【서지사항】

【서류명】 특허출원서

【참조번호】 0513

【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】 비휘발성 메모리를 이용한 로그 구조 병합 트리 기반의 데이터 베이스의 데이터 처리 방법

【발명의 영문명칭】 Method for Processing Data in Database Based on Log Structured Merge Tree Using Non-Volatile Memory

【발명자】

【성명】 박상현

【성명의 영문표기】 PARK, Sang Hyun

【주민등록번호】 670101-1XXXXXX

【우편번호】 08004

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

【발명자】

【성명】 이지환

【성명의 영문표기】 LEE, Ji Hwan

【주민등록번호】 950106-1XXXXXX

【우편번호】 03312

【주소】 서울특별시 은평구 통일로 1045, 102동 504호(진관동, 이룸채)

【발명자】

【성명】 최원기

【성명의 영문표기】 CHOI, Won Gi

【주민등록번호】 910204-1XXXXXX

【우편번호】 03724

【주소】 서울특별시 서대문구 연희로16길 17, 202호(연희동)

【발명자】

【성명】 성한승

【성명의 영문표기】 SUNG, Han Seung

【주민등록번호】 901103-1XXXXXX

【우편번호】 06116

【주소】 서울특별시 강남구 봉은사로21길 45, 3층(논현동)

【발명자】

【성명】 김도영

【성명의 영문표기】 KIM, Do Young

【주민등록번호】 930920-1XXXXXX

【우편번호】 07229

【주소】 서울특별시 영등포구 국회대로55길 28, 704호(당산동)

【출원언어】 국어

【심사청구】 청구

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 1711080997

【부처명】 과학기술정보통신부

【연구관리 전문기관】 정보통신기술진흥센터

【연구사업명】 정부-과학기술정보통신부-정보통신기술진흥센터(NIPA산하)-정보통신방송연구개발사업-SW컴퓨팅산업원천기술개발사업

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

【기여율】 1/1

【주관기관】 연세대학교 산학협력단

【연구기간】 2017.04.01 ~ 2024.12.31

【취지】 위와 같이 특허청장에게 제출합니다.

대리인 특허법인 우인 (서명 또는 인)

【수수료】

【출원료】 0 면 46,000 원

【가산출원료】	40	면	0 원
【우선권주장료】	0	건	0 원
【심사청구료】	16	항	847,000 원
【합계】	893,000 원		
【감면사유】	전담조직(50%감면)[1]		
【감면후 수수료】	446,500 원		
【수수료 자동납부번호】	05801106348		

【발명의 설명】

【발명의 명칭】

비휘발성 메모리를 이용한 로그 구조 병합 트리 기반의 데이터 베이스의 데이터 처리 방법 {Method for Processing Data in Database Based on Log Structured Merge Tree Using Non-Volatile Memory}

【기술분야】

【0001】 본 발명이 속하는 기술 분야는 비휘발성 메모리를 이용한 로그 구조 병합 트리 기반의 데이터 베이스 및 그 데이터 처리 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 데이터 베이스는 센서 데이터, 소셜 네트워크 데이터 등과 같이 비정형 데이터를 다루는데 유용하다. 키-값 기반의 데이터 베이스는 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-Tree 구조는 하나의 인메모리 데이터 구조와 여러 개의 블록(ex. 디스크 등)에 저장을 위한 이어쓰기(Append) 방식의 데이터 구조로 이루어져 있다.

【0005】LSM-Tree는 키-값 데이터 베이스에서 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. 하지만 나중에 발생하는 병합 동작은 쓰기 증폭을 발생시키며 시스템 성능과 저장장치의 수명에 영향을 준다.

【0006】LSM-Tree는 임의적인 순서로 데이터를 쓰지 않고 순차적으로 데이터를 쓴다. 데이터를 조회할 때 주어진 데이터가 트리 내의 어느 위치에 있는지 알 수 없어서 데이터를 찾기 위해서 상위 레벨부터 순차적으로 검색해야 한다. 디스크에 데이터가 없어도 모든 레벨의 모든 파일을 읽어야 한다.

【선행기술문헌】

【특허문헌】

【0007】**(특허문헌 0001)** 미국공개특허공보 US 2017-0344619 (2017.11.30.)

(특허문헌 0002) 한국공개특허공보 KR 10-2016-0121819 (2016.10.21.)

(특허문헌 0003) 미국공개특허공보 US 2018-0121121 (2018.05.03.)

【발명의 내용】

【해결하고자 하는 과제】

【0008】본 발명의 실시예들은 휘발성 메모리 및 비휘발성 메모리를 포함하는 데이터 베이스가 휘발성 메모리의 일정 용량을 초과한 데이터에 관하여 비휘발

성 메모리에 저장하고, 비휘발성 메모리의 리스트 구조 및 영속성 버퍼를 통해 플러시 동작 및 컴팩션 동작을 수행함으로써, 데이터 영속성을 유지하면서 쓰기 지연과 읽기 지연을 최소화하는 데 발명의 주된 목적이 있다.

【0009】 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】

【0010】 본 실시예의 일 측면에 의하면, 데이터 베이스의 데이터 처리 방법에 있어서, 상기 데이터 베이스의 휘발성 메모리에 데이터를 저장하는 단계, 및 상기 데이터 베이스의 비휘발성 메모리에 복수의 노드가 연결된 리스트 구조를 생성하고 상기 데이터를 상기 리스트 구조에 저장하는 방식으로 플러시 동작을 수행하는 단계를 포함하는 데이터 베이스의 데이터 처리 방법을 제공한다.

【0011】 상기 데이터 베이스는 키-값 형식으로 데이터를 저장하고, 상기 리스트 구조는 다수의 다음 포인터를 갖는 스킵 리스트일 수 있다.

【0012】 상기 데이터 베이스가 데이터 쓰기를 수행하지 않고, 새로운 리스트 구조를 생성하고 상기 새로운 리스트 구조가 기존 리스트 구조의 노드에 할당된 키-값을 포인팅하는 방식으로 컴팩션 동작을 수행하는 단계를 포함할 수 있다.

【0013】 상기 플러시 동작을 수행하는 단계는, 상기 비휘발성 메모리의 영속성 버퍼(Persistent Buffer)에 키-값을 순차적으로 복사하고, 상기 영속성 버퍼는 상기 노드에 할당된 키-값의 랜덤 접근을 방지하며, 상기 리스트 구조는 상기 리스

트 구조에 대응하는 영속성 버퍼의 오프셋을 포인팅할 수 있다.

【0014】 상기 컴팩션 동작을 수행하는 단계는, 상기 비휘발성 메모리에 새로운 리스트 구조를 생성하고 상기 새로운 리스트 구조가 이전 리스트 구조에 대응하는 영속성 버퍼의 오프셋을 포인팅할 수 있다.

【0015】 상기 데이터 베이스의 비휘발성 메모리에 저장된 데이터가 기 설정된 용량 범위를 초과하면, 단계화 정책(Tiering Policy)에 따라 상기 데이터 베이스의 블록 드라이브로 방출(Eviction)할 수 있다.

【0016】 상기 단계화 정책은, (i) 특정 레벨에 있는 데이터를 선택하여 상기 블록 드라이브에 저장하는 제1 단계화 정책, (ii) 데이터 접근이 오래된 데이터를 선택하여 상기 블록 드라이브에 저장하는 제2 단계화 정책, (iii) 모든 데이터를 상기 비휘발성 메모리에 저장하는 제3 단계화 정책, 또는 이들의 조합으로 설정될 수 있다.

【0017】 상기 데이터 베이스에서 시스템 오류가 발생하면, 상기 데이터 베이스의 비휘발성 메모리에 저장된 상기 리스트 구조가 포인팅하는 데이터를 조회한 결과를 통해 데이터를 순차적으로 복구하는 단계를 포함할 수 있다.

【0018】 본 실시예의 다른 측면에 의하면, 프로세서, 휘발성 메모리, 및 비휘발성 메모리를 포함하는 데이터 베이스에 있어서, 상기 휘발성 메모리에 데이터를 저장하고, 상기 비휘발성 메모리에 복수의 노드가 연결된 리스트 구조를 생성하고 상기 데이터를 상기 리스트 구조에 저장하는 방식으로 플러시 동작을 수행하는

것을 특징으로 하는 데이터 베이스를 제공한다.

【0019】본 실시예의 또 다른 측면에 의하면, 프로세서에 의해 실행 가능한 컴퓨터 프로그램 명령어들을 포함하는 비일시적(Non-Transitory) 컴퓨터 판독 가능한 매체에 기록되어 데이터 처리를 위한 컴퓨터 프로그램으로서, 상기 컴퓨터 프로그램 명령어들이 데이터 베이스의 적어도 하나의 프로세서에 의해 실행되는 경우에, 상기 데이터 베이스의 휘발성 메모리에 데이터를 저장하는 단계, 및 상기 데이터 베이스의 비휘발성 메모리에 복수의 노드가 연결된 리스트 구조를 생성하고 상기 데이터를 상기 리스트 구조에 저장하는 방식으로 플러시 동작을 수행하는 단계를 포함한 동작들을 수행하는 컴퓨터 프로그램을 제공한다.

【발명의 효과】

【0020】이상에서 설명한 바와 같이 본 발명의 실시예들에 의하면, 휘발성 메모리 및 비휘발성 메모리를 포함하는 데이터 베이스가 휘발성 메모리의 일정 용량을 초과한 데이터에 관하여 비휘발성 메모리에 저장하고, 비휘발성 메모리의 리스트 구조 및 영속성 버퍼를 통해 플러시 동작 및 컴팩션 동작을 수행함으로써, 데이터 영속성을 유지하면서 쓰기 지연과 읽기 지연을 최소화하는 효과가 있다.

【0021】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】

【0022】도 1은 기존의 로그 구조 병합 트리 기반의 데이터 베이스를 예시한 도면이다.

도 2는 기존의 로그 구조 병합 트리 기반의 데이터 베이스가 데이터 컴팩션 동작을 수행하는 것을 예시한 도면이다.

도 3은 본 발명의 일 실시예에 따른 데이터 베이스를 예시한 블록도이다.

도 4는 본 발명의 일 실시예에 따른 데이터 베이스의 내부 데이터 구조를 예시한 도면이다.

도 5는 본 발명의 다른 실시예에 따른 데이터 베이스의 데이터 처리 방법을 예시한 흐름도이다.

도 6은 본 발명의 다른 실시예에 따른 데이터 베이스가 비휘발성 메모리에 생성한 스킵 리스트를 예시한 도면이다.

도 7은 본 발명의 다른 실시예에 따른 데이터 베이스가 스킵 리스트에 대해 컴팩션을 수행한 것을 예시한 도면이다.

도 8은 본 발명의 다른 실시예에 따른 데이터 베이스가 영속성 버퍼를 통해 순차적 복사를 수행한 것을 예시한 도면이다.

도 9는 본 발명의 다른 실시예에 따른 데이터 베이스가 영속성 버퍼를 통해 바이트 어드레싱 컴팩션을 수행한 것을 예시한 도면이다.

도 10 내지 도 12는 본 발명의 실시예들에 따라 수행된 모의실험 결과를 도시한 것이다.

【발명을 실시하기 위한 구체적인 내용】

【0023】 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.

【0024】 도 1은 기존의 로그 구조 병합 트리(LSM-Tree) 기반의 데이터 베이스를 예시한 도면이고, 도 2는 기존의 로그 구조 병합 트리 기반의 데이터 베이스가 데이터 컴팩션 동작을 수행하는 것을 예시한 도면이다.

【0025】 LSM-Tree를 이용한 대표적인 데이터 베이스로는 LevelDB와 RocksDB가 있다.

【0026】 LSM-Tree는 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.

【0027】 LSM-Tree 기반의 데이터 베이스는 키-값 형태로 데이터를 저장한다. LSM-Tree 기반의 데이터 베이스에 데이터의 삽입 연산 요청이 들어오면 데이터를 메모리에 기록하기 전에 우선적으로 로그 파일에 로그를 기록한다. 로그를 기록한 다음 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.

【0028】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.

【0029】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.

【0030】트랜잭션 수행 도중에 시스템 오류 또는 전원 차단 등과 같은 문제가 발생하면, 아직 디스크에 반영되지 않고 버퍼에 남아있는 데이터는 유실된다. 시스템이 재부팅된 후 데이터 베이스가 복구를 수행할 때, 트랜잭션이 어떤 갱신 연산을 수행했는지 기록하는 로그를 사용한다. 로그 기록 방식으로는 WAL(Write-Ahead-Logging) 규칙이 있다. WAL은 트랜잭션으로 인해 변경된 데이터가 디스크에 기록되기 전에 관련된 로그를 로그 파일에 기록하는 규칙이다.

【0031】LSM-Tree 기반의 데이터 베이스는 두 개의 명령어를 수행한다. 하나는 메모리에서 디스크로 넘어가는 플러시 명령어이고, 다른 하나는 디스크의 레벨들을 조정하는 컴팩션 명령어이다.

【0032】플러시 명령어를 수행할 때, 불변 멤테이블은 단일 SST 파일로 변경된다. 대량의 데이터가 한꺼번에 입력되면, 플러시 속도의 균형을 맞추고 SST 파일의 레벨의 용량 한계치를 유지하기 위해서 의도적으로 플러시 속도를 조절한다. 이러한 의도된 지연을 'Write Stall'이라고 한다. 표 1에 누적된 Write Stall이 예시되어 있다.

【0033】**【표 1】**

Data size	Accumulated write stalls on SSD(us)
2 GB	61,902,809
4 GB	310,872,755
6 GB	569,200,925
8 GB	901,254,333

【0034】LSM-Tree의 각각의 레벨은 특성이 구분된다. 컴팩션 비용과 디스크 쓰기는 특정 레벨에 집중된다. 계층적 저장 구조는 상위 레벨에서 하위 레벨로 점진적인 데이터 누적을 야기한다. SST 파일의 수명은 해당하는 레벨에서 존재하는 동안 컴팩션을 수행하는 횟수를 의미한다. 컴팩션을 수행하는 동안 SST 파일이 특

정 레벨에서 삭제되지 않으면, 해당 SST 파일의 수명은 높게 나타난다. 표 2에 SST 파일의 수명, 컴팩션 파일의 개수, 컴팩션 파일의 비율, 및 컴팩션 동안 쓰기량이 예시되어 있다.

【0035】【표 2】

Level	Average lifetime of SST files	The number of compacted files	The ratio of compacted files	The amount of write during compaction (MB)
L0	4.498	35434	99.989%	113230
L1	10.679	118038	99.986%	196361
L2	51.163	379120	99.961%	511559
L3	403.795	481843	99.728%	650031
L4	4340.665	339725	96.629%	599837

【0036】컴팩션 명령어를 수행한 각 레벨의 결과를 살펴보면, 데이터 컴팩션을 수행하는 짧은 시간 동안에 상위 레벨(ex. L_0 to L_3)의 SST 파일들은 생성되고 삭제됨을 나타낸다. 컴팩션 파일의 개수와 컴팩션 파일의 비율을 보면, LSM-Tree의 계층적 구조로 인하여 컴팩션 파일의 크기가 작지 않음을 나타낸다.

【0037】컴팩션 동안 예비 자원을 수반하는 상위 레벨에서의 쓰기량은 중요하지 않아 보이지만, 디스크 I/O를 피할 수 없다. 상당히 많은 데이터를 입력했음에도, 하위 레벨 L_4의 컴팩션 파일의 개수와 쓰기량은 L_3보다 작은 값을 나타낸다.

【0038】본 실시예에 따른 데이터 베이스는 L_4와 같은 하위 레벨에서 높은 값을 갖는 SST 파일의 수명에 집중해서, NVM을 통해 상위 레벨에서 낭비되는 디스크 I/O를 감소시킨다. NVM은 바이트 어드레싱을 이용하여 쓰기 증폭(Write

Amplication)을 해결하고 영속적 컴팩션을 수행하게 한다.

【0039】 도 3은 본 발명의 일 실시예에 따른 데이터 베이스를 예시한 블록도이고, 도 4는 본 발명의 일 실시예에 따른 데이터 베이스의 내부 데이터 구조를 예시한 도면이다.

【0040】 도 3에 도시한 바와 같이, 데이터 베이스(10)는 프로세서(100), 휘발성 메모리(200), 및 비휘발성 메모리(300)를 포함한다. 데이터 베이스(10)는 도 3에서 예시적으로 도시한 다양한 구성요소들 중에서 일부 구성요소를 생략하거나 다른 구성요소를 추가로 포함할 수 있다. 예컨대, 데이터 베이스(10)는 단계화 정책에 따라 블록 디바이스(400)를 추가로 포함할 수 있다.

【0041】 데이터 베이스(10)는 데이터를 가공 및 저장하는 장치이다. 데이터 베이스(10)는 키-값 형식으로 데이터를 저장하고 읽을 수 있다. 키-값의 처리 명령은 (SET K, V), (DEL K, V) 등으로 정의될 수 있다.

【0042】 프로세서(100)는 휘발성 메모리(200), 비휘발성 메모리(300), 및 블록 디바이스(400)에 기 정의된 명령어를 전송하여, 각종 신호 및 데이터 흐름을 제어한다.

【0043】 휘발성 메모리(200)는 저장된 정보를 계속 유지하기 위하여 전원 공급이 필요한 메모리이다. 예컨대, 휘발성 메모리(300)로는 DRAM(Dynamic Random Access Memory) 등이 있다.

【0044】비휘발성 메모리(300)는 전원이 공급되지 않아도 저장된 정보를 계속 유지하는 메모리이다. 비휘발성 메모리(300)는 리스트 구조(310)를 포함할 수 있다. 리스트 구조(310)는 헤드(Head)와 리어(Rear)를 갖고, 키의 주소와 값의 주소를 각각 갖는다. 리스트 구조(310)는 다수의 다음 포인터를 갖는 스킵 리스트로 구현될 수 있다. 스킵 리스트는 각 노드마다 키 길이, 키, 값 길이, 및 값을 갖는다. 비휘발성 메모리(300)는 영속성 버퍼(320)를 포함할 수 있다.

【0045】블록 디바이스(400)는 블록 단위로 임의 접근이 가능한 저장매체이다. 예컨대, 블록 디바이스(400)로는 HDD(Hard Disk Drive), SSD(Solid State Drive) 등이 있다.

【0046】데이터 베이스(10)는 휘발성 메모리(200)에 1차적으로 데이터를 저장한다. 저장된 데이터가 기 설정된 용량을 초과하면, 일부 데이터를 비휘발성 메모리(300)에 2차적으로 저장한다. 데이터 베이스(10)는 비휘발성 메모리(300)에 복수의 노드가 연결된 리스트 구조(310)를 생성하고 데이터를 리스트 구조(310)에 저장하는 방식으로 플러시 동작을 수행할 수 있다.

【0047】도 5는 본 발명의 다른 실시예에 따른 데이터 베이스의 데이터 처리 방법을 예시한 흐름도이다.

【0048】단계 S210에서 데이터 베이스는 휘발성 메모리에 데이터를 저장한다.

【0049】단계 S220에서 데이터 베이스는 비휘발성 메모리에 플러시(flush) 동작을 수행한다. 플러시는 제1 저장소에서 제2 저장소로 복사하는 동작이다. 예컨대, 휘발성 메모리에서 비휘발성 메모리로 데이터를 복사한다.

【0050】단계 S230에서 데이터 베이스는 컴팩션(Compaction) 동작을 수행한다. 컴팩션은 병합 과정으로 특정 레벨의 임계치까지 데이터가 차면 해당 레벨의 데이터를 하위 레벨로 내려주는 동작이다.

【0051】단계 S240에서 데이터 베이스는 정책에 따라 블록 드라이브로 데이터를 방출(Eviction)하는 동작을 수행한다.

【0052】단계 S250에서 데이터 베이스는 오류가 발생하면 데이터를 복구(Recovery)하는 동작을 수행한다.

【0053】도 6은 본 발명의 다른 실시예에 따른 데이터 베이스가 비휘발성 메모리에 생성한 스킵 리스트를 예시한 도면이다.

【0054】비휘발성 메모리(Non-Volatile Memory, NVM)는 중간 유연 레벨에 해당하며, 스킵 리스트는 기존의 블록 파일 형식(ex. SST 파일)을 대체할 수 있다.

【0055】비휘발성 메모리(Non-Volatile Memory, NVM)는 바이트 어드레싱(Byte Addressability)이 가능하다. 바이트 어드레싱이 가능한 비휘발성 메모리의 예시로는 STT-MRAM(Spin-Transfer Torque Magnetic Random Access Memory), PCM(Phase-Change Memory) 등이 있다.

【0056】데이터 베이스 시스템에서 NVM이 일관적으로 동작하려면, 데이터 베이스 시스템은 'cflush' 및 'mfence' 명령어를 사용해야 한다. 'cflush' 명령어는 메모리에 캐시 라인을 플러시하는 명령어이며, NVM에 데이터를 완전하게 저장하는 것을 보장한다. 'mfence' 명령어는 명령어들의 재순서 배정으로부터 프로세서를 보호하는 메모리 장벽 명령어이다.

【0057】PMDK(Persistent Memory Development Kit) API는 NVM를 이용하여 키-값 기반의 데이터 베이스를 제공한다.

【0058】도 7은 본 발명의 다른 실시예에 따른 데이터 베이스가 스킵 리스트에 대해 컴팩션을 수행한 것을 예시한 도면이다.

【0059】데이터 베이스가 데이터 쓰기를 수행하지 않고, 새로운 리스트 구조를 생성하고 새로운 리스트 구조가 기존 리스트 구조의 노드에 할당된 키-값을 포인팅하는 방식으로 컴팩션 동작을 수행한다.

【0060】도 8은 본 발명의 다른 실시예에 따른 데이터 베이스가 영속성 버퍼를 통해 순차적 복사를 수행한 것을 예시한 도면이다.

【0061】데이터 베이스의 플러시 동작에 관한 알고리즘은 표 3와 같다.

【0062】 **【표 3】**

```
Input:
memtableIter : key-value iterator from immutable memtable
Output:
fileMetadata : metadata including new output number, smallest/largest key, filesize
    /* Step 1. Iteration about all data in memtable */
1:  buf ← [] // append all KV-pair, then memcpy at a time
2:  pBuf ← create new persistent buffer
3:  startOffset ← get new start offset from persistent buffer
4:  pos ← 0 // relative position from start_offset
5:  pSkiplist ← create new persistent skip list with new ID
    /* Step 2. Iteration about all data in memtable */
6:  for KV-pair in memtableIter do
7:  |  (key, value) ← Encode(KV-pair)
8:  |  entryLength ← key.length + value.length
9:  |  Append(buf ← key, value)
    |  /* Insertion into skip list node through the relative position of pBuf */
10: |  pSkiplist.current ← Node(startOffset + pos, pSkiplist.prev)
11: |  Persist(pSkiplist.current)
12: |  pSkiplist.current.next ← pSkiplist.prev.next
13: |  Persist(pSkiplist.current.next)
14: |  pSkiplist.prev.next ← pSkiplist.current
15: |  Persist(pSkiplist.prev.next)
16: |  pSkiplist.current ← pSkiplist.current.next
17: |  Persist(pSkiplist.current)
18: |  Update(pos ← pos + entryLength)
19: end
    /* Step 3. Finally, add null node for recovery */
20: pSkiplist.current ← NullNode()
21: Persist(pSkiplist.current)
    /* Step 4. For persistent buffer, bulk copy */
22: Copy(pBuf ← buf until buf.size)
23: Persist(pBuf)
24: Update(fileMetadata)
```

【0063】플러시 동작은, 비휘발성 메모리의 영속성 버퍼에 키-값을 순차적으로 복사하고, 영속성 버퍼는 노드에 할당된 키-값의 랜덤 접근을 방지한다. 리스트 구조는 리스트 구조에 대응하는 영속성 버퍼의 오프셋을 포인팅한다.

【0064】도 9는 본 발명의 다른 실시예에 따른 데이터 베이스가 영속성 버퍼를 통해 바이트 어드레싱 컴팩션을 수행한 것을 예시한 도면이다.

【0065】데이터 베이스의 바이트 어드레싱 컴팩션 동작에 관한 알고리즘은 표 4와 같다.

【0066】 【표 4】

```
Input:
mergeIter : merge and sorted iterater from skip list IDs to be compacted
Output:
fileMetadataList: list of metadata including new output number,
                  smallest/largest key, filesize
1:  for KV-pair in mergeIter do
    |  /* Step 1. Get key-value entry by buffer pointer*/
2:  |  if not KV-pair.isBufferPointerValid() then
3:  |  |  break // end of iteration or invalid status
4:  |  else
5:  |  |  bufPtr ← KV-pair.GetBufferPointer()
    |  /* Step 2. Check whether the persistent skip list is valid */
6:  |  if pSkiplist is Null then
7:  |  |  pSkiplist ← CreatePersistentSkiplist(new ID)
8:  |  |  Insert(pSkiplist.metadata ← inputIDs) // for recovery
    |  /* Step 3. Failure-atomic pointing operation with persistent buffer */
9:  |  pSkiplist.current ← Node(bufPtr, pSkiplist.prev)
10: |  Persist(pSkiplist.current)
11: |  pSkiplist.current.next ← pSkiplist.prev.next
12: |  Persist(pSkiplist.current.next)
13: |  pSkiplist.prev.next ← pSkiplist.current
14: |  Persist(pSkiplist.prev.next)
15: |  pSkiplist.current ← pSkiplist.current.next
16: |  Persist(pSkiplist.current)
    |  /* Step 4. Check whether persistent skiplist is full */
17: |  if pSkiplist.isFull() then
18: |  |  pSkiplist.current ← NullNode()
19: |  |  Persist(pSkiplist.current)
20: |  |  Insert(fileMetadataList ← new metadata)
21: |  |  pSkiplist ← Null
22: end
```

【0067】 컴팩션 동작은, 비휘발성 메모리에 새로운 리스트 구조를 생성하고 새로운 리스트 구조가 이전 리스트 구조에 대응하는 영속성 버퍼의 오프셋을 포인

팅한다. 컴팩션 동작을 수행할 때, 영속성 버퍼는 키 길이, 키, 값 길이, 및 값에 대해 쓰기를 수행하지 않는다. NVM에 대해서 'cflush' 및 'mfence' 명령어를 사용하여 영속성을 확보한다. 영속성 버퍼는 SST 파일과 달리 인덱스 블록과 푸터 블록을 포함하지 않는다. 영속성 버퍼의 오프셋을 통해 데이터 검색 성능을 향상시킨다. 컴팩션을 수행하기 전에 스킵 리스트의 최하위 레벨에 대한 반복자들을 생성하고, 바이트 어드레싱 컴팬션 과정에서 반복자를 병합한다.

【0068】 데이터 베이스는 저장소 단계화(Storage Tiering)를 수행한다. 데이터 베이스는 블록 드라이브를 포함한다. 데이터 베이스는 비휘발성 메모리에 저장된 데이터가 기 설정된 용량 범위를 초과하면, 단계화 정책(Tiering Policy)에 따라 데이터 베이스의 블록 드라이브로 방출(Eviction)한다.

【0069】 단계화 정책은, (i) 특정 레벨에 있는 데이터를 선택하여 블록 드라이브에 저장하는 제1 단계화 정책(Leveled Tiering), (ii) 데이터 접근이 오래된 데이터를 선택하여 블록 드라이브에 저장하는 제2 단계화 정책(LRU Tiering), (iii) 모든 데이터를 비휘발성 메모리에 저장하는 제3 단계화 정책(No Tiering), 또는 이들의 조합으로 설정될 수 있다. 특정 레벨은 구현되는 설계에 따라 통계적인 방식으로 산출되어 설정될 수 있다.

【0070】 데이터 베이스를 복구하는 동작에 관한 알고리즘은 표 5와 같다.

【0071】【표 5】

```
Input: None
Output:
fileMetadataList: list of metadata including new output number,
                  smallest/largest key, filesize
    /* Step 1. Get persistent pointer from NVM pool*/
1:  pools ← GetPoolsInDirectory()
2:  persistentPtrs ← GetPersistentPointers(pools)
3:  logFiles ← GetLogFilesInDirectory()
    /* Step 2. Restore metadata of persistent skip list from persistent pointer */
4:  for pptr in persistentPtrs do
5:  |  restoredSkiplist ← pptr.GetSkiplist()
6:  |  last ← restoredSkiplist.SeekToLastNode()
    |  /* Detect system failure occurs */
7:  |  if last.isNullNode() then
    |  |  /* Failure when flush data into skip list at L0 */
8:  |  |  if restoredSkiplist.ID = logFiles.ID then
9:  |  |  |  Reset(restoredSkiplist)
10: |  |  |  memtableIter ← logFiles.createIteratorAboutID()
11: |  |  |  Flush(memtableIter)
    |  |  /* Failure when byte-addressable compaction at lower levels */
12: |  |  else
13: |  |  |  IDs ← restoredSkiplist.metadata.IDs // Get IDs to be compacted
14: |  |  |  mergeIter ← Iterator(IDs)
15: |  |  |  while mergeIter.key <= restoredSkiplist.current.key do
16: |  |  |  |  mergeIter ← mergeIter.next // pass already inserted data
17: |  |  |  end
18: |  |  |  ByteAddressableCompaction(mergeIter) // resume for remaining mergeIter
19: |  |  metadata ← restoredSkiplist.generateMetadata()
20: |  |  Insert(fileMetadata ← metadata)
21: end
```

【0072】데이터 베이스에서 시스템 오류가 발생하면, 데이터 베이스의 비휘발성 메모리에 저장된 리스트 구조가 포인팅하는 데이터를 조회한 결과를 통해 데이터를 순차적으로 복구한다. NVM으로부터 영속성 포인터를 획득하고, 영속성 포인터를 이용하여 스킵 리스트를 복구한다. 스킵 리스트에서 노드를 찾고, 메타데이터

(ID), ID에 대한 반복자를 획득하고, 이미 입력된 데이터는 생략하고, 남은 바이트 어드레싱 컴팩션을 고려한다. 스킵 리스트의 메타 데이터를 복구하고, 메타 데이터를 리스트에 삽입한다.

【0073】 도 10 내지 도 12는 본 발명의 실시예들에 따라 수행된 모의실험 결과를 도시한 것이다.

【0074】 도 10에 도시된 바와 같이, 본 실시예에 따른 데이터 베이스는 쓰기 지연과 읽기 지연 측면에서 성능이 향상됨을 알 수 있다.

【0075】 도 11을 참조하면, 제3 단계화 정책(No Tiering), 제2 단계화 정책(LRU Tiering), 제1 단계화 정책(Leveled Tiering) 순으로 쓰기 지연과 읽기 지연 측면에서 성능이 향상됨을 알 수 있다.

【0076】 도 12를 참조하면, 본 실시예에 따른 데이터 베이스(TLSM)는 쓰기 지연(Write Stall)과 컴팩션의 쓰기량 측면에서 성능이 향상됨을 알 수 있다.

【0077】 데이터 베이스에 포함된 구성요소들이 도 3에서는 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.

【0078】 데이터 베이스는 하드웨어, 펌웨어, 소프트웨어 또는 이들의 조합에 의해 로직회로 내에서 구현될 수 있고, 범용 또는 특정 목적 컴퓨터를 이용하여 구

현될 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.

【0079】 데이터 베이스는 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.

【0080】 도 5에서는 각각의 과정을 순차적으로 실행하는 것으로 기재하고 있으나 이는 예시적으로 설명한 것에 불과하고, 이 분야의 기술자라면 본 발명의 실시예의 본질적인 특성에서 벗어나지 않는 범위에서 도 5에 기재된 순서를 변경하여 실행하거나 또는 하나 이상의 과정을 병렬적으로 실행하거나 다른 과정을 추가하는 것으로 다양하게 수정 및 변형하여 적용 가능할 것이다.

【0081】 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 시스템 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.

【0082】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.

【부호의 설명】

【0083】10: 데이터 베이스　　100: 프로세서

200: 휘발성 메모리　　300: 비휘발성 메모리

310: 리스트 구조　　320: 영속성 버퍼

400: 블록 디바이스

【청구범위】

【청구항 1】

데이터 베이스의 데이터 처리 방법에 있어서,

상기 데이터 베이스의 휘발성 메모리에 데이터를 저장하는 단계; 및

상기 데이터 베이스의 비휘발성 메모리에 복수의 노드가 연결된 리스트 구조를 생성하고 상기 데이터를 상기 리스트 구조에 저장하는 방식으로 플러시 동작을 수행하는 단계

를 포함하는 데이터 베이스의 데이터 처리 방법.

【청구항 2】

제1항에 있어서,

상기 데이터 베이스는 키-값 형식으로 데이터를 저장하고,

상기 리스트 구조는 다수의 다음 포인터를 갖는 스킵 리스트인 것을 특징으로 하는 데이터 베이스의 데이터 처리 방법.

【청구항 3】

제1항에 있어서,

상기 데이터 베이스가 데이터 쓰기를 수행하지 않고, 새로운 리스트 구조를 생성하고 상기 새로운 리스트 구조가 기존 리스트 구조의 노드에 할당된 키-값을 포인팅하는 방식으로 컴팩션 동작을 수행하는 단계를 포함하는 것을 특징으로 하는 데이터 베이스의 데이터 처리 방법.

【청구항 4】

제3항에 있어서,

상기 플러시 동작을 수행하는 단계는,

상기 비휘발성 메모리의 영속성 버퍼(Persistent Buffer)에 키-값을 순차적으로 복사하고,

상기 영속성 버퍼는 상기 노드에 할당된 키-값의 랜덤 접근을 방지하며,

상기 리스트 구조는 상기 리스트 구조에 대응하는 영속성 버퍼의 오프셋을 포인팅하는 것을 특징으로 하는 데이터 베이스의 데이터 처리 방법.

【청구항 5】

제3항에 있어서,

상기 컴팩션 동작을 수행하는 단계는,

상기 비휘발성 메모리에 새로운 리스트 구조를 생성하고 상기 새로운 리스트 구조가 이전 리스트 구조에 대응하는 영속성 버퍼의 오프셋을 포인팅하는 것을 특징으로 하는 데이터 베이스의 데이터 처리 방법.

【청구항 6】

제1항에 있어서,

상기 데이터 베이스의 비휘발성 메모리에 저장된 데이터가 기 설정된 용량 범위를 초과하면, 단계화 정책(Tiering Policy)에 따라 상기 데이터 베이스의 블록 드라이브로 방출(Eviction)하는 단계를 포함하는 것을 특징으로 하는 데이터 베이

스의 데이터 처리 방법.

【청구항 7】

제6항에 있어서,

상기 단계화 정책은,

(i) 특정 레벨에 있는 데이터를 선택하여 상기 블록 드라이브에 저장하는 제1 단계화 정책, (ii) 데이터 접근이 오래된 데이터를 선택하여 상기 블록 드라이브에 저장하는 제2 단계화 정책, (iii) 모든 데이터를 상기 비휘발성 메모리에 저장하는 제3 단계화 정책, 또는 이들의 조합으로 설정되는 것을 특징으로 하는 데이터 베이스의 데이터 처리 방법.

【청구항 8】

제1항에 있어서,

상기 데이터 베이스에서 시스템 오류가 발생하면, 상기 데이터 베이스의 비휘발성 메모리에 저장된 상기 리스트 구조가 포인팅하는 데이터를 조회한 결과를 통해 데이터를 순차적으로 복구하는 단계를 포함하는 것을 특징으로 하는 데이터 베이스의 데이터 처리 방법.

【청구항 9】

프로세서, 휘발성 메모리, 및 비휘발성 메모리를 포함하는 데이터 베이스에 있어서,

상기 휘발성 메모리에 데이터를 저장하고,

상기 비휘발성 메모리에 복수의 노드가 연결된 리스트 구조를 생성하고 상기 데이터를 상기 리스트 구조에 저장하는 방식으로 플러시 동작을 수행하는 것을 특징으로 하는 데이터 베이스.

【청구항 10】

제9항에 있어서,

상기 데이터 베이스는 키-값 형식으로 데이터를 저장하고,

상기 리스트 구조는 다수의 다음 포인터를 갖는 스킵 리스트인 것을 특징으로 하는 데이터 베이스.

【청구항 11】

제9항에 있어서,

상기 데이터 베이스가 데이터 쓰기를 수행하지 않고, 새로운 리스트 구조를 생성하고 상기 새로운 리스트 구조가 기존 리스트 구조의 노드에 할당된 키-값을 포인팅하는 방식으로 컴팩션 동작을 수행하는 것을 특징으로 하는 데이터 베이스.

【청구항 12】

제11항에 있어서,

상기 플러시 동작은,

상기 비휘발성 메모리의 영속성 버퍼에 키-값을 순차적으로 복사하고,

상기 영속성 버퍼는 상기 노드에 할당된 키-값의 랜덤 접근을 방지하며,

상기 리스트 구조는 상기 리스트 구조에 대응하는 영속성 버퍼의 오프셋을

포인팅하는 것을 특징으로 하는 데이터 베이스.

【청구항 13】

제11항에 있어서,

상기 컴팩션 동작은,

상기 비휘발성 메모리에 새로운 리스트 구조를 생성하고 상기 새로운 리스트 구조가 이전 리스트 구조에 대응하는 영속성 버퍼의 오프셋을 포인팅하는 것을 특징으로 하는 데이터 베이스.

【청구항 14】

제9항에 있어서,

상기 데이터 베이스는 블록 드라이브를 포함하며,

상기 데이터 베이스는 상기 비휘발성 메모리에 저장된 데이터가 기 설정된 용량 범위를 초과하면, 단계화 정책(Tiering Policy)에 따라 상기 데이터 베이스의 블록 드라이브로 방출(Eviction)하는 것을 특징으로 하는 데이터 베이스.

【청구항 15】

제14항에 있어서,

상기 단계화 정책은,

(i) 특정 레벨에 있는 데이터를 선택하여 상기 블록 드라이브에 저장하는 제1 단계화 정책, (ii) 데이터 접근이 오래된 데이터를 선택하여 상기 블록 드라이브에 저장하는 제2 단계화 정책, (iii) 모든 데이터를 상기 비휘발성 메모리에 저장

하는 제3 단계화 정책, 또는 이들의 조합으로 설정되는 것을 특징으로 하는 데이터 베이스.

【청구항 16】

제9항에 있어서,

상기 데이터 베이스에서 시스템 오류가 발생하면, 상기 데이터 베이스의 비휘발성 메모리에 저장된 상기 리스트 구조가 포인팅하는 데이터를 조회한 결과를 통해 데이터를 순차적으로 복구하는 것을 특징으로 하는 데이터 베이스.

【요약서】

【요약】

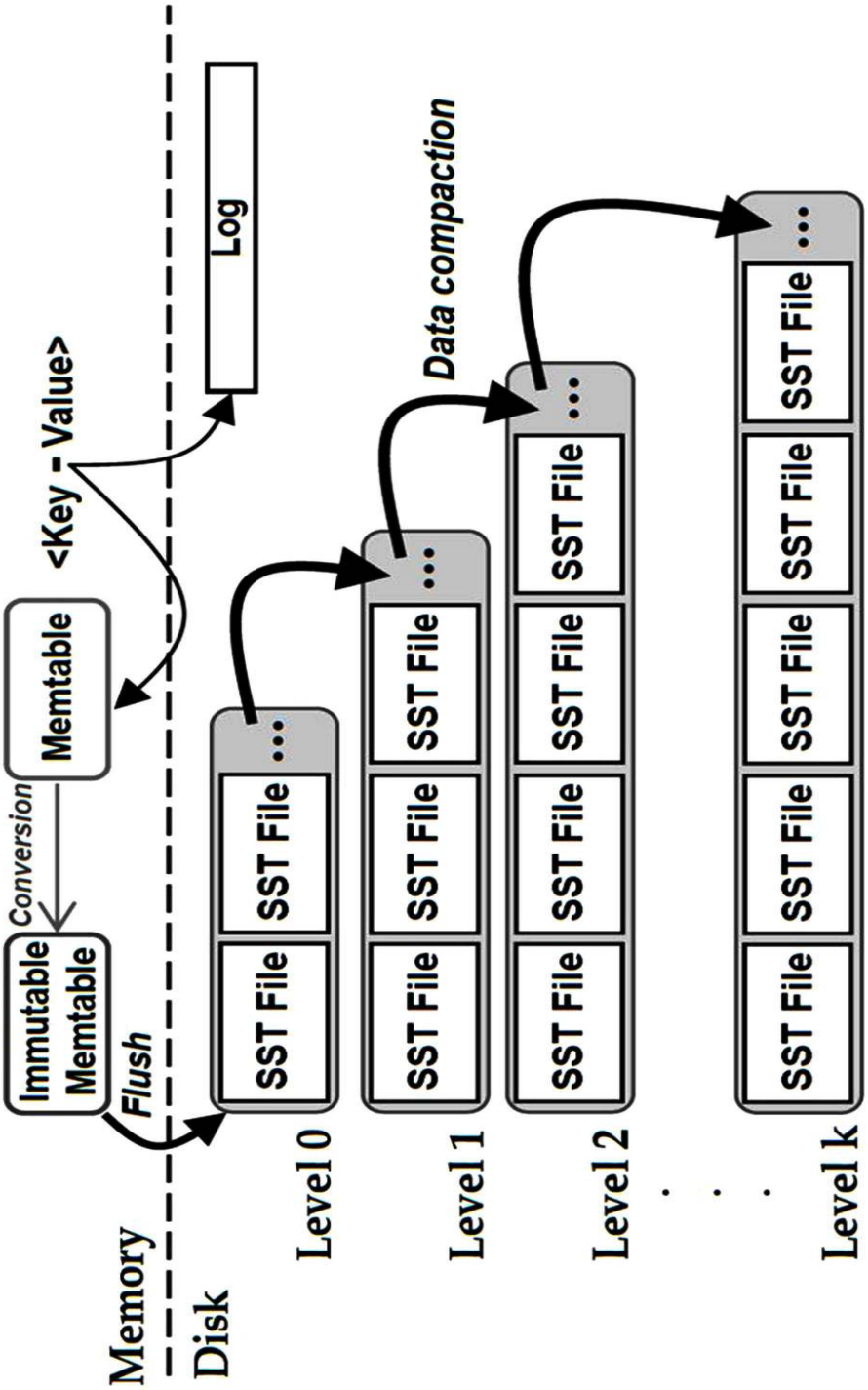
본 실시예들은 휘발성 메모리의 일정 용량을 초과한 데이터에 관하여 비휘발성 메모리에 저장하고, 비휘발성 메모리의 리스트 구조 및 영속성 버퍼를 통해 플러시 동작 및 컴팩션 동작을 수행함으로써, 데이터 영속성을 유지하면서 쓰기 지연과 읽기 지연을 최소화할 수 있는 데이터 베이스를 제공한다.

【대표도】

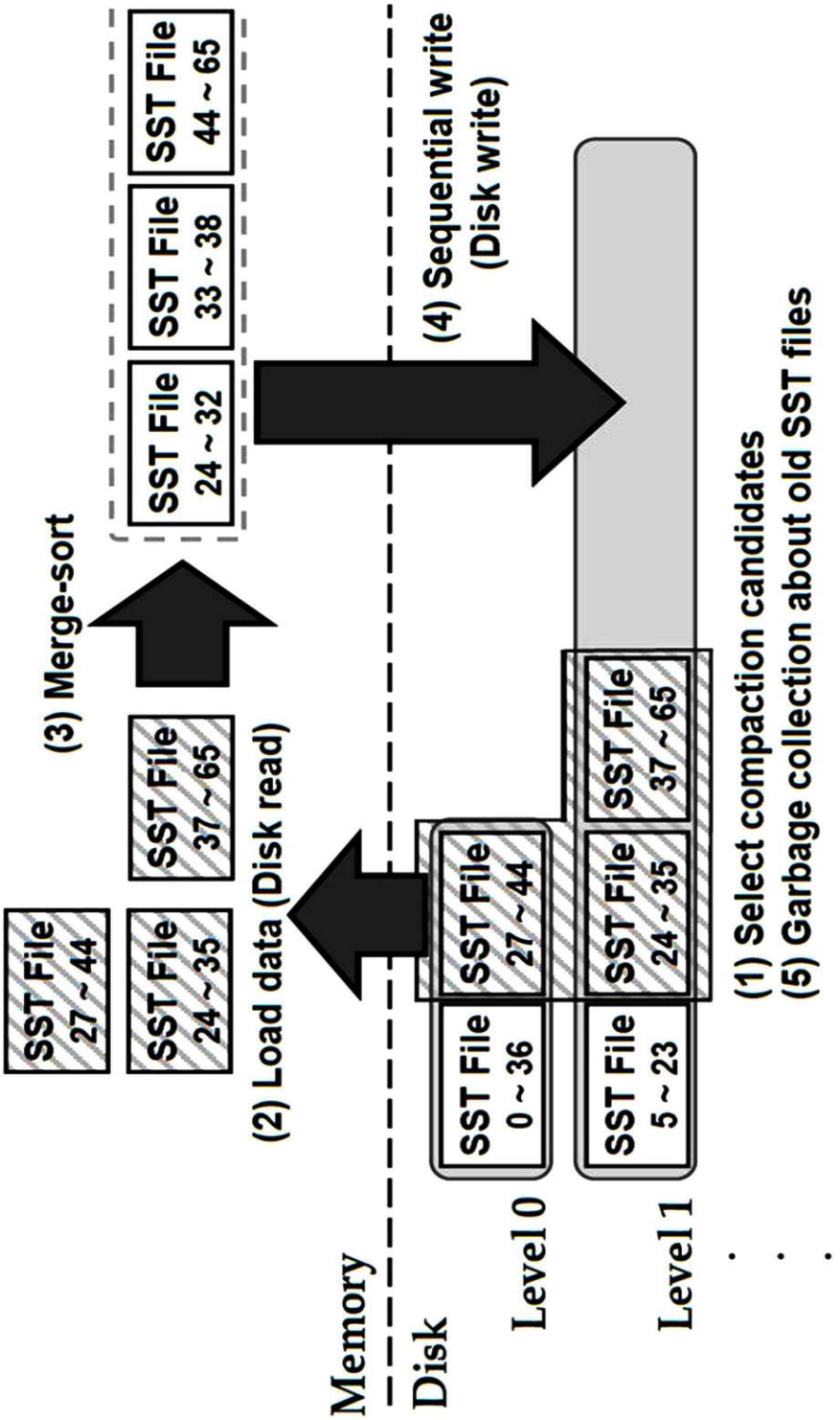
도 3

【도면】

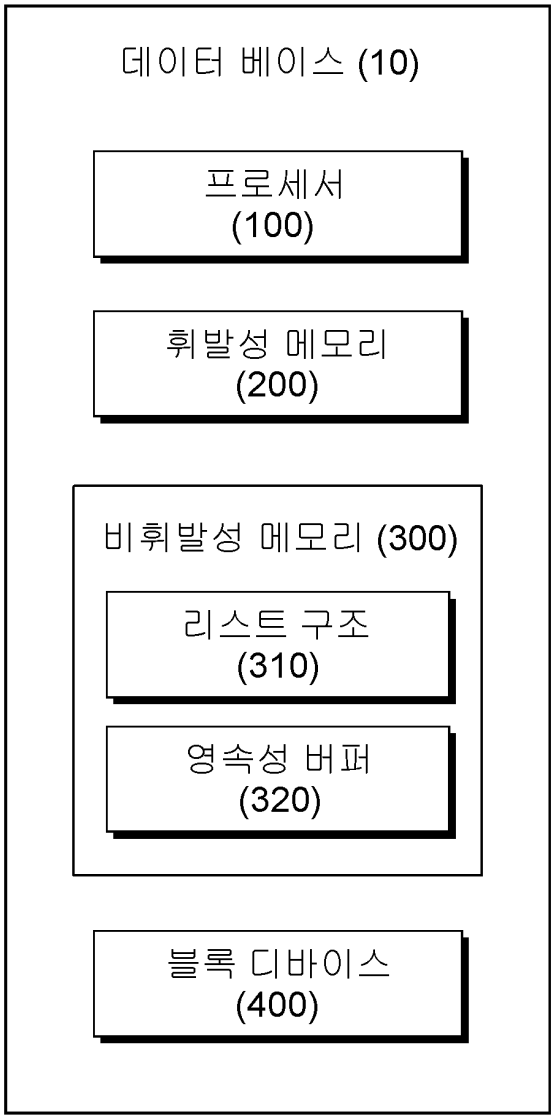
【도 1】


Memory
Disk
Log
<Key - Value>
Memtable
Conversion
Immutable Memtable
Flush
Data compaction
SST File
Level 0
Level 1
Level 2
Level k

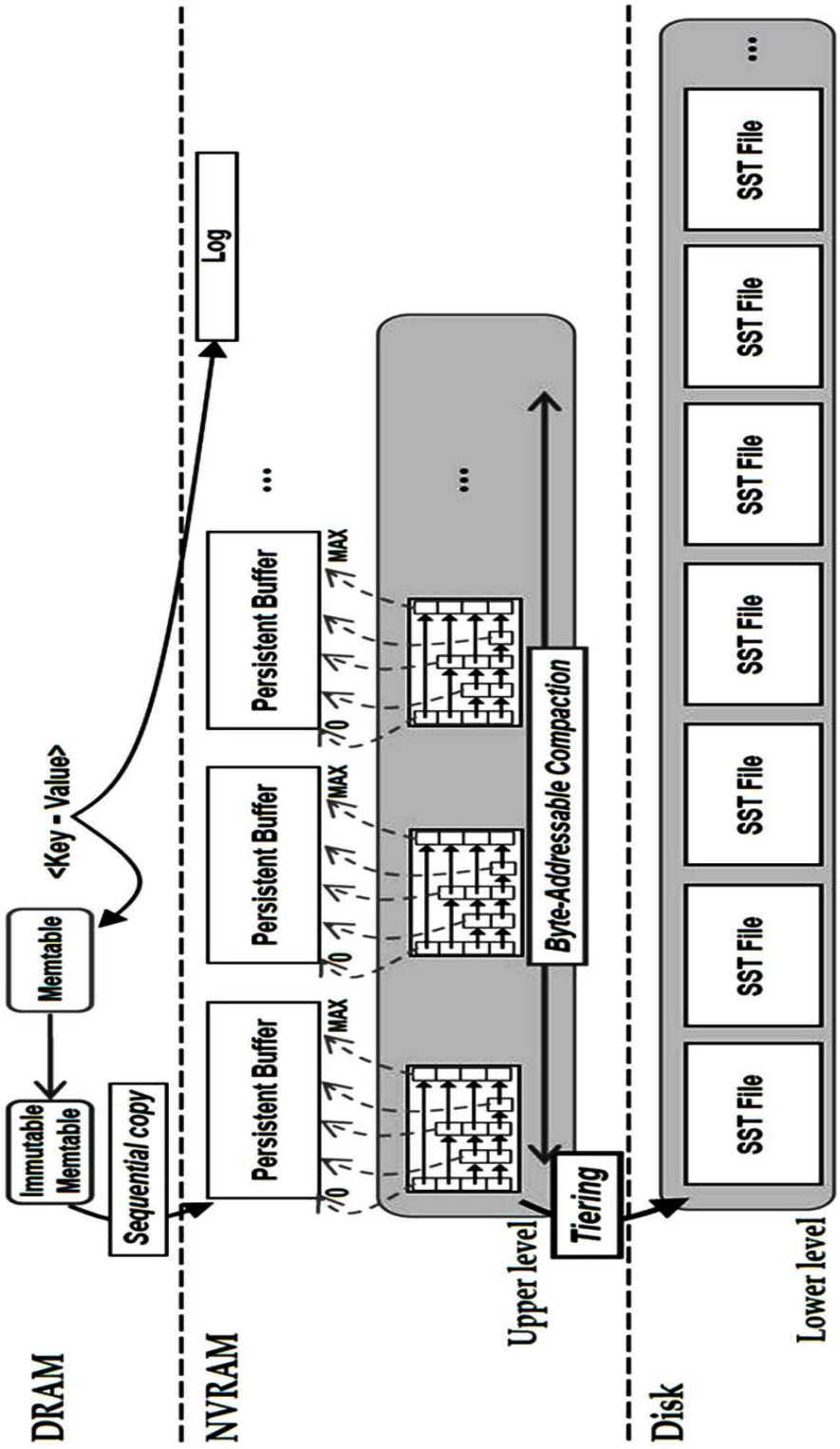
【도 2】


SST File 27 ~ 44
SST File 24 ~ 35
SST File 37 ~ 65
(3) Merge-sort
SST File 24 ~ 32
SST File 33 ~ 38
SST File 44 ~ 65
(2) Load data (Disk read)
(4) Sequential write (Disk write)
Memory
Disk
Level 0
Level 1
SST File 0 ~ 36
SST File 27 ~ 44
SST File 5 ~ 23
SST File 24 ~ 35
SST File 37 ~ 65
(1) Select compaction candidates
(5) Garbage collection about old SST files

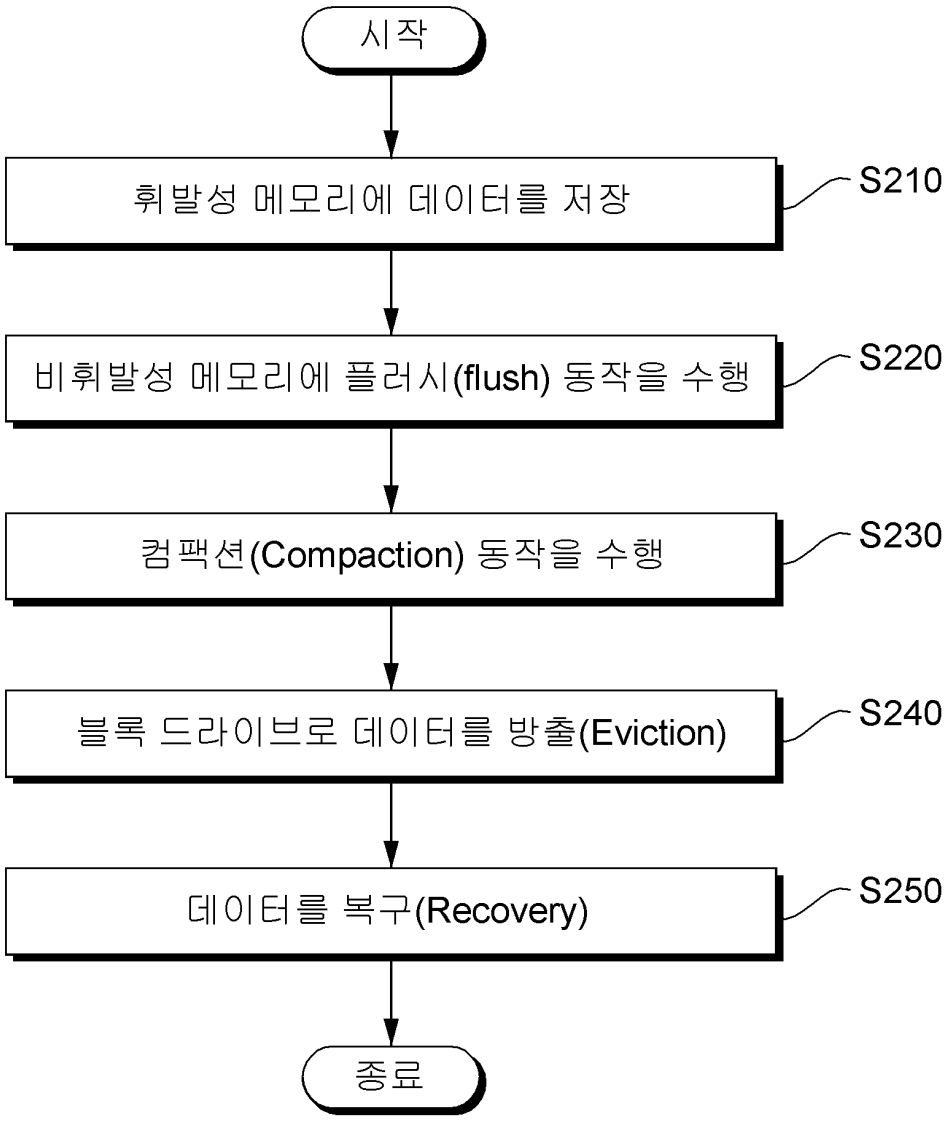
【도 3】


데이터 베이스 (10)
프로세서
(100)
휘발성 메모리
(200)
비휘발성 메모리 (300)
리스트 구조
(310)
영속성 버퍼
(320)
블록 디바이스
(400)

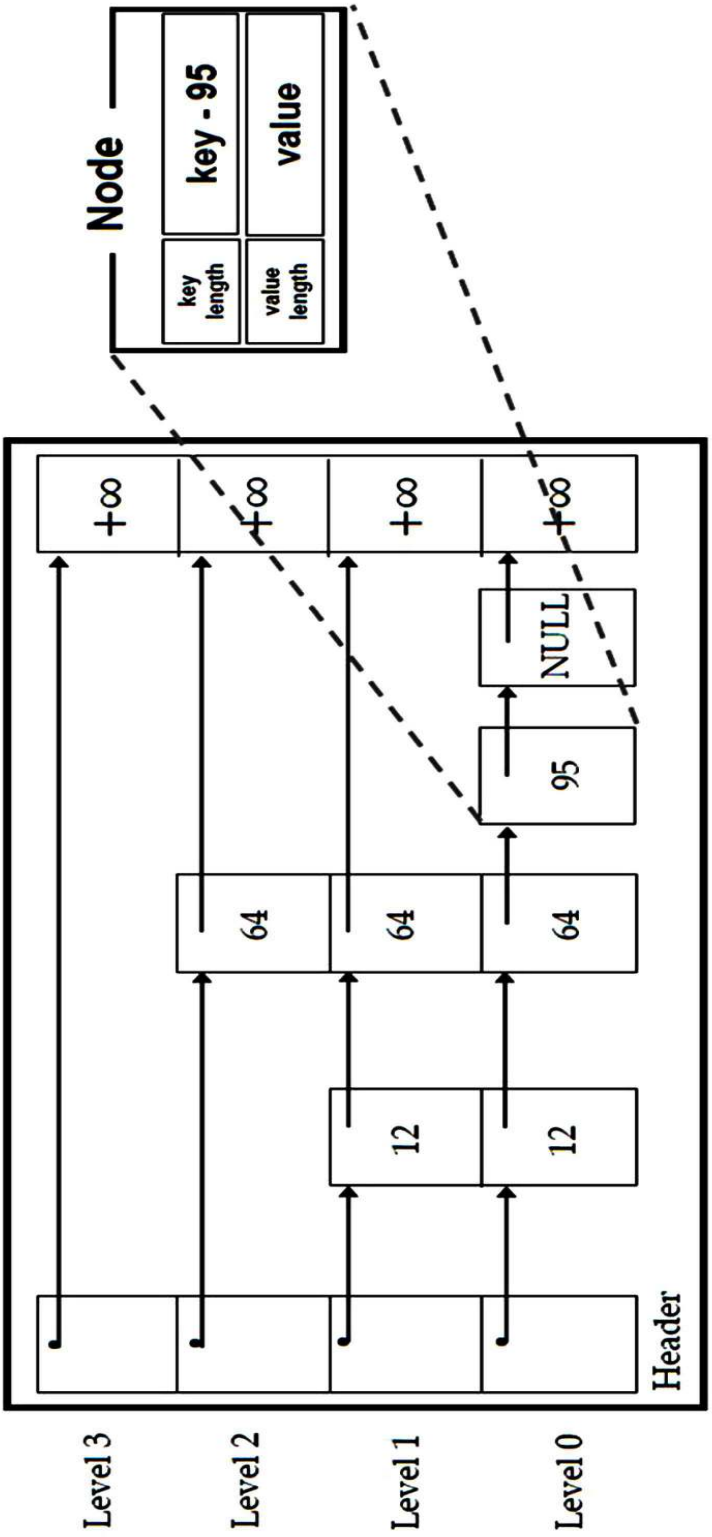
【도 4】


DRAM
NVRAM
Disk
Log
<Key - Value>
Memtable
Immutable Memtable
Sequential copy
Persistent Buffer
MAX
Byte-Addressable Compaction
Tiering
Upper level
Lower level
SST File

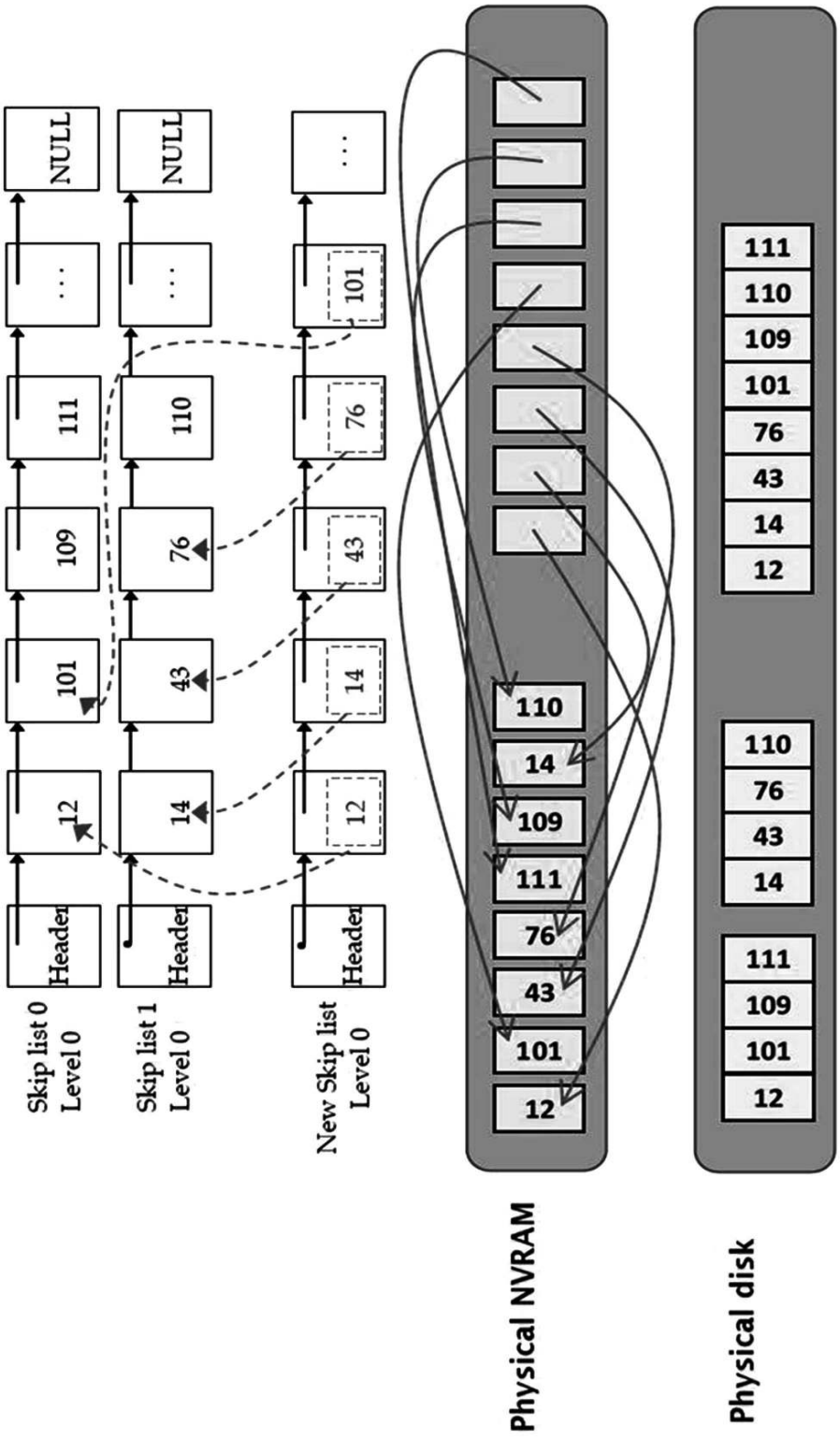
【도 5】


시작
휘발성 메모리에 데이터를 저장
S210
비휘발성 메모리에 플러시(flush) 동작을 수행
S220
컴팩션(Compaction) 동작을 수행
S230
블록 드라이브로 데이터를 방출(Eviction)
S240
데이터를 복구(Recovery)
S250
종료

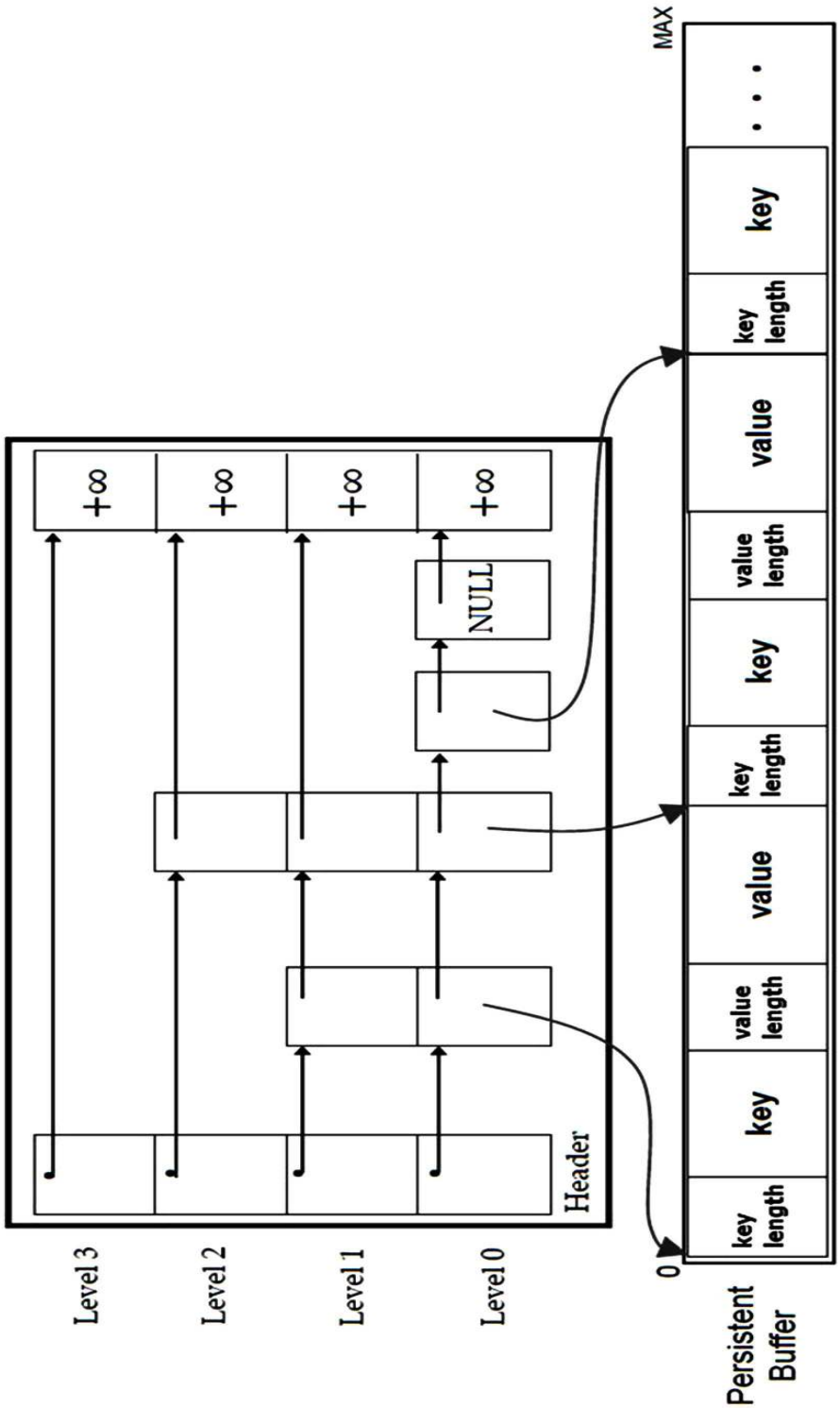
【도 6】


Node
key - 95
value
key length
value length
+∞
+∞
+∞
+∞
NULL
95
64
64
64
12
12
Header
Level 3
Level 2
Level 1
Level 0

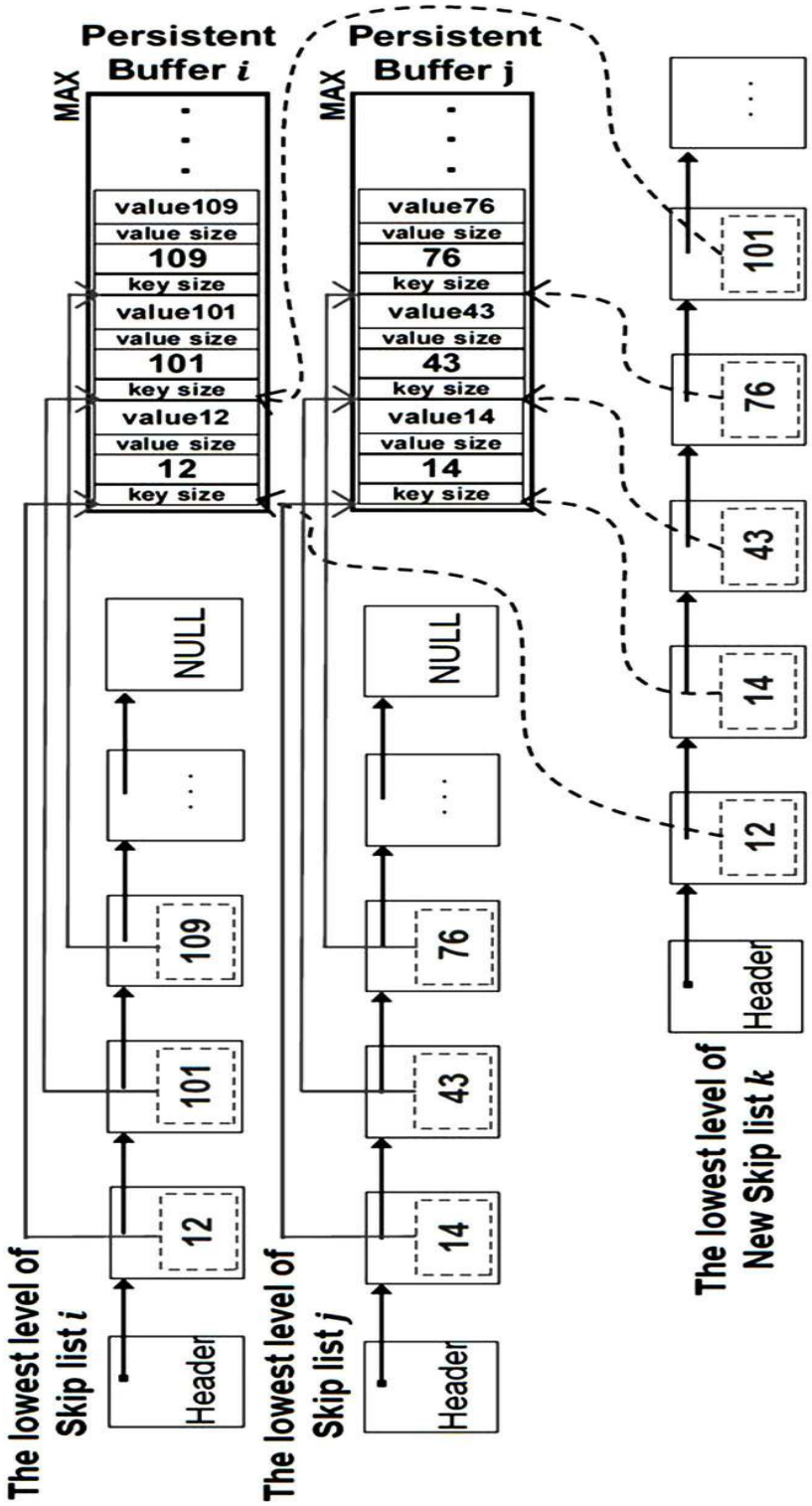
【도 7】


Header
12
101
109
111
...
NULL
Skip list 0
Level 0
Header
14
43
76
110
...
NULL
Skip list 1
Level 0
Header
12
14
43
76
101
...
New Skip list
Level 0
Physical NVRAM
12
101
43
76
111
109
14
110
Physical disk
12
43
14
101
109
111
12
43
14
76
101
109
110
111

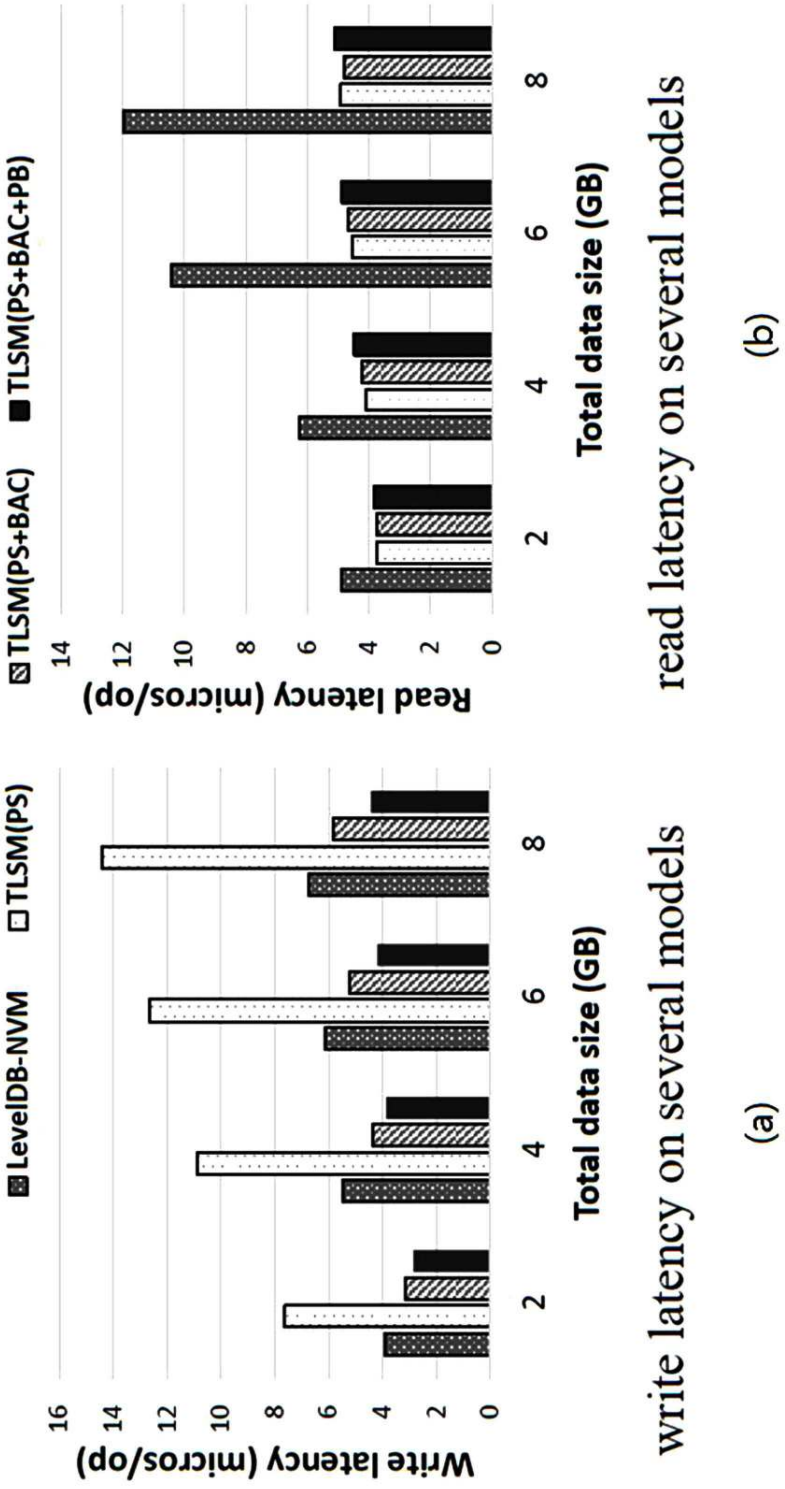
【도 8】


Level 3
Level 2
Level 1
Level 0
Header
+∞
NULL
0
MAX
Persistent Buffer
key length
key
value length
value
. . .

【도 9】


Persistent Buffer i
MAX
value109
value size
109
key size
value101
value size
101
key size
value12
value size
12
key size
Persistent Buffer j
MAX
value76
value size
76
key size
value43
value size
43
key size
value14
value size
14
key size
NULL
109
101
12
Header
76
43
14
Header
101
76
43
14
12
Header
The lowest level of Skip list i
The lowest level of Skip list j
The lowest level of New Skip list k

【도 10】



write latency on several models (a)

read latency on several models (b)

【도 11】

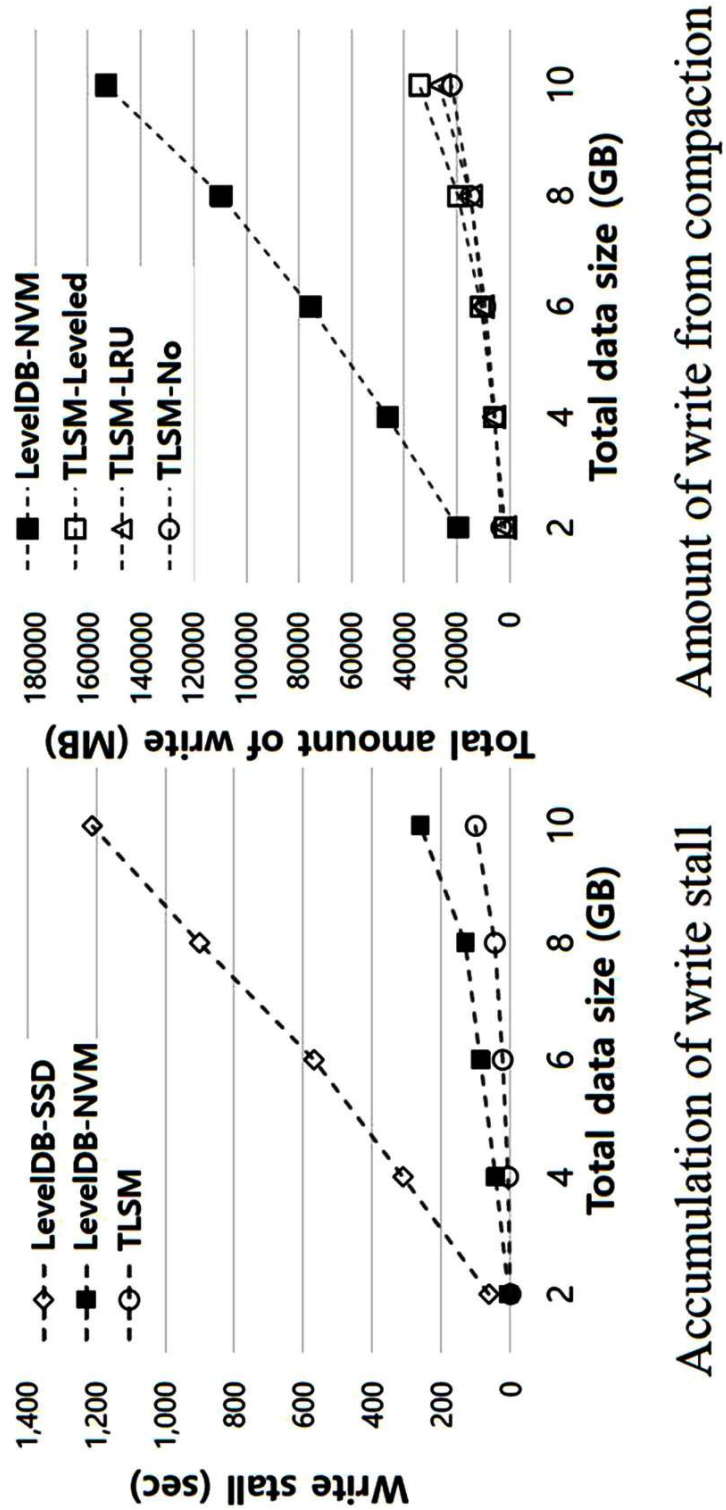
write latency on tiering options

(a)

read latency on tiering options

(b)

【도 12】


Write stall (sec)
1,400
1,200
1,000
800
600
400
200
0
LevelDB-SSD
LevelDB-NVM
TLSM
2
4
6
8
10
Total data size (GB)
Accumulation of write stall
(a)
Total amount of write (MB)
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
LevelDB-NVM
TLSM-Leveled
TLSM-LRU
TLSM-No
2
4
6
8
10
Total data size (GB)
Amount of write from compaction
(b)