 구조로 이루어져 있다.

【0005】LSM-트리는 비휘발성 메모리 익스프레스(NVMe) SSD(Solid-State Drive)를 비롯한 하드웨어 플랫폼의 높은 대역폭을 유도하도록 설계되어 다양한 데이터베이스 관리 시스템(DBMS)의 스토리지 엔진에 적용된다. 쓰기 연산에 최적화된 구조체인 LSM-트리는 우수한 쓰기 및 공간 효율성으로 인하여 데이터베이스의 스토리지 엔진으로 널리 활용되고 있다.

【0006】LSM-트리는 키-값 데이터베이스에서 빈번히 발생하는 삽입 및 수정을 효율적으로 수행한다. 데이터를 우선 로그 형식으로 저장하고, 로그 상의 데이터 정렬, 수정 작업의 처리 등의 병합을 미루는 쓰기 친숙형 구조(Write Friendly Structure)이다.

【0007】LSM-트리 구조의 구조적 특성 때문에 데이터베이스가 분석 쿼리를 처리할 때마다 중앙 처리 장치(CPU)와 관련된 읽기 증폭이 발생한다. 이 증폭은 대규모 데이터베이스에서 상당한 영향을 준다. 분석 질의를 처리하는데 필요한 테이블 스캔 연산은 데이터 레코드가 LSM-트리의 각 레벨에 분산되어 있어 이를 병합하고 정렬하여 반복적으로 질의 엔진에 반환하는 작업과 해당 레코드 내부 열 값과 필터 조건이 만족하는지 비교하는 연산으로 인하여 CPU 리소스를 과도하게 사용하는 문제가 있다. 이는 NVMe(non-volatile memory express)와 같은 고성능 디바이스의 밴드폭을 충분히 사용하는 것을 방해하며 전체적 질의 처리 성능에 영향을 준다.

【선행기술문헌】

【특허문헌】

【0008】**(특허문헌 0001)** 한국등록특허공보 제10-1736406호 (2017.05.29.)

【발명의 내용】

【해결하고자 하는 과제】

【0009】본 발명의 실시예들은 GPU를 활용하여 LSM-트리의 파일을 읽고 필터 조건 비교 연산을 병렬 처리하며, 질의 처리와 디스크 스캔을 파이프라이닝하고, 직접 메모리 접근(DMA)을 활용하여 LSM-트리 내 파일을 GPU 메모리에 적재하고 GPU 스레드가 파일에 접근하여 필터 연산을 수행하고 결과를 반환하여, 스캔 성능을 향상시키는 데 발명의 주된 목적이 있다.

【0010】본 발명의 명시되지 않은 또 다른 목적들은 하기의 상세한 설명 및 그 효과로부터 용이하게 추론할 수 있는 범위 내에서 추가적으로 고려될 수 있다.

【과제의 해결 수단】

【0011】본 실시예의 일 측면에 의하면, 질의 엔진으로부터 테이블 조건을 수신하는 단계, 데이터베이스 엔진으로부터 데이터 블록에 관한 정보를 수신하는 단계, 및 그래픽 처리 유닛의 커널에 의해 상기 테이블 조건에 따라 상기 데이터 블록을 처리하고 결과를 출력하는 단계를 포함하는 데이터베이스의 스캔 연산 방법을 제공한다.

【0012】상기 데이터베이스 엔진은 로그 구조 병합 트리(Log Structured Merge Tree, LSM-Tree)를 이용하는 키-값 데이터베이스에 해당하며, 상기 로그 구조 병합 트리는 데이터를 우선 로그 형식으로 저장하고, 로그 상의 데이터 정렬 및 수정 작업의 병합을 미루는 쓰기 친숙형 구조(Write Friendly Structure)일 수 있다.

【0013】상기 데이터 블록을 처리하고 결과를 출력하는 단계는, 상기 그래픽 처리 유닛의 커널을 이용하여 질의 처리와 테이블 스캔을 파이프라이닝할 수 있다.

【0014】상기 데이터 블록을 처리하고 결과를 출력하는 단계는, 직접 메모리 접근(Direct Memory Access)을 이용하여 상기 로그 구조 병합 트리의 파일을 상기 그래픽 처리 유닛의 메모리에 적재할 수 있다.

【0015】상기 데이터 블록을 처리하고 결과를 출력하는 단계에서, 상기 그래픽 처리 유닛의 커널은 상기 그래픽 처리 유닛의 스레드가 상기 데이터베이스 엔진의 메모리 영역에 접근하기 위한 메타 데이터를 상기 데이터베이스 엔진에 요청할 수 있다.

【0016】상기 데이터 블록을 처리하고 결과를 출력하는 단계에서, 상기 그래픽 처리 유닛의 스레드가 상기 그래픽 처리 유닛의 메모리에 적재된 파일에 접근하여 필터 연산을 수행하고 상기 질의 엔진으로 결과를 반환할 수 있다.

【0017】본 실시예의 다른 측면에 의하면, 데이터베이스의 스캔 연산 장치에 있어서, 병렬 연산을 수행하는 그래픽 처리 유닛, 상기 그래픽 처리 유닛으로 테이

블 조건을 전송하는 질의 엔진, 및 상기 그래픽 처리 유닛으로 데이터 블록에 관한 정보를 전송하는 데이터베이스 엔진을 포함하며, 상기 그래픽 처리 유닛의 커널에 의해 상기 테이블 조건에 따라 상기 데이터 블록을 처리하고 결과를 출력하는 것을 특징으로 하는 데이터베이스의 스캔 연산 장치를 제공한다.

【0018】상기 데이터베이스 엔진은 로그 구조 병합 트리(Log Structured Merge Tree, LSM-Tree)를 이용하는 키-값 데이터베이스에 해당하며, 상기 로그 구조 병합 트리는 데이터를 우선 로그 형식으로 저장하고, 로그 상의 데이터 정렬 및 수정 작업의 병합을 미루는 쓰기 친숙형 구조(Write Friendly Structure)일 수 있다.

【0019】상기 데이터베이스의 스캔 연산 장치는 상기 그래픽 처리 유닛의 커널을 이용하여 질의 처리와 테이블 스캔을 파이프라이닝할 수 있다.

【0020】상기 데이터베이스의 스캔 연산 장치는 직접 메모리 접근(Direct Memory Access)을 이용하여 상기 로그 구조 병합 트리의 파일을 상기 그래픽 처리 유닛의 메모리에 적재할 수 있다.

【0021】상기 그래픽 처리 유닛의 커널은 상기 그래픽 처리 유닛의 스레드가 상기 데이터베이스 엔진의 메모리 영역에 접근하기 위한 메타 데이터를 상기 데이터베이스 엔진에 요청할 수 있다.

【0022】상기 그래픽 처리 유닛의 스레드가 상기 그래픽 처리 유닛의 메모리에 적재된 파일에 접근하여 필터 연산을 수행하고 상기 질의 엔진으로 결과를 반환

할 수 있다.

【발명의 효과】

【0023】 이상에서 설명한 바와 같이 본 발명의 실시예들에 의하면, GPU를 활용하여 LSM-트리의 파일을 읽고 필터 조건 비교 연산을 병렬 처리하며, 질의 처리와 디스크 스캔을 파이프라이닝하고, 직접 메모리 접근(DMA)을 활용하여 LSM-트리 내 파일을 GPU 메모리에 적재하고 GPU 스레드가 파일에 접근하여 필터 연산을 수행하고 결과를 반환하여, 스캔 성능을 향상시킬 수 있는 효과가 있다.

【0024】 여기에서 명시적으로 언급되지 않은 효과라 하더라도, 본 발명의 기술적 특징에 의해 기대되는 이하의 명세서에서 기재된 효과 및 그 잠정적인 효과는 본 발명의 명세서에 기재된 것과 같이 취급된다.

【도면의 간단한 설명】

【0025】 도 1은 LST-트리에서 검색 프로세스를 예시한 도면이다.

도 2는 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치를 예시한 블록도이다.

도 3은 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 질의 검색 동작을 예시한 도면이다.

도 4는 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 GPU 커널 동작을 예시한 도면이다.

도 5는 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 파이

프라이닝 동작을 예시한 도면이다.

도 6은 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 DMA 동작을 예시한 도면이다.

도 7은 본 발명의 다른 실시예에 따른 데이터베이스의 스캔 연산 방법을 예시한 흐름도이다.

【발명을 실시하기 위한 구체적인 내용】

【0026】 이하, 본 발명을 설명함에 있어서 관련된 공지기능에 대하여 이 분야의 기술자에게 자명한 사항으로서 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략하고, 본 발명의 일부 실시예들을 예시적인 도면을 통해 상세하게 설명한다.

【0027】 도 1은 LST-트리에서 검색 프로세스를 예시한 도면이다.

【0028】 LSM-트리는 삽입 연산이 수행되면 먼저 메모리 영역에 데이터를 저장한다. 메모리의 일정 용량까지 데이터가 쌓이면 메모리의 내용을 디스크로 플러시(Flush)를 수행한다. 플러시되는 데이터는 디스크에 저장되어 있던 기존 데이터와 병합 정렬을 하여 기록된다. 디스크 영역의 각 레벨이 임계치를 넘으면 병합 정렬을 실행하여 하위 레벨을 생성한다.

【0029】 LSM-트리 기반의 데이터베이스는 키-값 형태로 데이터를 저장한다. LSM-트리 기반의 데이터베이스에 데이터의 삽입 연산 요청이 들어오면 데이터를 메모리에 기록하기 전에 우선적으로 로그 파일에 로그를 기록한다. 로그를 기록한 다

음 메모리 영역에 있는 멤테이블(Memtable)에 데이터를 저장한다. 쓰기 요청이 계속되어 멤테이블(Memtable)에 데이터가 일정 용량까지 기록되면, 멤테이블(Memtable)은 변경이 불가능한 불변 멤테이블(Immutable Memtable, Read-Only Memtable)로 변경된다. 불변 멤테이블이 가득 차게 되면 블록(디스크) 영역으로 플러시가 발생한다.

【0030】 플러시 동작을 수행하면, 멤테이블의 파일은 키 순서에 따라 정렬되어 SST(Storted String Table) 파일로 변경된다. SST 파일은 복수의 블록을 갖는다. 블록의 예시로는 데이터를 저장하는 데이터 블록(Data Block), 데이터 블록의 위치를 인덱싱하는 인덱스 블록(Index Block), 인덱스 블록의 위치를 처리하는 푸터 블록(Footer Block) 등이 있다.

【0031】 SST 파일은 디스크 영역에서 컴팩션(Compaction)을 통해 업데이트된다. 한 번 생성된 SST 파일은 사라지지 않을 수 있다. 하위 레벨에 상주하는 SST 파일일수록 상위 레벨의 SST 파일보다 오래된 데이터가 위치할 수 있다.

【0032】 LSM-트리 기반 데이터베이스는 아키텍처에 의한 읽기 증폭과 대역폭이 높은 저장 장치의 제한적인 사용으로 인해 분석 질의를 처리할 때 취약성에 직면한다. 분석 질의를 사용하면 동일하거나 다른 데이터 소스의 여러 질의의 데이터를 하나의 결과 집합으로 결합할 수 있다.

【0033】 본 실시예에 따른 데이터베이스의 스캔 연산 장치는 트랜잭션 및 분석 워크로드를 유연하게 처리하기 위해 NVMe SSD와 GPU 장치를 최대한 활용하여 검색 성능을 향상시킨다. NVMe SSD는 다중 GB/s I/O 속도를 제공하지만 GPU 프로세싱

의 이점을 제한하는 데이터 전송 오버헤드를 해결해야 한다. 본 실시예에 따른 데이터베이스의 스캔 연산 장치는 필터링 조건절(Predicate) 푸시다운을 통해 스캔 작업을 GPU로 오프로드하고 DMA(Direct Memory Access)가 있는 기기 간 데이터 전송으로 인한 병목 현상을 해결한다.

【0034】 도 2는 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치를 예시한 블록도이다.

【0035】 데이터베이스의 스캔 연산 장치(10)는 질의 엔진(100), 데이터베이스 엔진(200), 및 그래픽 처리 유닛(GPU, 300)을 포함한다. 데이터베이스의 스캔 연산 장치(10)는 도 2에서 예시적으로 도시한 다양한 구성요소들 중에서 일부 구성요소를 생략하거나 다른 구성요소를 추가로 포함할 수 있다.

【0036】 GPU 매니저 모듈은 GPU 자원을 관리하고 데이터베이스 파일의 데이터를 GPU 메모리로 전송하여 커널 기능을 실행하도록 구현된다.

【0037】 데이터베이스의 스캔 연산 장치(10)는 필터링 조건절 푸시다운(Filtering Predicate Pushdown, FPP) 접근 방식을 구현하여 LSM-트리 기반 스토리지 엔진의 테이블 스캔 성능을 향상시킨다. 이러한 접근 방식은 필요한 값을 레코드의 물리적 형식에서 해독하고 데이터를 조건절의 피벗 값과 비교하는 계산 오버헤드를 오프로드한다. GPU는 풍부한 가용성의 컴퓨팅 장치에 오버헤드를 분배한다.

【0038】 도 3은 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 질의 검색 동작을 예시한 도면이다.

【0039】도 3을 참조하면, 질의 엔진의 질의 파서 및 플래너 모듈은 필터링 조건절에 대한 정보를 얻고 FFP 접근방식을 통해 검색할 대상 테이블을 결정한다.

【0040】데이터베이스 엔진은 테이블의 기본 인덱스에 대한 고유 ID와 파일의 키 범위를 비교하여 LSM-트리를 검색하고 대상 테이블의 레코드가 포함된 SST 파일을 수집한다.

【0041】파일의 데이터 블록은 디스크에서 시스템 메모리로 읽히고, 데이터 블록의 레코드와 레코드 인덱스는 GPU 메모리에 복제될 배열 형태로 구성된다. 레코드 인덱스는 블록에서 레코드의 오프셋을 나타낸다.

【0042】GPU 스레드는 레코드 인덱스를 참조하여 레코드의 주소에 접근한다. 인덱스는 레코드와 관련하여 드문드문 구성되므로, 단일 스레드는 여러 레코드를 처리할 수 있다. 데이터 파일을 일괄적으로 읽고 GPU 자원을 활용해 테이블에 레코드가 들어 있는지 검증하고 조건에 따라 레코드를 평가하는 작업을 처리한다. 필터링된 결과는 정렬된 순서대로 질의 엔진으로 반환된다.

【0043】도 4는 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 GPU 커널 동작을 예시한 도면이다.

【0044】도 4를 참조하면, GPU 매니저 모듈은 레코드와 레코드 인덱스 배열을 GPU 메모리로 전송한다.

【0045】GPU 매니저 모듈은 CUDA 스트림을 활용하여 데이터 전송과 커널 실행을 오버랩하여 대량의 데이터를 처리하는 데 상당한 오버헤드를 완화한다. GPU

매니저 모듈은 레코드 데이터의 내용과 레코드 인덱스를 지정된 스트림에 분배한다. 스트림의 데이터와 인덱스도 GPU 블록 크기 단위로 나눈다. GPU 블록 크기는 동일한 GPU 리소스에서 작동하는 실행 단위를 나타낸다.

【0046】스레드는 레코드 인덱스가 지정한 레코드 배열의 영역을 검색하는 한편 레코드를 디코딩하고 필터 조건을 만족하는지 여부를 점검한다. 조건이 충족되면 레코드의 키와 포인터 및 값의 크기로 구성된 어레이가 CUDA 스트림을 활용하여 호스트 메모리에 비동기적으로 복제된다. GPU 매니저 모듈은 데이터 전송을 위해 저장된 호스트 메모리의 레코드 어레이와 중복된 어레이를 참조하여 논리적 레코드를 형성한다. 질의의 나머지 부분을 실행하도록 레코드의 논리적 포맷의 배치를 질의 엔진에 전파한다.

【0047】도 5는 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 파이프라이닝 동작을 예시한 도면이다.

【0048】GPU를 이용하는 애플리케이션에서 호스트 메모리와 GPU 장치 메모리 사이의 데이터 전송에 대한 오버헤드는 전체 시스템 성능에 영향을 준다. 대량의 데이터 전송은 애플리케이션의 다른 작업이 지연될 수 있도록 GPU 복사 엔진의 부담을 악화시킨다.

【0049】데이터베이스의 스캔 연산 장치는 다른 질의 실행과 테이블 스캔을 병렬로 수행하는데, 이러한 프로세스를 스캔 파이프라이닝이라고 한다.

【0050】데이터베이스의 스캔 연산 장치는 다음 레코드 배치에 대한 테이블 스캔을 수행하는 동안 연결(join) 또는 집계(aggregate) 작업을 실행할 수 있다. 스캔 파이프라이닝은 데이터 전송 지연 시간을 숨기고 전체 처리량을 개선할 수 있다.

【0051】도 6은 본 발명의 일 실시예에 따른 데이터베이스의 스캔 연산 장치의 DMA 동작을 예시한 도면이다.

【0052】파일의 내용은 GPU에서 처리하려면 두 번의 별도 데이터 복사 작업이 필요하다. 먼저 디스크에서 시스템 메모리로 내용을 읽는다. 다음 시스템 메모리의 데이터가 GPU 장치 메모리로 전송된다. 이러한 두 번의 데이터 전송은 GPU에 의한 병렬 처리의 이득을 상쇄할 정도로 심각한 지연을 제공한다. 전송 비용은 대부분의 데이터를 디스크에 저장하기 때문에 데이터베이스의 전체 처리량을 결정하는 데 지배적인 요인이다.

【0053】DMA 전송은 PCIe 디바이스에 지정할 메모리 공간을 공유하여 복사 경로를 단축한다. 이를 통해 NVMe SSD 디바이스의 파일을 호스트 CPU의 개입 없이 PCIe 컨트롤러에 의해 GPU 메모리에 비동기적으로 복사할 수 있다.

【0054】파싱(구문 분석)을 시작하고 연속적인 레이어에 조건절과 테이블 정보를 채운다. 다음 프로세스에 필요한 메모리 공간을 확보한다. 메타데이터 블록의 메모리의 작은 부분과 DMA 전송을 위한 고정된 공간만을 필요로 한다.

【0055】읽기 및 쓰기 작업을 포함하는 혼합 워크로드를 지원하기 위해 이러한 메타데이터 블록의 일관성을 유지한다. 새로운 SST 파일은 기존 파일을 수정하지 않고 외부 업데이트를 따름으로써 생성된다. 분리 레벨을 읽기 권한으로 제한하고 SST 파일의 버전을 추적한다. 메타데이터가 참조하는 SST 파일은 업데이트되지 않으며, 파일이 유효하지 않은 것으로 판단될 경우 검색 질의 실행 후 가비지 수집된다.

【0056】DMA 전송은 고정 메모리를 통해 파일을 비동기식으로 GPU 메모리에 매핑할 수 있게 한다. GPU 장치의 연산 코어는 장치 메모리에 매핑된 파일에 접근할 수 있다. 연산 코어가 파일을 논리적으로 해석하기 위해서 데이터 블록의 크기를 포함한 파일의 물리적 형식에 관한 메타데이터가 필요하다. 파일에서 관련 메타데이터를 읽어 GPU 메모리로 전송한다. 이러한 프로세스는 I/O 비용을 발생시키지만, 파일당 작은 블록만 필요하므로 I/O 비용은 무시할 수 있다.

【0057】사전 작업이 완료된 후 파일의 데이터 블록에 있는 레코드의 정확한 주소에 액세스하는 커널 함수를 실행한다. 파일의 데이터 블록은 GPU 블록이라고 하는 GPU 프로 프로세싱의 블록 단위에 대해 함께 그룹화된다. GPU 블록은 데이터 블록 집합을 포함하고 GPU 블록의 스레드는 데이터 블록에 분산되어 커널 기능을 수행한다. 데이터 블록에 배포된 각 스레드는 처리를 위해 여러 데이터 레코드를 포함하는 자체 장치 메모리를 할당한다. 이러한 메모리는 데이터 블록에 저장된 레코드 인덱스 항목을 기반으로 결정된다.

【0058】표 1을 참조하여 DMAFilterKernel이라 불리는 DMA 전송을 이용하기

위해 사용하는 커널 함수의 세부사항을 설명한다.

【0059】【표 1】

Algorithm 1 DMAFilterKernel

```
     Input
     - file_ptr_list : list of device memory address for a file
     - block_num_list : list of number of data blocks in a file
     - g_block_num_list : list of number of GPU blocks in a file
     - g_block_unit : number of data blocks in a GPU block
     - block_size_info : information for all data blocks' size
     - schema : information for table schema and predicates
     Output
     -d_results_idx : number of filtered records
     -d_results : key-value format for filtered records
 1   /** obtain GPU address of a file
 2   containing the space covered by a GPU block **/
 3   idx ← getFileIdx(blockIdx.x, g_block_num_list);
 4   file_ptr ← file_ptr_list[idx];
 5   /** obtain GPU address of a data block
 6   covered by the GPU block **/
 7   block_offset
 8     ←getBlockOffset
 9     (block_num_list, g_block_num_list, block_size_info);
10   block_ptr ← file_ptr + block_offset;
11   /** obtain the number of index in a data block
12   to allocate the index to a thread **/
13   num_restarts ← getNumIdx(block_ptr, block_size_info);
14   /** allocate the start offset and number of task
15   for a thread to process **/
16   start_offset, num_task
17     ← allotTask(threadIdx.x, num_restarts, g_block_unit);
18   /** execute filtering function **/
19   d_results_idx, d_results
20     ← DecodeNFilterOnSchema
21       (block_ptr, start_offset, num_task, schema);
22
```

【0060】이러한 함수는 DMA 전송에 의해 매핑된 파일의 주소 목록을 포함한 입력 매개변수를 필요로 한다. 커널 함수는 스레드가 접근할 메모리 영역을 지정하기 위해 여러 메타데이터를 필요로 한다. 메타데이터는 메타데이터에 포함된 데이터 블록 수를 나타내는 GPU 블록의 기본 단위가 포함된다. 파일에 할당된 데이터 블록 및 GPU 블록 수가 메타데이터에 포함되어 있다. 스키마 변수에는 조건절 정보와 열 값의 길이가 포함된다. 커널 함수는 스키마를 활용하여 데이터베이스 엔트리의 물리적 형식에서 열 값을 디코딩한다. 매개변수를 갖는 커널 함수는 결과값으로 필터링된 레코드의 키 값 배열을 쿼리 엔진으로 전송한다.

【0061】GPU 스레드는 파일의 GPU 주소, 파일의 데이터 블록 오프셋 및 데이터 블록의 레코드 오프셋을 참조하여 액세스할 메모리 영역을 식별한다. DMAFilterKernel은 모든 파일 주소 목록에서 스레드가 액세스해야 하는 주소를 선택하는 것으로 시작한다.

【0062】파일의 GPU 블록에 ID를 태그하면 함수는 GPU 블록 ID를 기준으로 파일 주소를 결정할 수 있다. 다음으로 데이터 및 GPU 블록 관련 메타데이터를 기반으로 파일 내 데이터 블록의 오프셋을 계산한다. 데이터 블록의 테일은 스레드의 작업을 할당하는 기준이 되는 레코드 인덱스 수를 저장한다. 함수는 블록 크기 정보와 데이터 블록의 오프셋을 보유하므로 레코드 인덱스 수를 저장하는 주소에 접근하여 해당 값을 얻을 수 있다. 인덱스 수는 num_restarts로 표시된다. 스레드의 ID, num_restarts, GPU 블록의 기본 단위를 기반으로 함수는 각 스레드에 해당하는

메모리 영역을 할당한다. 데이터 블록의 레코드 인덱스는 스레드에 분산된다. 스레드의 영역은 데이터 블록에서 레코드의 시작 오프셋과 고려할 레코드 수를 나타내는 인덱스에 의해 결정된다. 마지막으로 장치 함수 DecodeNFilterOnSchema는 데이터 블록 주소, 시작 오프셋, 끝 오프셋 및 스키마를 포함하여 실행된다.

【0063】표 2를 참조하여 DecodeNFilterOnSchema의 세부사항을 설명한다.

【0064】 【표 2】

Algorithm 2 DecodeNFilterOnSchema

```
Input
- block_ptr : device memory address for a data block
- start_offset : thread's start offset in a data block
- num_task : number of records for a thread to process
- schema : information for table schema and predicates

Output
-d_results_idx : number of filtered records
-d_results : key-value format for filtered records
1  /** allocate the space range for a thread to process **/
2  subblock ← startPtr (block_ptr, start_offset);
3  limit ← endPtr(block_ptr, start_offset, num_task);
4  while subblock < limit do
5  |  /** decode key from the designated pointer **/
6  |  key_ptr, shared, non_shared, value_ptr, value_size
7  |     ← decodeEntry(subblock)
8  |  key ← makeKey(key_ptr, shared, non_shared)
9  |  /** check whether the table contains the record **/
10 |   if first4Byte(key) != schema.table_key then
11 |   |continue;
12 |   endif
13 |  /** decode values from record's value pointer **/
14 |  decoded_values ← convertRecord(schema, value_ptr);
15 |  /** compare values with predicate's pivots **/
16 |  bool match ← condValid(decoded_values, schema);
17 |  if match == true then
18 |  |idx ← atomicAdd(d_results_idx, 1);
19 |  |d_results[idx] ← (key, value_ptr, value_size);
20 |  endif
21 |  /** move to the next record's pointer **/
22 |  subblock = value_ptr + value_size;
23 endwhile
24
```

【0065】DecodeNFilterOnSchema는 하나의 스레드에 의해 실행된다. DMAFilterKernel은 각 스레드에서 전근할 메모리 영역을 지정한다. 데이터 블록의 데이터 블록 주소와 시작 오프셋, 처리할 재귀 횟수로 표시된다. 종료 오프셋은 처리할 레코드 수에 따라 계산할 수 있다. 함수는 각 스레드가 접근할 메모리 영역의 시작점과 끝점을 지정하는데, 이를 서브블록 및 한계치라고 한다. 하위 블록 및 한계치는 데이터 블록에 저장된 레코드 인덱스를 참고하여 계산할 수 있다.

【0066】스레드는 메모리를 서브블록에서 한계치로 이동시킨다. 함수는 시작점에서 순차적으로 레코드 입력을 디코딩한다. 각 레코드 엔트리는 해당 키와 값에 대응하는 데이터와 크기의 순서로 구성된다.

【0067】데이터베이스 엔진은 예비 부분 저장을 방지하는 접두 키 인코딩으로 키를 저장하기 때문에 스레드는 검색하는 동안 추적을 통해 새로운 부분과 공유된 부분을 결합해 키를 구성한다. 스레드는 기본 키 인덱스의 고유 ID를 나타내는 생성된 키의 처음 소정의 바이트를 확인하여 디코딩할 레코드가 대상 테이블 내에 포함되는지 여부를 확인한다. 만약 레코드가 대상 테이블 내에 포함되도록 결정된다면, 필요한 열의 값은 데이터베이스 엔진의 값 포인터로부터 디코딩된다. 값의 길이가 포함된 스키마 정보를 참조하여 데이터베이스 엔진의 값 포인터를 사용한다. 열의 값을 디코딩한 후 스키마 정보에 존재하는 피벗 및 조건 연산자를 갖는 값을 결합하여 유효성 검사 함수를 실행한다. 레코드의 값이 조건에 유효한 것으로 결정되면 레코드의 키와 값 주소와 크기는 결과 배열에 일일이 삽입된다. 결과 배열을 참고하여 전체 레코드의 배열이 구성되며, 이후 질의 엔진으로 전달된

다. 스레드는 지정된 메모리 영역의 끝에 도달할 때까지 동일한 프로세스를 계속 실행한다.

【0068】도 7은 본 발명의 다른 실시예에 따른 데이터베이스의 스캔 연산 방법을 예시한 흐름도이다. 데이터베이스의 스캔 연산 방법은 데이터베이스의 스캔 연산 장치에 의해 수행될 수 있다.

【0069】단계 S21에서 질의 엔진으로부터 테이블 조건을 수신한다.

【0070】단계 S22에서 데이터베이스 엔진으로부터 데이터 블록에 관한 정보를 수신한다.

【0071】단계 S23에서 그래픽 처리 유닛의 커널에 의해 테이블 조건에 따라 데이터 블록을 처리하고 결과를 출력한다.

【0072】데이터베이스 엔진은 로그 구조 병합 트리(Log Structured Merge Tree, LSM-Tree)를 이용하는 키-값 데이터베이스에 해당하며, 로그 구조 병합 트리는 데이터를 우선 로그 형식으로 저장하고, 로그 상의 데이터 정렬 및 수정 작업의 병합을 미루는 쓰기 친숙형 구조(Write Friendly Structure)일 수 있다.

【0073】데이터 블록을 처리하고 결과를 출력하는 단계(S23)는 그래픽 처리 유닛의 커널을 이용하여 질의 처리와 테이블 스캔을 파이프라이닝할 수 있다.

【0074】데이터 블록을 처리하고 결과를 출력하는 단계(S23)는 직접 메모리 접근(Direct Memory Access)을 이용하여 로그 구조 병합 트리의 파일을 그래픽 처리 유닛의 메모리에 적재할 수 있다.

【0075】데이터 블록을 처리하고 결과를 출력하는 단계(S23)에서 그래픽 처리 유닛의 커널은 상기 그래픽 처리 유닛의 스레드가 상기 데이터베이스 엔진의 메모리 영역에 접근하기 위한 메타 데이터를 상기 데이터베이스 엔진에 요청할 수 있다.

【0076】상기 데이터 블록을 처리하고 결과를 출력하는 단계에서, 상기 그래픽 처리 유닛의 스레드가 상기 그래픽 처리 유닛의 메모리에 적재된 파일에 접근하여 필터 연산을 수행하고 상기 질의 엔진으로 결과를 반환할 수 있다.

【0077】본 발명은 기존 방식과 대비하여 가격 대비 성능비, 에너지 효율성, 분석 질의 처리 실험 결과, 고성능 디바이스의 밴드폭의 활용도 면에서 우수한 결과를 보인다.

【0078】데이터베이스에 포함된 구성요소들이 도 2에서는 분리되어 도시되어 있으나, 복수의 구성요소들은 상호 결합되어 적어도 하나의 모듈로 구현될 수 있다. 구성요소들은 장치 내부의 소프트웨어적인 모듈 또는 하드웨어적인 모듈을 연결하는 통신 경로에 연결되어 상호 간에 유기적으로 동작한다. 이러한 구성요소들은 하나 이상의 통신 버스 또는 신호선을 이용하여 통신한다.

【0079】데이터베이스는 하드웨어, 펌웨어, 소프트웨어 또는 이들의 조합에 의해 로직회로 내에서 구현될 수 있고, 범용 또는 특정 목적 컴퓨터를 이용하여 구현될 수도 있다. 장치는 고정배선형(Hardwired) 기기, 필드 프로그램 가능한 게이트 어레이(Field Programmable Gate Array, FPGA), 주문형 반도체(Application

Specific Integrated Circuit, ASIC) 등을 이용하여 구현될 수 있다. 또한, 장치는 하나 이상의 프로세서 및 컨트롤러를 포함한 시스템온칩(System on Chip, SoC)으로 구현될 수 있다.

【0080】 데이터베이스는 하드웨어적 요소가 마련된 컴퓨팅 디바이스에 소프트웨어, 하드웨어, 또는 이들의 조합하는 형태로 탑재될 수 있다. 컴퓨팅 디바이스는 각종 기기 또는 유무선 통신망과 통신을 수행하기 위한 통신 모뎀 등의 통신장치, 프로그램을 실행하기 위한 데이터를 저장하는 메모리, 프로그램을 실행하여 연산 및 명령하기 위한 마이크로프로세서 등을 전부 또는 일부 포함한 다양한 장치를 의미할 수 있다.

【0081】 도 7에서는 각각의 과정을 순차적으로 실행하는 것으로 기재하고 있으나 이는 예시적으로 설명한 것에 불과하고, 이 분야의 기술자라면 본 발명의 실시예의 본질적인 특성에서 벗어나지 않는 범위에서 도 7에 기재된 순서를 변경하여 실행하거나 또는 하나 이상의 과정을 병렬적으로 실행하거나 다른 과정을 추가하는 것으로 다양하게 수정 및 변형하여 적용 가능할 것이다.

【0082】 본 실시예들에 따른 동작은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능한 매체에 기록될 수 있다. 컴퓨터 판독 가능한 매체는 실행을 위해 프로세서에 명령어를 제공하는 데 참여한 임의의 매체를 나타낸다. 컴퓨터 판독 가능한 매체는 프로그램 명령, 데이터 파일, 데이터 구조 또는 이들의 조합을 포함할 수 있다. 예를 들면, 자기 매체, 광기록 매체, 메모리 등이 있을 수 있다. 컴퓨터 프로그램은 네트워크로 연결된 컴퓨터 시

스템 상에 분산되어 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수도 있다. 본 실시예를 구현하기 위한 기능적인(Functional) 프로그램, 코드, 및 코드 세그먼트들은 본 실시예가 속하는 기술분야의 프로그래머들에 의해 용이하게 추론될 수 있을 것이다.

【0083】본 실시예들은 본 실시예의 기술 사상을 설명하기 위한 것이고, 이러한 실시예에 의하여 본 실시예의 기술 사상의 범위가 한정되는 것은 아니다. 본 실시예의 보호 범위는 아래의 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 실시예의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

【청구범위】

【청구항 1】

데이터베이스의 스캔 연산 장치에 있어서,

병렬 연산을 수행하는 그래픽 처리 유닛;

상기 그래픽 처리 유닛으로 테이블 조건을 전송하는 질의 엔진; 및

상기 그래픽 처리 유닛으로 데이터 블록에 관한 정보를 전송하는 데이터베이스 엔진을 포함하며,

상기 그래픽 처리 유닛의 커널에 의해 상기 테이블 조건에 따라 상기 데이터 블록을 처리하고 결과를 출력하는 것을 특징으로 하는 데이터베이스의 스캔 연산 장치.

【청구항 2】

제1항에 있어서,

상기 데이터베이스 엔진은 로그 구조 병합 트리(Log Structured Merge Tree, LSM-Tree)를 이용하는 키-값 데이터베이스에 해당하며,

상기 로그 구조 병합 트리는 데이터를 우선 로그 형식으로 저장하고, 로그 상의 데이터 정렬 및 수정 작업의 병합을 미루는 쓰기 친숙형 구조(Write Friendly Structure)인 것을 특징으로 하는 데이터베이스의 스캔 연산 장치.

【청구항 3】

제2항에 있어서,

상기 그래픽 처리 유닛의 커널을 이용하여 질의 처리와 테이블 스캔을 파이프라이닝하는 것을 특징으로 하는 데이터베이스의 스캔 연산 장치.

【청구항 4】

제2항에 있어서,

직접 메모리 접근(Direct Memory Access)을 이용하여 상기 로그 구조 병합 트리의 파일을 상기 그래픽 처리 유닛의 메모리에 적재하는 것을 특징으로 하는 데이터베이스의 스캔 연산 장치.

【청구항 5】

제4항에 있어서,

상기 그래픽 처리 유닛의 커널은 상기 그래픽 처리 유닛의 스레드가 상기 데이터베이스 엔진의 메모리 영역에 접근하기 위한 메타 데이터를 상기 데이터베이스 엔진에 요청하고,

상기 그래픽 처리 유닛의 스레드가 상기 그래픽 처리 유닛의 메모리에 적재된 파일에 접근하여 필터 연산을 수행하고 상기 질의 엔진으로 결과를 반환하는 것을 특징으로 하는 데이터베이스의 스캔 연산 장치.

【요약서】

【요약】

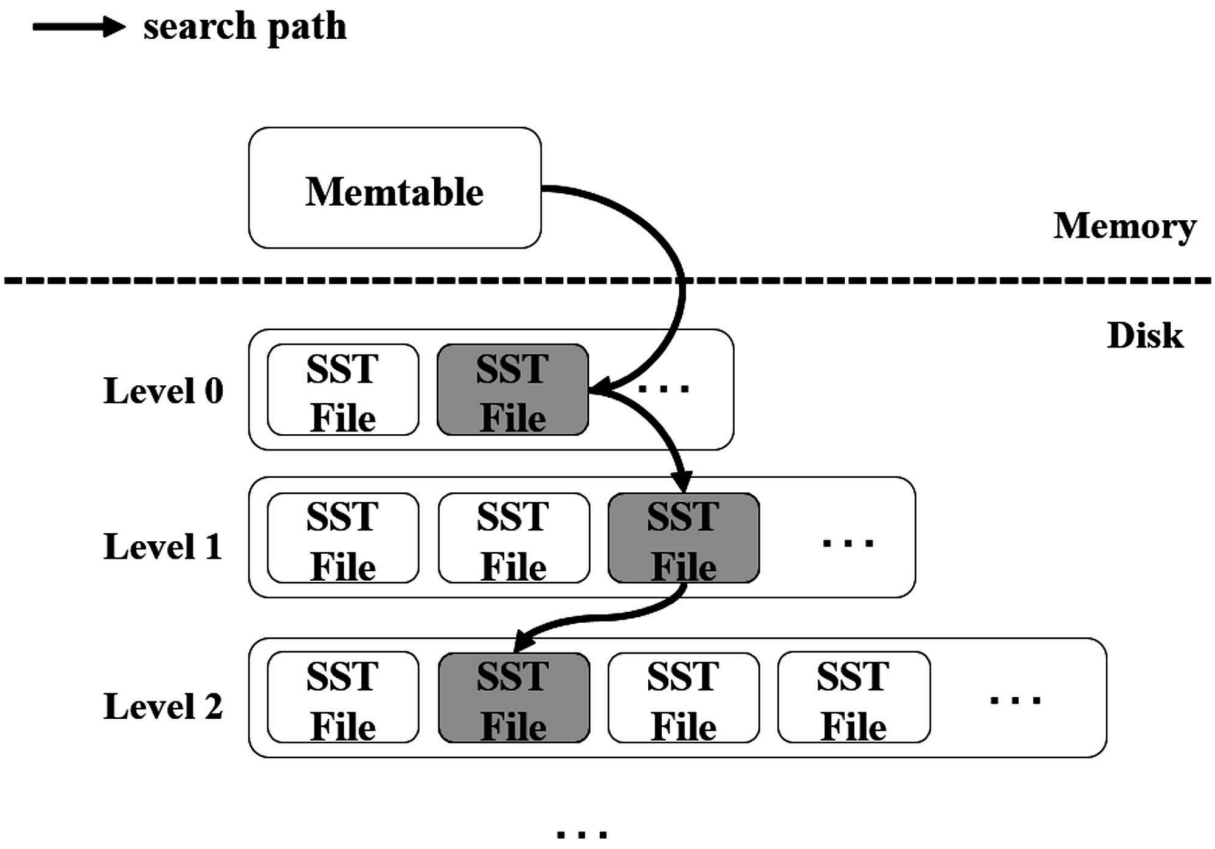
본 실시예들은 GPU를 활용하여 LSM-트리의 파일을 읽고 필터 조건 비교 연산을 병렬 처리하며, 질의 처리와 디스크 스캔을 파이프라이닝하고, 직접 메모리 접근(DMA)을 활용하여 LSM-트리 내 파일을 GPU 메모리에 적재하고 GPU 스레드가 파일에 접근하여 필터 연산을 수행하고 결과를 반환하여, 스캔 성능을 향상시키는 데이터베이스의 스캔 연산 방법 및 장치를 제공한다.

【대표도】

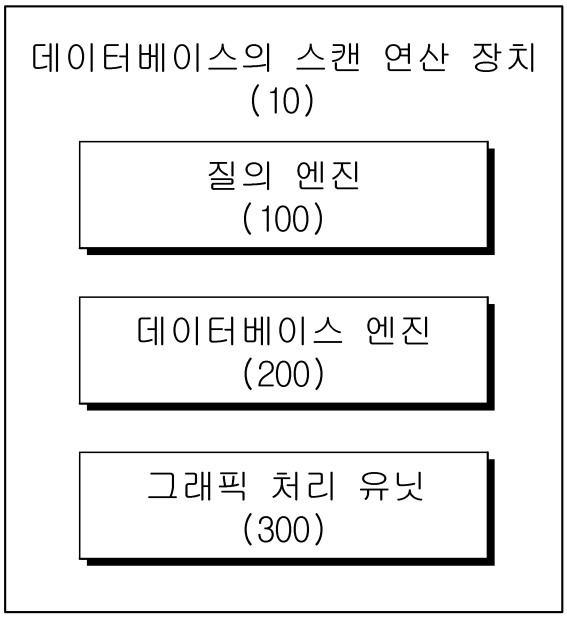
도 2

【도면】

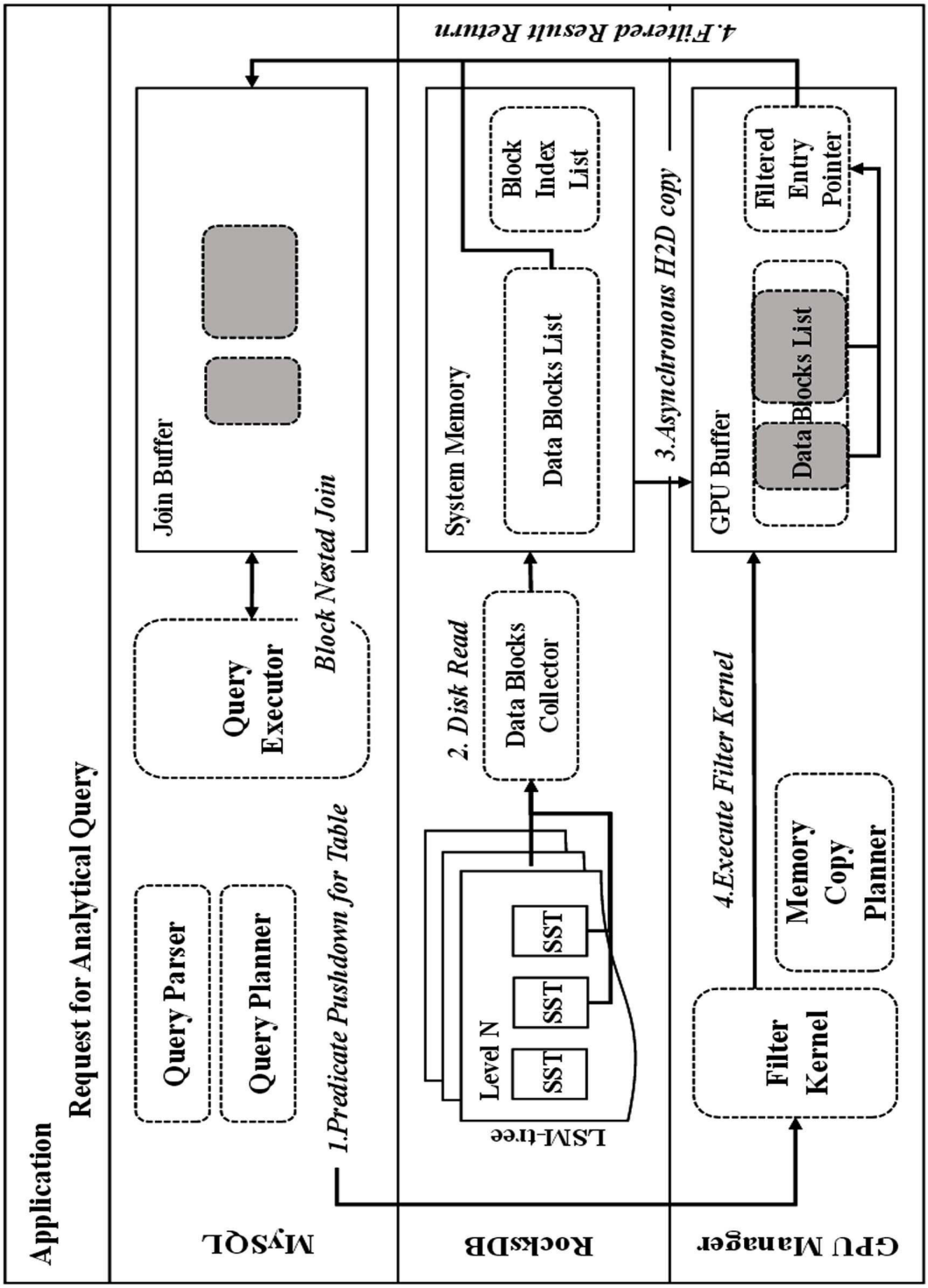
【도 1】


search path
Memtable
Memory
Disk
Level 0
SST File
SST File
Level 1
SST File
SST File
SST File
Level 2
SST File
SST File
SST File
SST File

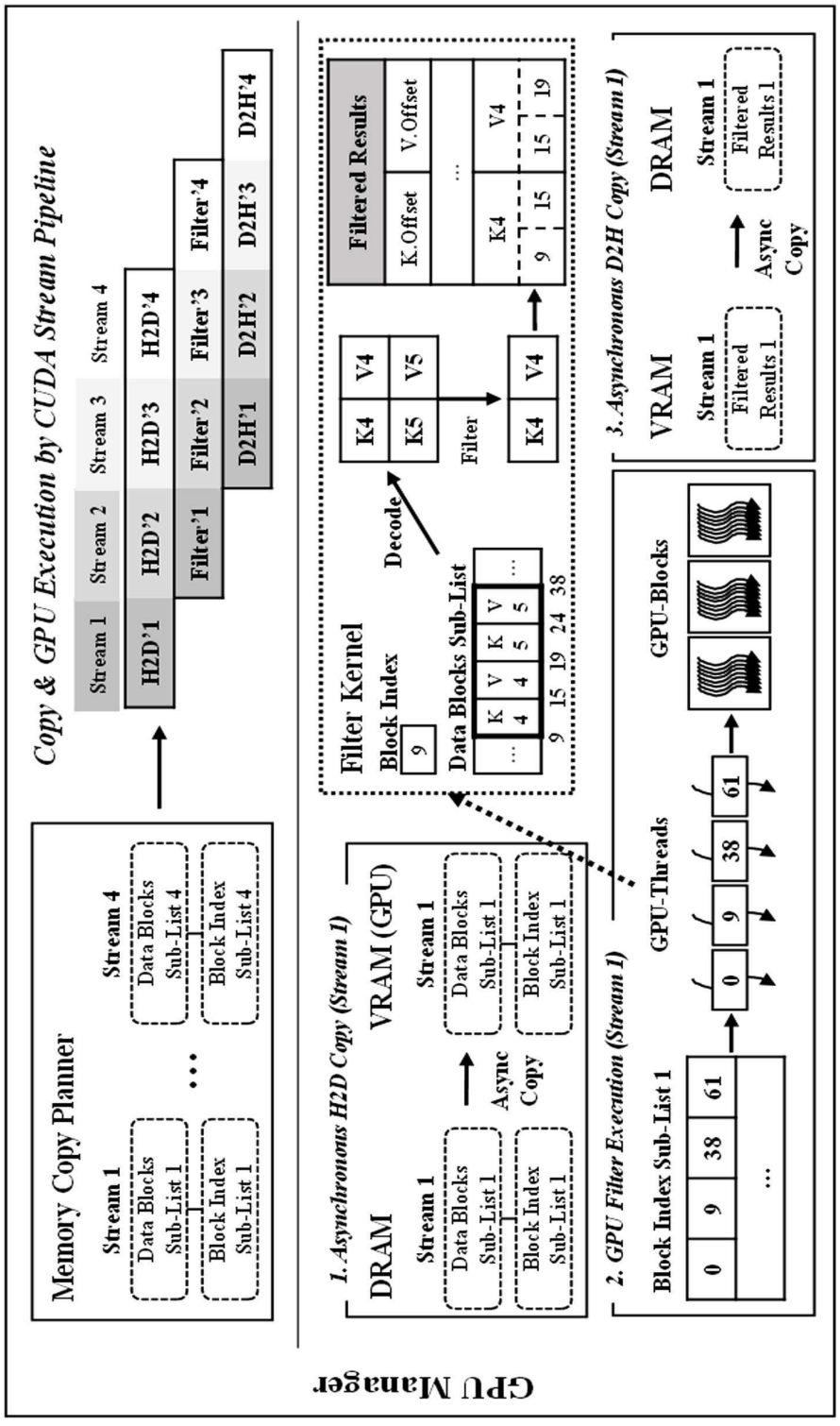
【도 2】



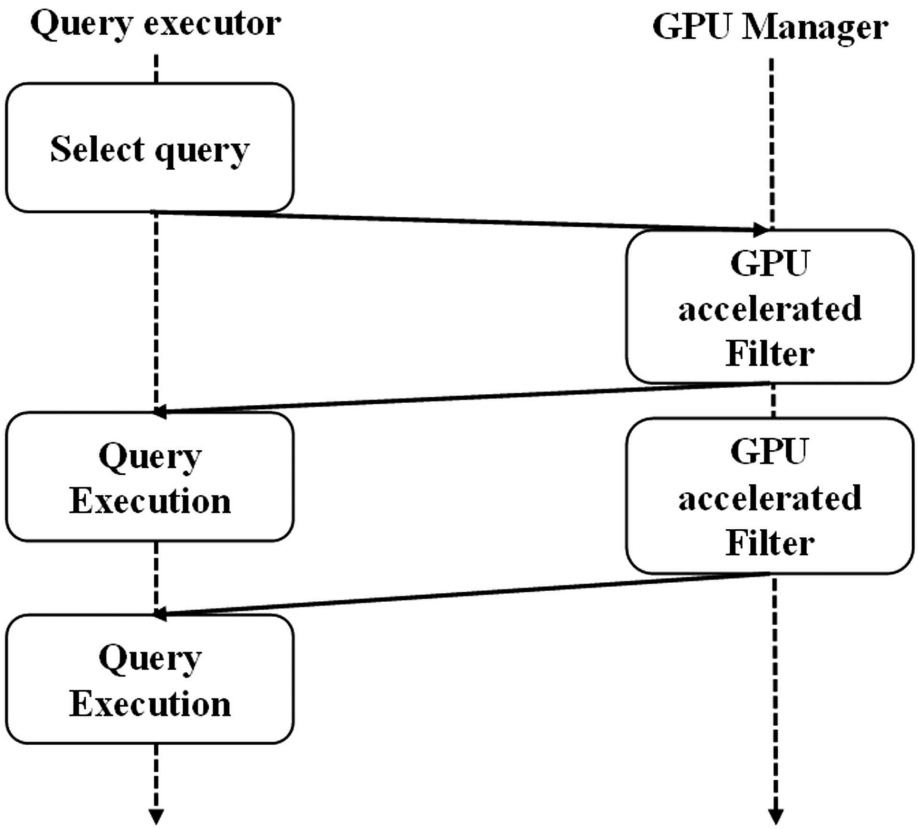
【도 3】


Application
Request for Analytical Query
MySQL
Query Parser
Query Planner
Query Executor
Block Nested Join
Join Buffer
1.Predicate Pushdown for Table
RocksDB
2. Disk Read
Data Blocks Collector
LSM-tree
Level N
SST
SST
SST
System Memory
Data Blocks List
Block Index List
4.Filtered Result Return
3.Asynchronous H2D copy
GPU Manager
4.Execute Filter Kernel
Filter Kernel
Memory Copy Planner
GPU Buffer
Data Blocks List
Filtered Entry Pointer

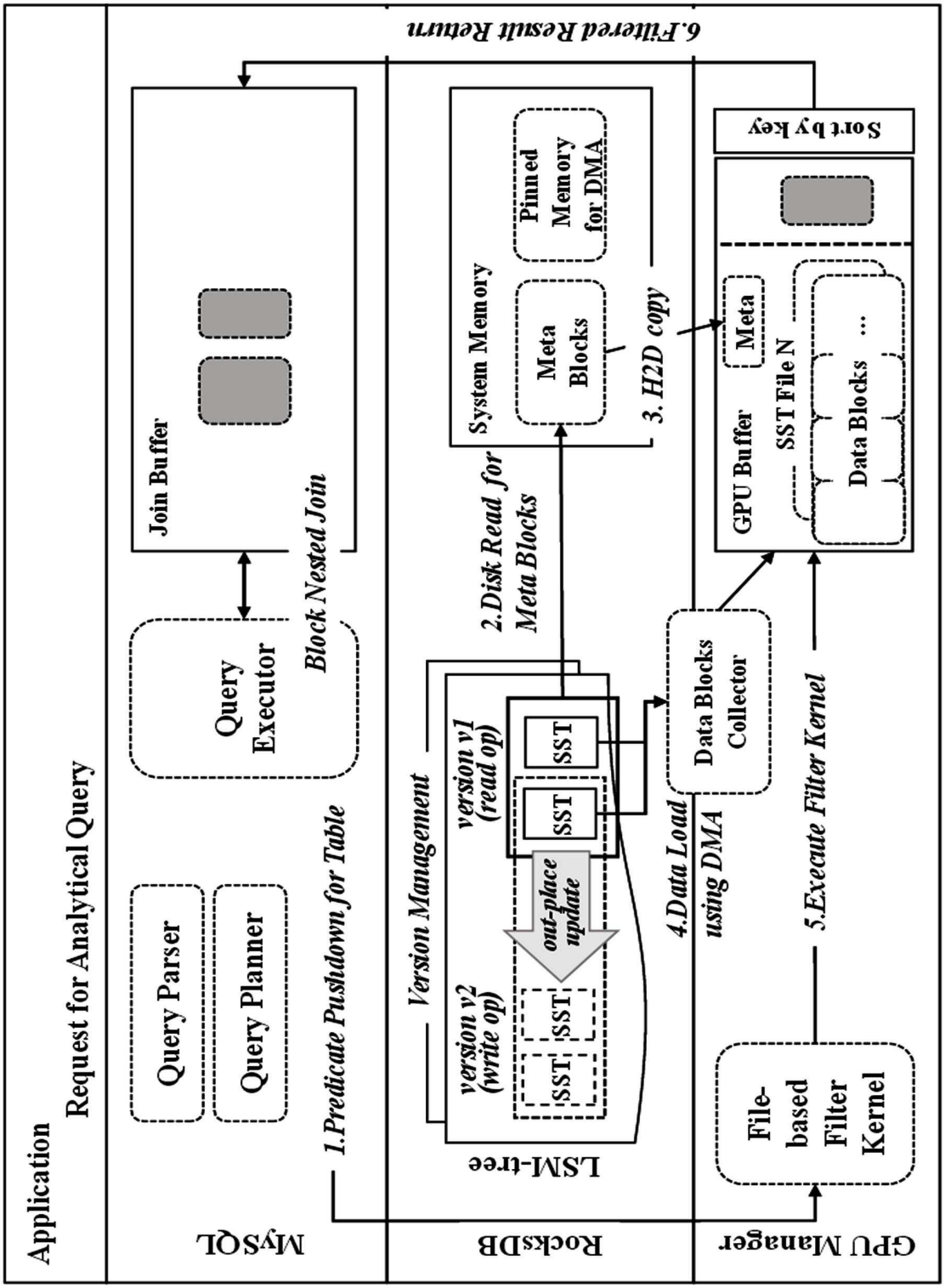
【도 4】


GPU Manager
Memory Copy Planner
Stream 1
Data Blocks Sub-List 1
Block Index Sub-List 1
Stream 4
Data Blocks Sub-List 4
Block Index Sub-List 4
Copy & GPU Execution by CUDA Stream Pipeline
Stream 1
Stream 2
Stream 3
Stream 4
H2D'1
H2D'2
H2D'3
H2D'4
Filter'1
Filter'2
Filter'3
Filter'4
D2H'1
D2H'2
D2H'3
D2H'4
1. Asynchronous H2D Copy (Stream 1)
DRAM
VRAM (GPU)
Async Copy
2. GPU Filter Execution (Stream 1)
Block Index Sub-List 1
GPU-Threads
GPU-Blocks
Filter Kernel
Block Index
Data Blocks Sub-List
Decode
Filter
Filtered Results
K.Offset
V.Offset
3. Asynchronous D2H Copy (Stream 1)
VRAM
DRAM
Filtered Results 1
Async Copy

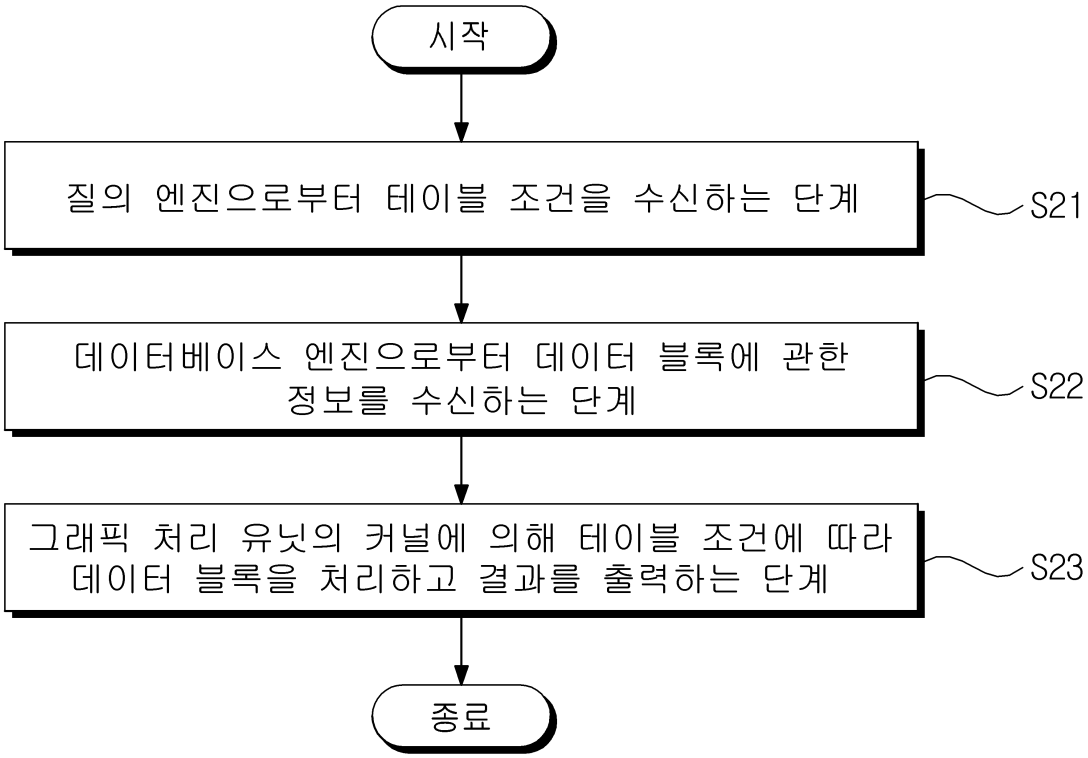
【도 5】


Query executor
GPU Manager
Select query
GPU accelerated Filter
Query Execution
GPU accelerated Filter
Query Execution

【도 6】


Application
Request for Analytical Query
MySQL
Query Parser
Query Planner
Query Executor
Block Nested Join
Join Buffer
1.Predicate Pushdown for Table
RocksDB
Version Management
version v1 (read op)
version v2 (write op)
SST
SST
SST
SST
out-place update
LSM-tree
2.Disk Read for Meta Blocks
System Memory
Meta Blocks
Pinned Memory for DMA
3. H2D copy
6.Filtered Result Return
GPU Manager
Data Blocks Collector
4.Data Load using DMA
5.Execute Filter Kernel
File-based Filter Kernel
Sort by key
GPU Buffer
Meta
SST File N
Data Blocks ...

【도 7】


시작
질의 엔진으로부터 테이블 조건을 수신하는 단계
S21
데이터베이스 엔진으로부터 데이터 블록에 관한 정보를 수신하는 단계
S22
그래픽 처리 유닛의 커널에 의해 테이블 조건에 따라 데이터 블록을 처리하고 결과를 출력하는 단계
S23
종료