



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2021-0135776
(43) 공개일자 2021년11월16일

(51) 국제특허분류(Int. Cl.)
G06F 16/174 (2019.01) G06F 16/13 (2019.01)
(52) CPC특허분류
G06F 16/1744 (2019.01)
G06F 16/13 (2019.01)
(21) 출원번호 10-2020-0053848
(22) 출원일자 2020년05월06일
심사청구일자 2020년05월06일

(71) 출원인
인하대학교 산학협력단
인천광역시 미추홀구 인하로 100(용현동, 인하대학교)
(72) 발명자
김덕환
서울특별시 양천구 목동로 230, 103동 1004호(목동, 아이파크아파트)
키크마틀로
인천광역시 미추홀구 재념이길 141-21, 5동 402호(용현동, 다흥답스빌)
김주환
경기도 과천시 부림4길 29, 101호(부림동)
(74) 대리인
양성보

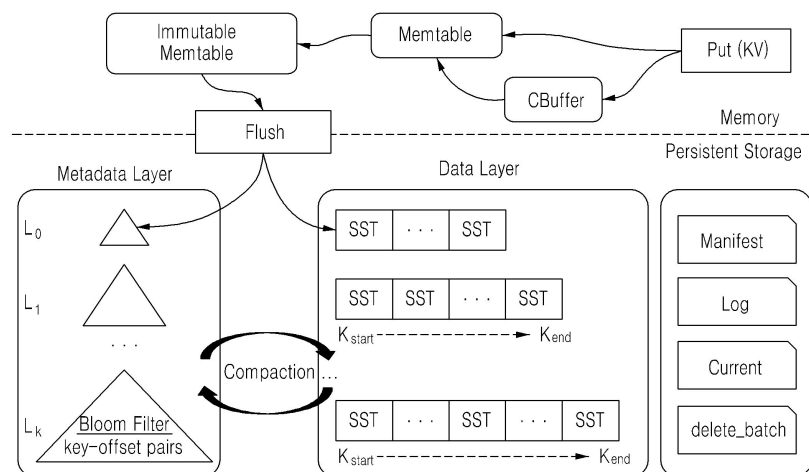
전체 청구항 수 : 총 7 항

(54) 발명의 명칭 엣지 컴퓨팅을 위해 데이터중복제거 기술이 적용된 CASEDB(키 벨류 저장장치)

(57) 요약

일 실시예에 따른 키-벨류(Key-value) 저장장치에서 수행되는 엣지 컴퓨터용 데이터 중복제거 방법은, 키-벨류 저장장치에서 메타데이터 계층으로의 압축 요청이 발생됨을 수신하는 단계; 상기 수신된 압축 요청에 기초하여 메타데이터 파일의 압축을 수행함에 따라 중복된 데이터를 제거하기 위한 중복제거 여부를 체크하는 단계; 및 상기 중복제거 여부를 체크하여 상기 중복된 데이터를 제거하는 단계를 포함할 수 있다.

대표도



이 발명을 지원한 국가연구개발사업

과제고유번호	1711103286
과제번호	2019-0-00064-002
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	방송통신산업기술개발사업
연구과제명	[Ezbaro] 커넥티드 카를 위한 지능형 모바일 엣지 클라우드 솔루션 개발(2차년도)
기 여 율	1/2
과제수행기관명	인하대학교산학협력단
연구기간	2020.01.01 ~ 2020.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호	1345317869
과제번호	2018R1D1A1B07042602
부처명	교육부
과제관리(전문)기관명	한국연구재단
연구사업명	기본연구(교육부)
연구과제명	[Ezbaro] 고가용성, 고신뢰성 클라우드 시스템을 위해 딥러닝 기술을 적용한 Key-Value 기반의 지능형 소프트웨어 정의 스토리지
기 여 율	1/2
과제수행기관명	인하대학교산학협력단
연구기간	2020.03.01 ~ 2021.02.28

명세서

청구범위

청구항 1

키-밸류(Key-value) 저장장치에서 수행되는 엡지 컴퓨터용 데이터 중복제거 방법에 있어서,
 키-밸류 저장장치에서 메타데이터 계층으로의 압축 요청이 발생됨을 수신하는 단계;
 상기 수신된 압축 요청에 기초하여 메타데이터 파일의 압축을 수행함에 따라 중복된 데이터를 제거하기 위한 중복제거 여부를 체크하는 단계; 및
 상기 중복제거 여부를 체크하여 상기 중복된 데이터를 제거하는 단계
 를 포함하는 데이터 중복제거 방법.

청구항 2

제1항에 있어서,
 상기 체크하는 단계는,
 상기 키-밸류 저장장치에서 메타데이터의 압축을 실시하는 제1 과정 및 상기 제1 과정을 통해 업데이트된 메타데이터에 따라 문자열 정렬 테이블(SSTable)을 재배치하는 제2 과정을 수행하는 단계
 를 포함하는 데이터 중복제거 방법.

청구항 3

제2항에 있어서,
 상기 체크하는 단계는,
 상기 제1 과정에서 크기(size) 임계값에 도달하는 레벨에 대해 메타데이터 압축을 지속한 후, 제1 과정 및 제2 과정을 완료함에 따라 메타데이터의 블록 필터를 사용하여 데이터 중복제거 체크를 수행하는 단계
 를 포함하고,
 상기 블록 필터는, 메타데이터를 읽지 않고 키밸류 항목(item)의 존재 여부를 확인하는
 데이터 중복제거 방법.

청구항 4

제1항에 있어서,
 상기 중복된 데이터를 제거하는 단계는,
 상기 메타데이터의 블록 필터를 사용하여 데이터 중복제거 체크를 수행함에 따라 상기 블록 필터로부터 양의 결과 또는 음의 결과가 반환되고, 상기 음의 결과가 반환될 경우, 키가 스킵(skip)되고, 상기 양의 결과가 반환될 경우, 중복 로그(dup_log) 파일에 삽입되는 단계
 를 포함하는 데이터 중복제거 방법.

청구항 5

제4항에 있어서,
 상기 중복된 데이터를 제거하는 단계는,
 블록 필터를 사용하여 데이터 중복 체크를 수행함에 따라 메타데이터의 하위 레벨의 중복된 키를 확인한 후, 중복 비율을 계산하고, 상기 계산된 중복 비율의 값을 다음의 중복제거 여부 체크를 위한 임계값을 확인하는데 사

용하는 단계

를 포함하는 데이터 중복제거 방법.

청구항 6

제5항에 있어서,

상기 중복된 데이터를 제거하는 단계는,

상기 임계값을 확인하여 다음의 중복된 데이터를 제거하기 위한 중복제거 여부를 결정하고, 상기 중복 비율의 값이 기 설정된 기준 이하일 경우, 메타데이터의 압축을 하위 레벨로 확대하고, 중복된 데이터를 자동으로 제거하는 단계

를 포함하는 데이터 중복제거 방법.

청구항 7

옛지 컴퓨터용 데이터 중복제거를 위한 키-밸류(Key-value) 저장장치에 있어서,

메모리에 포함된 컴퓨터 판독 가능한 명령을 실행하도록 구현되는 적어도 하나의 프로세서

를 포함하고,

상기 적어도 하나의 프로세서에 의해,

키-밸류 저장장치에서 메타데이터 계층으로의 압축 요청이 발생됨을 수신하고,

상기 수신된 압축 요청에 기초하여 메타데이터 파일의 압축을 수행함에 따라 중복된 데이터를 제거하기 위한 중복제거 여부를 체크하고,

상기 중복제거 여부를 체크하여 상기 중복된 데이터를 제거하는

것을 특징으로 하는 키-밸류 저장장치.

발명의 설명

기술 분야

[0001] 아래의 설명은 옛지 컴퓨터용 중복제거를 수행하는 키 밸류 저장 장치 및 방법에 관한 것이다.

배경 기술

[0003] 로그형 병합 트리(LSM tree) 계열 키-값 스토어는 빅데이터 시스템에 대한 가장 많은 수요의 데이터베이스가 되고 있다. 이들은 구현하기 쉬운 인터페이스를 제공하며, 다단계 구조 위에 컴파일 절차를 적용하여 가비지 컬렉션(Garbage(쓰레기 수집))을 자동으로 수행한다. CaseDB는 메타데이터 압축 기법을 사용하여 쓰기 증폭을 상당히 줄임으로써 다양한 장점을 제공한다. 그러나 업데이트 집약적인 작업부하에서 공간 증폭 문제를 겪고 있다.

[0004] 이러한 로그형 병합 트리는 하위 레벨이 상위 레벨보다 더 큰 공간을 채용하는 다단계 구조를 구현한다. 새 데이터는 공간이 가장 작고 가장 상위 레벨에 처음 위치하며 공간이 채워짐에 따라 압축 프로세스를 사용하여 더 낮은 레벨로 이동된다. 압축 프로세스는 인접한 두 레벨의 데이터를 병합한 후 더 낮은 레벨 병합된 데이터를 정렬한다. 특정 키 범위에 따라 데이터를 읽고 압축 후 정렬된 데이터를 하위 레벨로 다시 쓴 후 이전 레벨에 있는 오래된 데이터를 삭제한다.

[0005] LSM tree 구조의 구현으로 CaseDB는 삭제를 즉시 수행하지는 않지만 압축 프로세스를 위해 삭제 작업을 지연시킴으로써 사라지게 될(deprecated) 데이터의 양이 증가한다.

발명의 내용

해결하려는 과제

[0007] 키-밸류 저장장치를 위한 데이터 중복제거 확장 압축 방법을 제공할 수 있다.

과제의 해결 수단

- [0009] 키-밸류(Key-value) 저장장치에서 수행되는 엡지 컴퓨터용 데이터 중복제거 방법은, 키-밸류 저장장치에서 메타데이터 계층으로의 압축 요청이 발생됨을 수신하는 단계; 상기 수신된 압축 요청에 기초하여 메타데이터 파일의 압축을 수행함에 따라 중복된 데이터를 제거하기 위한 중복제거 여부를 체크하는 단계; 및 상기 중복제거 여부를 체크하여 상기 중복된 데이터를 제거하는 단계를 포함할 수 있다.
- [0010] 상기 체크하는 단계는, 상기 키-밸류 저장장치에서 메타데이터의 압축을 실시하는 제1 과정 및 상기 제1 과정을 통해 업데이트된 메타데이터에 따라 문자열 정렬 테이블(SSTable)을 재배치하는 제2 과정을 수행하는 단계를 포함할 수 있다.
- [0011] 상기 체크하는 단계는, 상기 제1 과정에서 크기(size) 임계값에 도달하는 레벨에 대해 메타데이터 압축을 지속한 후, 제1 과정 및 제2 과정을 완료함에 따라 메타데이터의 블록 필터를 사용하여 데이터 중복제거 체크를 수행하는 단계를 포함하고, 상기 블록 필터는, 메타데이터를 읽지 않고 키밸류 항목(item)의 존재 여부를 확인할 수 있다.
- [0012] 상기 중복된 데이터를 제거하는 단계는, 상기 메타데이터의 블록 필터를 사용하여 데이터 중복제거 체크를 수행함에 따라 상기 블록 필터로부터 양의 결과 또는 음의 결과가 반환되고, 상기 음의 결과가 반환될 경우, 키가 스킵(skip)되고, 상기 양의 결과가 반환될 경우, 중복 로그(dup_log) 파일에 삽입되는 단계를 포함할 수 있다.
- [0013] 상기 중복된 데이터를 제거하는 단계는, 블록 필터를 사용하여 데이터 중복 체크를 수행함에 따라 메타데이터의 하위 레벨의 중복된 키를 확인한 후, 중복 비율을 계산하고, 상기 계산된 중복 비율의 값을 다음의 중복제거 여부 체크를 위한 임계값을 확인하는데 사용하는 단계를 포함할 수 있다.
- [0014] 상기 중복된 데이터를 제거하는 단계는, 상기 임계값을 확인하여 다음의 중복된 데이터를 제거하기 위한 중복제거 여부를 결정하고, 상기 중복비율의 값이 기 설정된 기준 이하일 경우, 메타데이터의 압축을 하위 레벨로 확대하고, 중복된 데이터를 자동으로 제거하는 단계를 포함할 수 있다.
- [0015] 엡지 컴퓨터용 데이터 중복제거를 위한 키-밸류(Key-value) 저장장치는, 메모리에 포함된 컴퓨터 판독 가능한 명령을 실행하도록 구현되는 적어도 하나의 프로세서를 포함하고, 상기 적어도 하나의 프로세서에 의해, 키-밸류 저장장치에서 메타데이터 계층으로의 압축 요청이 발생됨을 수신하고, 상기 수신된 압축 요청에 기초하여 메타데이터 파일의 압축을 수행함에 따라 중복된 데이터를 제거하기 위한 중복제거 여부를 체크하고, 상기 중복제거 여부를 체크하여 상기 중복된 데이터를 제거할 수 있다.

발명의 효과

- [0017] 일 실시예에 따르면, 데이터 중복 문제를 해결할 수 있다.
- [0018] 일 실시예에 따르면, 공간 증폭을 감소시키고, 공간 증폭과 쓰기 증폭의 균형을 유지시킬 수 있다.
- [0019] 일 실시예에 따르면, 공간 증폭과 쓰기 처리량 간의 균형을 유지할 수 있다.

도면의 간단한 설명

- [0021] 도 1은 일 실시예에 따른 키-밸류 저장장치를 설명하기 위한 도면이다.
- 도 2는 일 실시예에 따른 키-밸류 저장장치의 단위를 설명하기 위한 도면이다.
- 도 3은 일 실시예에 따른 키-밸류 저장 장치에서 블록 필터를 설명하기 위한 예이다.
- 도 4는 일 실시예에 따른 키-밸류 저장장치에서 중복제거 확장 압축 방법을 설명하기 위한 도면이다.

도 5는 일 실시예에 따른 키-밸류 저장장치의 중복제거 검사 방법을 설명하기 위한 흐름도이다.

도 6은 일 실시예에 따른 키-밸류 저장장치의 중복제거 확장 압축 알고리즘을 설명하기 위한 예이다.

발명을 실시하기 위한 구체적인 내용

[0022] 이하, 실시예를 첨부한 도면을 참조하여 상세히 설명한다.

[0024] 실시예에서 제안된 중복제거 확장 압축 방법의 구현은 키-밸류 저장장치(실시예에서는 CaseDB를 예를 들어 설명하기로 함)의 공간 증폭 문제를 제거하는 데 초점을 맞추고 있으며, 혼합 작업부하 애플리케이션에 대해 안정적이고 유연한 키-밸류(KV) 스토어를 제공한다. 이러한 데이터 중복제거 확장 압축 기술은 데이터 중복제거 비용이 임계값에 도달하면 낮은 레벨에서 압축을 수행하여 문제를 해결한다. 게다가 이 방법은 공간 증폭과 쓰기 증폭 사이에서 신뢰할 수 있는 균형을 유지한다. 낮은 레벨에서 압축을 실행하면 빈도가 증가하지 않고 지연 시간이 연장된다. 다시 말해서, 실시예에서 제안된 데이터 중복제거 확장 압축 방법은 데이터 중복제거를 기반으로 한 압축을 실행하지 않고 각 압축 발생 시 데이터 중복제거를 실행한다. 중복제거 검사 프로세스와의 압축을 지연하면 추가 I/O가 발생하지만, 이 프로세스는 중복된 데이터에만 영향을 미치며 처음부터 압축을 실행하는 것보다 비용이 훨씬 적게 든다.

[0025] 도 1은 일 실시예에 따른 키-밸류 저장장치를 설명하기 위한 도면이다.

[0026] LSM-tree를 키-밸류(KV) 스토어로서 구현한 것이 몇 가지 있다. 레벨DB(LevelDB)는 memtable이라 불리는 고정 크기 데이터 단위와 문자열 정렬 테이블(SSTables)로 구성된 다중 레벨 메모리 및 디스크 구성 요소로 구성될 수 있다. 데이터는 메모리의 skip tree 기반 memtable에 먼저 삽입되고 영구 저장소의 문자열 정렬 테이블(SSTable)로 플러시될 수 있다. 압축 프로세스는 여러 번의 압축 후 모든 데이터가 최저 레벨에 도달함에 따라 SSTable을 낮은 레벨로 이동시킬 수 있다. RocksDB는 디스크 레벨에 멀티스레드 키 검색을 적용하여 레벨DB의 읽기 성능을 향상시킨다. 케이스디비(CaseDB)는 키와 값을 공격적으로 분리해 디스크에 새로운 메타데이터 계층을 구현함으로써 쓰기 성능을 향상시키는 데 주력한다. 일례로, 케이스디비(CaseDB)는 CBuffer를 사용한 작은 데이터 쓰기를 피하고 블록 필터를 문자열 정렬 테이블(SSTable)에서 메타데이터 파일로 이동시킬 수 있다. 또한 메타데이터 파일에서 대부분의 압축을 처리하면서 쓰기 증폭을 줄이고 데이터 계층에서 불필요한 파일 입출력(I/O)을 감소시킬 수 있다. 케이스디비(CaseDB)는 키-밸류 스토어 중 LSM-트리의 가장 최신 및 고급 구현 중 하나이다. 도 1과 같이, 레벨DB에서 memtable과 SSTable을 상속받고, 메모리와 디스크 레벨 모두에서 추가 데이터 단위를 사용할 수 있다. 새로 삽입된 키 밸류(KV) 항목(item)은 크기가 임계값보다 클 경우, 먼저 memtable에 직접 삽입될 수 있다. 그렇지 않으면 복수 개의 작은 값을 더 큰 청크(chunk)로 병합하기 위해 써버퍼(CBuffer)에 삽입하고, 그룹화 상태에 대한 메타데이터를 로딩한 후 memtable로 이동할 수 있다. memtable이 가득 차면, 그것은 불변의 memtable (immutable memtable)로 변환될 수 있다. 여기서 데이터는 메모리에서 디스크로 플러시된다. 모든 메모리 구성 요소는 모든 데이터 작업에 일관된 $O(\log n)$ 복잡성을 제공하는 skip 목록을 사용하여 구현될 수 있다. 영구 저장소에서 CaseDB는 도 2(a) 및 도 2(b)와 같이 메타데이터 파일 및 문자열 정렬 테이블(SSTable)에 의해 구성된 메타데이터와 데이터에 대해 2개의 계층을 사용한다.

[0027] 도 2는 일 실시예에 따른 키-밸류 저장장치의 단위를 설명하기 위한 도면이다.

[0028] 영구 저장소에서 CaseDB는 도 2(a) 및 도 2(b)와 같이 메타데이터 파일 및 문자열 정렬 테이블(SSTable)에 의해 구성된 메타데이터와 데이터에 대해 복수 개(예를 들면, 2개)의 계층을 사용할 수 있다. 2 계층 모두 복수 개의 레벨의 LSM tree 구조를 구현할 수 있다. 이러한 LSM tree 구조는 현재의 레벨을 기준으로 각 다음 레벨이 이전 레벨보다 10배 더 커진다. 데이터가 플러시되면 키는 값에서 공격적으로 분리되어 해당 값에 대한 오프셋 참조가 있는 메타데이터 계층에 삽입될 수 있다. 데이터 검색 성능을 향상시키기 위해 블록 필터는 키오프셋 쌍이 있는 메타데이터 파일에 저장될 수 있다. 실제 값은 문자열 정렬 테이블(SSTable) 파일에 기록되고 데이터 계층에 위치할 수 있다. L0이 채워지면 압축이 발생하여 L0과 L1의 메타데이터 파일이 L1로 다시 작성될 수 있다. 이 절차는 각 레벨이 최대 크기에 도달할 때까지 계속되며, 메타데이터 파일은 하위 레벨로 이동될 수 있다. 메타데이터 계층으로 압축이 완료되면, 데이터 계층의 SSTable 파일은 메타데이터 파일 다음에 해당하는 레벨로 재배치될 수 있다. CaseDB는 또한 delete_batch 로그 파일을 사용하여 데이터 계층에서 사라지게 될 키-밸류(KV) 항목을 추적하여 데이터 일관성을 제공할 수 있다. delete_batch 파일은 메타데이터 계층만 관여하는 압축의 첫 번째 파트에서 초기화되며, 문자열 정렬 테이블(SSTable)이 재배치되는 동안 업데이트되고, 삭제된 키-밸류(KV) 항목을 데이터베이스에서 제거하는 데 중요한 역할을 할 수 있다. 메타데이터 압축과 값 재배

치가 지연되면 LevelDB의 가장 취약한 지점이자 대부분의 LSM 트리 계열 KV 스토어인 CaseDB의 쓰기 증폭 문제가 감소할 수 있다.

[0029] CaseDB의 읽기 프로세스는 특정 키에 대한 하향식 데이터 검색을 따른다. 검색된 키는 CBuffer 및 memtable의 메모리 구성 요소에서 먼저 점검될 수 있다. 메모리 구성 요소에서 검색된 키를 찾을 수 없는 경우 메타데이터 계층이 검색될 수 있다. 가장 높은 레벨의 메타데이터 계층에서 시작하여 bloom 필터(Bloom Filter) 체크아웃만 먼저 수행될 수 있다. bloom 필터가 양(positive)을 반환하면 메타데이터 파일이 키를 검색할 수 있다. bloom 필터는 데이터가 집합에 존재하지 않더라도 양의 결과를 반환할 수 있으므로, 메타데이터 파일에서 키를 찾지 못할 수 있으며, 다음 단계로 프로세스가 계속될 수 있다. 일단, 메타데이터 파일에서 키가 발견되면, 키의 값을 보유하는 SSTable을 참조할 수 있다. 실시예에서 CaseDB는 읽기 프로세스 중에 SSTable을 하나만 읽도록 보장하는 것을 가정하기로 한다.

[0030] 도 3은 일 실시예에 따른 키-밸류 저장 장치에서 bloom 필터를 설명하기 위한 예이다.

[0031] bloom 필터는 집합(set)을 읽지 않고 집합 내에서 특정 항목의 존재 여부를 확인하는 데 사용되는 데이터 구조이다. 다른 데이터 구조에 비해 디스크 공간 할당이 적기 때문에 가볍고 적응성이 뛰어난 필터이다. 반환되는 피드백이 음수(negative)라면, 그 항목은 집합에 있지 않는 것을 의미한다. 이러한 장점들을 위해 bloom 필터는 많은 애플리케이션에서 널리 사용된다.

[0032] bloom 필터는 n 비트와 k 해시함수($H_i(k)$)의 조합이다. 도 3을 참고하면, 16비트 및 5개의 해시함수를 갖는 bloom 필터의 예를 도시한 것이다. 도 3(a)는 초기화 상태, 도 3(b)초기화 이후의 상태, 도 3(c)는 긍정 오류(False Positive)를 설명하기 위한 예이다. 모든 비트의 값은 도 3(a)과 같이 처음에는 0으로 초기화될 수 있다. CaseDB에서 데이터를 메모리에서 디스크로 플러시할 때, 키는 값에서 분리되어 해당 오프셋을 가진 메타데이터 파일에 기록될 수 있다. 이 프로세스에서 각 키는 모든 해시함수($H_i(k)$)를 확인하여 bloom 필터 초기화에 참여한다. 도 3(b)는 해시함수를 이용하여 bloom 필터에 복수 개(예를 들면, 2개)의 키를 삽입하는 예를 나타낸 것이다. 각 해시함수($H_i(k)$)는 bloom 필터의 최대 비트 수보다 작은 각 키에 대해 고유한 숫자를 반환할 수 있다. 해시함수($H_i(k)$)에서 생성된 번호는 필터의 인덱스로 간주되며, 해당 값은 1로 초기화될 수 있다. 해시함수($H_i(k)$)가 이전 키에 의해 이미 초기화된 숫자를 생성할 가능성이 있다. 값이 이미 초기화된 경우에도 값을 다시 0으로 변경하지 않고 1을 재할당할 수 있다. 도 3(b)를 참고하면, $H_2(k)$ 는 K_2 에 대해 4를 반환하고, $H_1(k)$ 은 K_1 에 대해 4를 반환하며, 둘 다 집합에서 동일한 비트를 초기화한다. 모든 해시함수를 통해 모든 키를 확인하고 bloom 필터를 초기화할 수 있다.

[0033] CaseDB에서 키를 읽을 때는 메타데이터의 bloom 필터만 먼저 체크할 수 있다. 검색된 키는 동일한 해시함수에서 제공되며, 키에 따라 인덱스를 반환할 수 있다. 모든 해시함수($H_i(k)$)는 동일한 키에 대해 동일한 번호를 반환할 수 있다. 이에 따라, 키를 이미 삽입한 경우 모든 해시함수($H_i(k)$)의 인덱스는 이미 1이어야 한다. 비트 값이 0이면 bloom 필터가 음수로 돌아오는데, 이것은 키가 메타데이터에 존재하지 않는다는 것을 의미한다. 모든 비트가 이미 1인 경우 bloom 필터는 양의 결과를 반환한 다음, 메타데이터 파일을 검색하여 특정 키를 검색할 수 있다. 이렇게 해서 bloom 필터는 전체 파일을 읽지 않고 집합의 데이터 존재에 대한 예측을 획득할 수 있는 가능성을 제공한다. 그러나 bloom 필터는 긍정 오류(false positive)를 겪을 수 있다. 도 3(c)에서 K_3 는 앞서 설명한 것과 같이, 초기화된 동일한 bloom 필터에서 검색될 수 있다. K_1 과 K_2 가 bloom 필터에 삽입된 것으로 알고 있지만, K_3 는 집합에 존재하지 않는다. 경우에 따라 모든 해시함수가 이미 1이 할당된 인덱스를 반환할 수 있다. 모든 값이 1이기 때문에 데이터가 집합에 존재하지 않더라도 bloom 필터는 양의 결과를 반환할 수 있다. 이에 따라 bloom 필터는 양의 결과가 반환 되더라도 집합에 있는 항목의 존재에 대한 신뢰할 수 있는 정보를 제공할 수 없지만, 음의 출력의 경우 데이터가 집합에 있지 않음을 보장한다. 긍정 오류율을 줄이기 위해서는 집합 내 항목 수에 따라 bloom 필터 크기와 해시함수 수를 신중하게 계산해야 한다. 고정 크기 메타데이터 파일을 사용하면 메타데이터의 키 수가 제한되며, CaseDB는 항상 긍정 오류율을 낮은 비율로 유지한다.

[0034] 도 4는 일 실시예에 따른 키-밸류 저장장치에서 중복제거 확장 압축 방법을 설명하기 위한 도면이다.

[0035] 단계(410)에서 키-밸류 저장장치는 키-밸류 저장장치에서 메타데이터 계층으로의 압축 요청이 발생됨을 수신할 수 있다.

[0036] 단계(420)에서 키-밸류 저장장치는 수신된 압축 요청에 기초하여 메타데이터 파일의 압축을 수행함에 따라 중복된 데이터를 제거 하기 위한 중복제거 여부를 체크할 수 있다. 키-밸류 저장장치는 메타데이터의 압축을 실시하는 제1 과정 및 제1 과정을 통해 업데이트된 메타데이터에 따라 문자열 정렬 테이블(SSTable)을 재배치하는

제2 과정을 수행할 수 있다. 키-밸류 저장장치는 제1 과정에서 크기(size) 임계값에 도달하는 레벨에 대해 메타데이터 압축을 지속한 후, 제1 과정 및 제2 과정을 완료함에 따라 메타데이터의 블록 필터를 사용하여 데이터 중복제거 체크를 수행할 수 있다. 이때, 블록 필터는, 메타데이터를 읽지 않고 키밸류 항목(item)의 존재 여부를 확인할 수 있다. 블록 필터를 사용하여 중복된 데이터의 수가 계산될 수 있다.

[0037] 단계(430)에서 키-밸류 저장장치는 중복제거 여부를 체크하여 중복된 데이터를 제거할 수 있다. 키-밸류 저장장치는 메타데이터의 블록 필터를 사용하여 데이터 중복제거 체크를 수행함에 따라 블록 필터로부터 양의 결과 또는 음의 결과가 반환되고, 음의 결과가 반환될 경우, 키가 스킵(skip)되고, 양의 결과가 반환될 경우, 중복 로그(dup_log) 파일에 삽입될 수 있다. 이에, 메타데이터의 위치를 참조하며 복제된 키를 기록하기 위해 중복 로그 파일이 구현될 수 있다. 압축 절차에서 중복 로그 파일 다음에 복제된 키-밸류 항목이 디스크에서 제거되고 공간이 확보될 수 있다. 키-밸류 저장장치는 블록 필터를 사용하여 데이터 중복 체크를 수행함에 따라 메타데이터의 하위 레벨의 중복된 키를 확인한 후, 중복 비율을 계산하고, 계산된 중복 비율의 값을 다음의 중복제거 여부 체크를 위한 임계값을 확인하는데 사용할 수 있다. 키-밸류 저장장치는 임계값을 확인하여 다음의 중복된 데이터를 제거하기 위한 중복제거 여부를 결정하고, 중복비율의 값이 기 설정된 기준 이하일 경우, 메타데이터의 압축을 하위 레벨로 확대하고, 중복된 데이터를 자동으로 제거할 수 있다.

[0038] 도 5는 일 실시예에 따른 키-밸류 저장장치의 중복제거 검사 방법을 설명하기 위한 흐름도이다.

[0039] 키-밸류 저장장치(CaseDB)는 하위 레벨에 대한 중복제거 확장 압축을 수행할 수 있다. 이러한 방법을 통하여 SAR을 감소시키고, SAR과 WAR의 균형을 유지시킬 수 있다. 중복제거 확장 압축 방법의 구현은 키-밸류 저장장치(CaseDB)의 공간 증폭 문제를 제거하는 데 초점을 맞추고 있으며, 혼합 작업부하 애플리케이션에 대해 안정적이고 유연한 키-밸류(KV) 스토어를 제공한다. 이러한 중복제거 확장 압축 기술은 중복제거 비율이 임계값에 도달하면 낮은 레벨에서 압축을 수행하여 문제를 해결할 수 있다. 또한, 공간 증폭과 쓰기 증폭 사이에서 신뢰할 수 있는 균형을 유지할 수 있다. 낮은 레벨에서 압축을 실행하면 빈도가 증가하지 않고 지연 시간이 연장된다. 다시 말해서, 제안된 방법은 중복제거를 기반으로 한 압축을 실행하지 않고 각 압축 발생 시 중복제거를 실행할 수 있다. 중복제거 검사 프로세스와의 압축을 지연하면 추가 I/O가 발생하지만, 이 프로세스는 중복된 데이터에만 영향을 미치며 처음부터 압축을 실행하는 것보다 비용이 훨씬 적게 든다.

[0040] 도 5는 CaseDB 아키텍처의 데이터 중복제거 체크아웃과 함께 확장된 압축을 위한 흐름도와 절차를 보여준다. CaseDB 압축 프로세스는 두 부분으로 구분될 수 있다. 파트 1에서는 메타데이터 압축을 실시(510)하고 파트 2에서는 업데이트된 메타데이터에 따라 문자열 정렬 테이블(SSTable)을 재배치(511)할 수 있다. 파트 1은 크기(size) 임계값에 도달하는 모든 레벨에 대해 지속될 수 있다. 두 파트를 모두 마친 뒤 데이터 중복 체크(520)가 시작될 수 있다. 데이터 중복제거 체크아웃(DedupCheckout()) 방법은 파트 1이 도달하지 못한 모든 하위 레벨만 포함할 수 있다. 데이터 중복제거 체크아웃(DedupCheckout()) 방식에서 블록 필터는 메타데이터를 읽지 않고 키밸류 항목의 존재 여부를 확인하는 데 중요한 역할을 한다. 이 단계에서는 상위 레벨의 업데이트된 각 키를 확인하고 중복된 데이터로 표시할 수 있다. 앞서 설명한 바와 같이, 블록 필터는 양 또는 음의 결과를 반환하며, 피드백이 음일 경우 집합에서 데이터를 사용할 수 없음을 보장한다. 그래서, 음의 결과가 나온 후에 키는 스킵(skip)될 것이고, 그 프로세스는 다음 결과가 반복될 수 있다. 또한 키-밸류 저장장치(CaseDB)에서는 긍정 오류 가능성이 가능한 한 작게 감소하기 때문에 중복제거 체크아웃(DedupCheckout())은 블록 필터의 피드백이 양이 되어 중복 로그(dup_log) 파일에 삽입(530)되면 메타데이터 파일에 키가 존재한다고 가정한다. 집합에 존재하지 않더라도 확장 압축은 다음 단계에서 키를 찾아내고, 이는 데이터 일관성에 영향을 주지 않는다. 중복 체크아웃 프로세스는 블록 필터만 포함함으로써 놀라울 정도로 빠르게 진행된다.

[0041] 식 1:

[0042]
$$DUP_RATIO = TOTAL_KEYS / DUP_KEYS$$

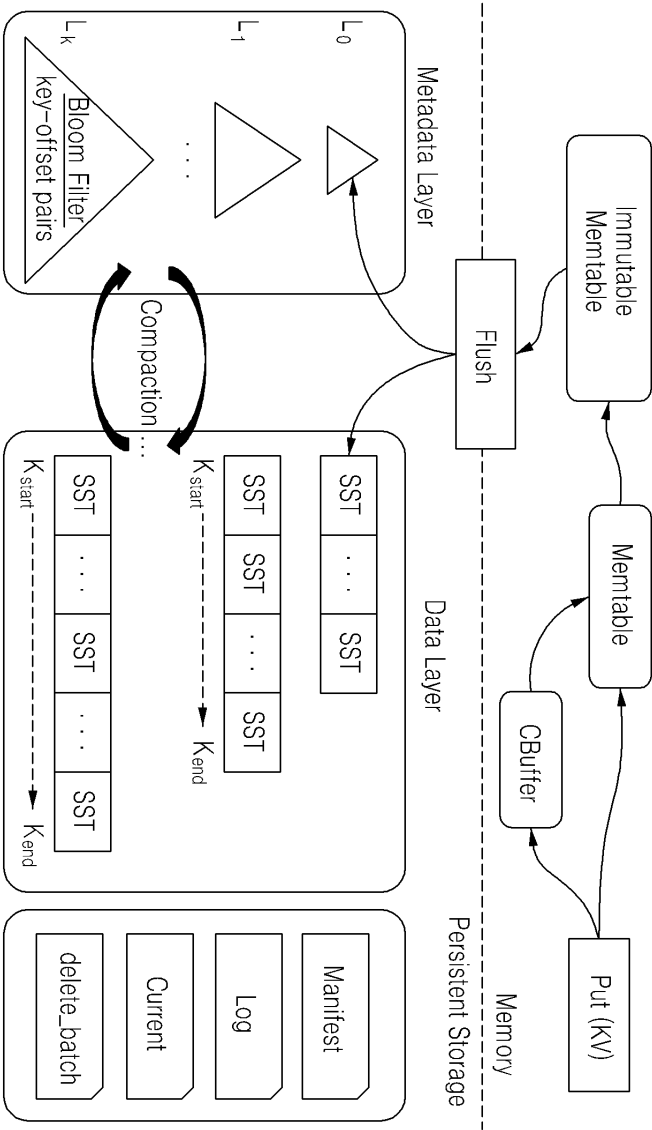
[0043] 모든 하위 레벨의 중복 키를 확인한 후 식 1을 사용하여 중복 비율이 계산될 수 있다. 이 방정식(식 1)은 중복 비율을 계산하는 두 변수에 따라 달라질 수 있다. 하나는 메타데이터 파일에서 쉽게 추출할 수 있는 키의 총 수이고, 다른 하나는 중복된 키의 수로서, 이전 단계에서 초기화 된 중복 로그(dup_log) 파일에서 사용할 수 있다. 계산된 값은 다음 단계에서 중복제거 임계값을 확인하는 데 사용될 수 있다. 이 방정식을 보면 DUP_RATIO(중복 비율)의 값이 높을수록 중복제거 빈도가 높아지고 공간 증폭 측면에서 더 나은 결과를 보장한다는 것을 쉽게 알 수 있다. 또한 전체 키 수에는 압축이 이미 수행된 상위 레벨의 키 수도 포함되며, 중복된 키의 수에는 하위 레벨만 포함되기 때문에 토탈 키(TOTAL_KEYS)는 항상 DUP_KIES보다 높다. 모든 키가 하위 레벨에서 중복되어 있더라도 중복 비율(DUP_RATIO)의 값은 결코 1에 도달하지 못한다. 이에 따라, 더 나은 성능을

위해 더 큰 임계값을 사용하는 것이 좋다.

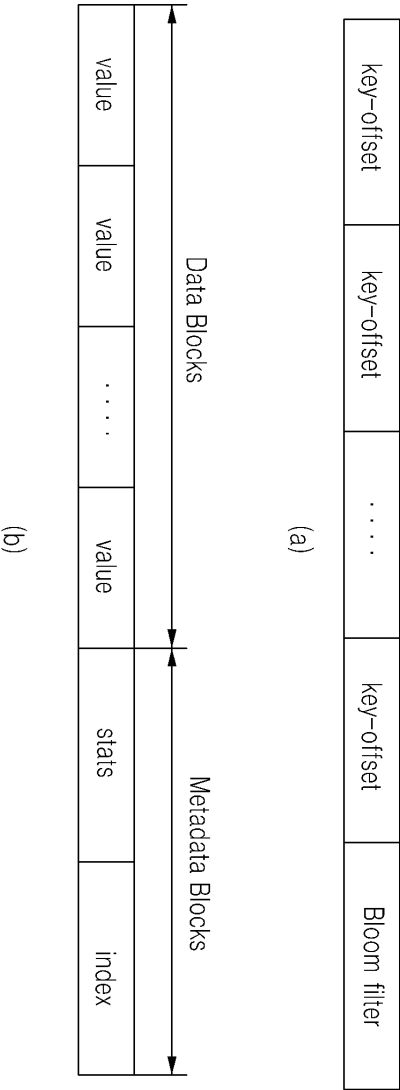
- [0044] 도 6은 일 실시예에 따른 키-밸류 저장장치의 중복제거 확장 압축 알고리즘을 설명하기 위한 예이다. 15행에서는 임계값을 확인하여 중복제거 필요 여부를 결정할 수 있다. DUP_RATIO의 값이 작을 경우, 압축은 모든 하위 레벨로 확대되며, 중복된 데이터는 자동으로 제거될 수 있다. 쓰기 증폭(WAR)과 공간 증폭(SAR)의 균형은 시스템 요구와 작업부하에 따라 중복제거 임계값을 변경하여 관리할 수 있다.
- [0045] 이상에서 설명된 장치는 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPGA(field programmable gate 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(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 기술분야에서 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.
- [0046] 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, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치에 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.
- [0047] 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 가능 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), 및 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.
- [0048] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 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 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.
- [0049] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

도면

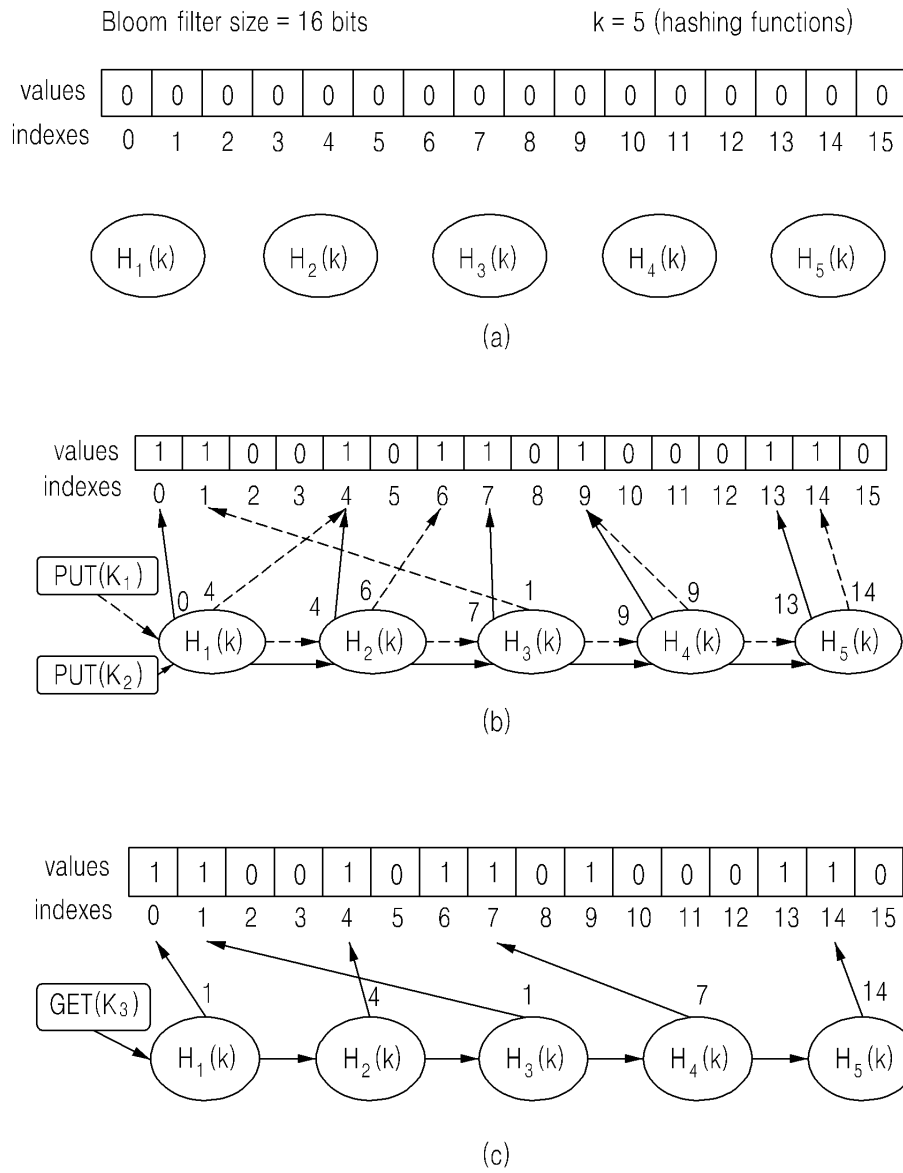
도면1



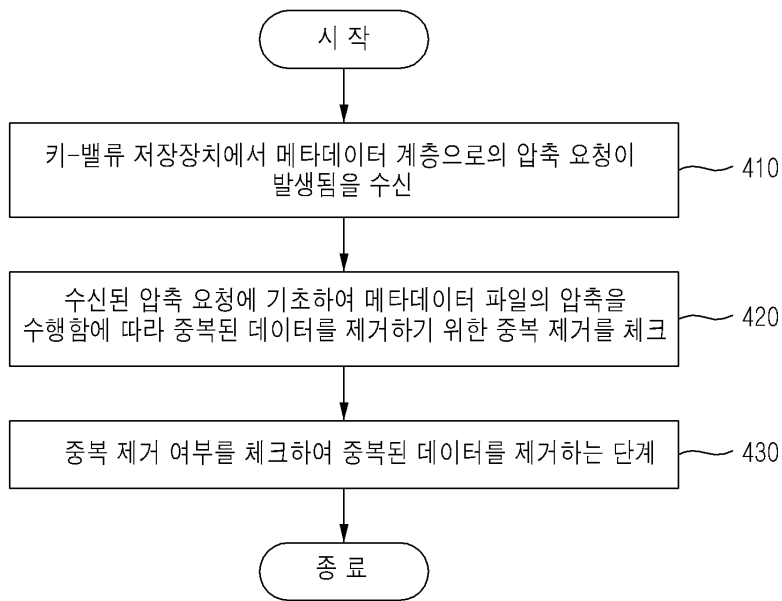
도면2



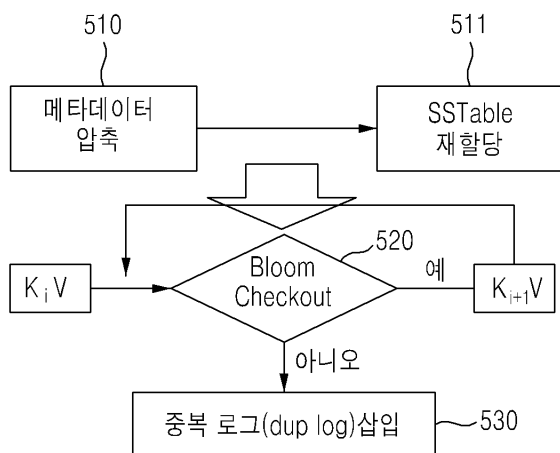
도면3



도면4



도면5



도면6

Algorithm 1 Deduplication Extended Compaction

```

1: function COMPACT()
2:
3: procedure
4:   if NeedsCompaction() then
5:     while NextLevelSize >= THRESHOLD do
6:       SelectKeyRange()
7:       MetadataCompaction()
8:       SSTableRelocation()
9:
10:    //Deduplication starts here
11:    DedupCheckout()
12:    TOTAL_KEYS ← GetTotalKeys()
13:    DUP_KEYS ← GetDupKeys()
14:    DUP_RATIO ← TOTAL_KEYS / EUP_KEYS
15:    if DUP_RATIO <= DEDUP_THRESHOLD() then
16:      while NextLevel ≠ NULL do
17:        SelectKeyRange()
18:        MetadataCompaction()
19:        SSTableRelocaion()
20:    //Deduplication ends here

```
