



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2024-0077233
(43) 공개일자 2024년05월31일

(51) 국제특허분류(Int. Cl.)
G06F 3/06 (2006.01) G06F 12/02 (2018.01)
(52) CPC특허분류
G06F 3/0614 (2013.01)
G06F 12/0292 (2013.01)
(21) 출원번호 10-2022-0159513
(22) 출원일자 2022년11월24일
심사청구일자 없음

(71) 출원인
재단법인대구경북과학기술원
대구 달성군 현풍읍 테크노중앙대로 333
(72) 발명자
이성진
대구광역시 달성군 현풍읍 테크노중앙대로 333
구진형
대구광역시 달성군 유가읍 테크노중앙대로6길 9, 202호
배진욱
경기도 성남시 분당구 정자일로 135 정자동3차푸르지오시티 C동 1311호
(74) 대리인
특허법인지담

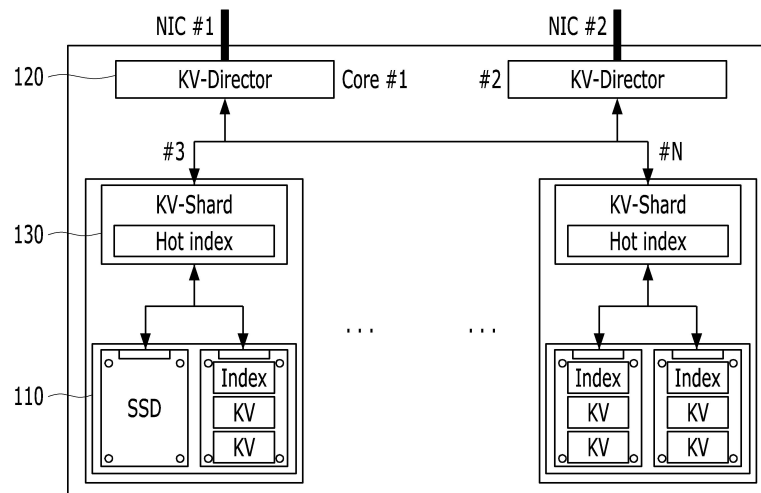
전체 청구항 수 : 총 15 항

(54) 발명의 명칭 샤딩 기반의 키-값 캐싱 시스템 및 방법

(57) 요약

일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템은 복수의 에스에스디들을 포함하는 키-값 저장소, 클라이언트로부터 수신되는 상기 키-값 저장소 내 키-값 쌍들의 캐싱 요청을 처리하는 키-값 디렉터 및 상기 키-값 디렉터가 수신한 상기 캐싱 요청을 전달받아 인덱싱하고 상기 키-값 저장소로부터 상기 키-값 쌍들의 입출력을 처리하는 키-값 샤드를 포함하고, 상기 키-값 샤드는 복수 개로 제공되고 복수의 키-값 샤드들 각각은 상기 복수의 에스에스디들 중 적어도 하나를 관리한다.

대표도 - 도3



(52) CPC특허분류

G06F 2212/1032 (2013.01)

G06F 2212/7207 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711157269
과제번호	2018R1A5A1060031
부처명	과학기술정보통신부
과제관리(전문)기관명	한국연구재단
연구사업명	집단연구지원
연구과제명	암흑데이터 극한활용 연구센터
기 여 율	1/1
과제수행기관명	대구경북과학기술원
연구기간	2022.03.01 ~ 2023.02.28

명세서

청구범위

청구항 1

복수의 에스에스디들을 포함하는 키-값 저장소;

클라이언트로부터 수신되는 상기 키-값 저장소 내 키-값 쌍들의 캐싱 요청을 처리하는 키-값 디렉터; 및

상기 키-값 디렉터가 수신한 상기 캐싱 요청을 전달받아 인덱싱하고 상기 키-값 저장소로부터 상기 키-값 쌍들의 입출력을 처리하는 키-값 샤드를 포함하고,

상기 키-값 샤드는 복수 개로 제공되고 복수의 키-값 샤드들 각각은 상기 복수의 에스에스디들 중 적어도 하나를 관리하는 샤딩 기반의 키-값 캐싱 시스템.

청구항 2

제1항에 있어서, 상기 클라이언트와 통신하여 상기 키-값 디렉터에 상기 캐싱 요청을 전달하는 복수의 인터페이스부들을 더 포함하고,

상기 키-값 디렉터는 복수 개로 제공되어 상기 복수의 인터페이스부들과 일대일 연결되는 샤딩 기반의 키-값 캐싱 시스템.

청구항 3

제1항에 있어서, 상기 복수의 키-값 샤드들 각각은 서로 독립적 구조를 가지고 독립적으로 동작하는 것을 특징으로 하는 샤딩 기반의 키-값 캐싱 시스템.

청구항 4

제3항에 있어서, 상기 키-값 디렉터는 상기 복수의 키-값 샤드들 중 고장난 키-값 샤드를 감지하고, 상기 감지된 고장난 키-값 샤드만을 비활성화시키는 샤딩 기반의 키-값 캐싱 시스템.

청구항 5

제1항에 있어서, 상기 키-값 샤드는 상기 키-값 쌍의 위치 정보를 포함하는 메타데이터를 상기 복수의 에스에스디들에 전부 저장하는 샤딩 기반의 키-값 캐싱 시스템.

청구항 6

제5항에 있어서, 상기 복수의 에스에스디들에는 상기 키-값 쌍 및 상기 키-값 쌍의 속성정보가 저장되는 데이터 영역과 상기 메타데이터가 저장되는 메타데이터 영역이 각각 정의된 샤딩 기반의 키-값 캐싱 시스템.

청구항 7

제6항에 있어서, 상기 키-값 샤드는 상기 메타데이터의 전부 중 단위 시간에 일정 기준 이상 접근되는 일부 메타데이터들을 서버의 DRAM에 캐싱하는 샤딩 기반의 키-값 캐싱 시스템.

청구항 8

제7항에 있어서, 상기 키-값 샤드는 상기 키-값 쌍을 인덱싱하는 과정에서 서로 다른 2개의 키-값 쌍이 동일한 하나의 해시 엔트리를 가지는 해시 충돌을 무시하고,

상기 서로 다른 2개의 키-값 쌍들의 상기 동일한 하나의 해시 엔트리 중 기존의 해시 엔트리를 가장 최근의 해시 엔트리로 대체하는 샤딩 기반의 키-값 캐싱 시스템.

청구항 9

제1항에 있어서, 상기 키-값 저장소는 상기 키-값 쌍의 속성정보에 포함된 만료시간 정보가 유사한 키-값 쌍들

끼리 묶어서 저장하는 샤딩 기반의 키-값 캐싱 시스템.

청구항 10

샤딩 기반의 키-값 캐싱 시스템이 키-값 디렉터를 통해 클라이언트로부터 키-값 저장소 내 키-값 쌍들에 대한 캐싱 요청을 수신하고 키-값 샤드로 전달하는 단계; 및

샤딩 기반의 키-값 캐싱 시스템이 상기 키-값 샤드를 통해 상기 캐싱 요청을 인덱싱하고 상기 키-값 저장소로부터 상기 키-값 쌍들의 입출력을 처리하는 단계를 포함하고,

상기 키-값 샤드는 복수 개로 제공되고, 독립적인 구조를 가지고 독립적으로 동작하는 상기 복수의 키-값 샤드들 각각은 상기 복수의 에스에스디들 중 적어도 하나를 관리하는 샤딩 기반의 키-값 캐싱 방법.

청구항 11

제10항에 있어서, 상기 키-값 쌍들에 대한 캐싱 요청을 수신하고 키-값 샤드로 전달하는 단계는,

상기 키-값 디렉터가 상기 복수의 키-값 샤드들 중 고장난 키-값 샤드를 감지하고, 상기 감지된 고장난 키-값 샤드만을 비활성화시키는 단계를 포함하는 샤딩 기반의 키-값 캐싱 방법.

청구항 12

제10항에 있어서, 상기 복수의 에스에스디들에는 상기 키-값 쌍 및 상기 키-값 쌍의 속성정보가 저장되는 데이터 영역과 상기 키-값 쌍의 위치 정보를 포함하는 메타데이터가 저장되는 메타데이터 영역이 각각 정의되고,

상기 키-값 샤드는 상기 메타데이터를 상기 복수의 에스에스디들의 상기 메타데이터 영역에 전부 저장하는 것을 특징으로 하는 샤딩 기반의 키-값 캐싱 방법.

청구항 13

제12항에 있어서, 상기 키-값 샤드가 상기 키-값 쌍들의 입출력을 처리하는 단계는,

상기 메타데이터의 전부 중 단위 시간 내에 일정 기준 이상 접근되는 일부 메타데이터들을 서버의 DRAM에 캐싱하는 단계를 포함하는 샤딩 기반의 키-값 캐싱 방법.

청구항 14

제10항에 있어서, 상기 키-값 샤드가 상기 키-값 쌍들의 입출력을 처리하는 단계는,

상기 키-값 쌍을 인덱싱하는 과정에서 서로 다른 2개의 키-값 쌍이 동일한 하나의 해시 엔트리를 가지는 해시 충돌을 무시하고,

상기 서로 다른 2개의 키-값 쌍의 상기 동일한 하나의 해시 엔트리 중 기존의 해시 엔트리를 가장 최근의 해시 엔트리로 대체하는 단계를 포함하는 샤딩 기반의 키-값 캐싱 방법.

청구항 15

제10항에 있어서, 상기 샤딩 기반의 키-값 캐싱 시스템이 상기 캐싱 요청을 인덱싱하고 상기 키-값 저장소로부터 상기 키-값 쌍들의 입출력을 처리하는 단계는,

제1 키-값 샤드에 상기 캐싱 요청을 인덱싱하고 상기 제1 키-값 샤드의 고장 여부를 확인하여, 상기 제1 키-값 샤드가 정상인 경우 상기 제1 키-값 샤드에서 상기 캐싱 요청을 처리하고, 상기 제1 키-값 샤드가 고장인 경우, 정상 상태인 제2 키-값 샤드로 상기 캐싱 요청을 리디렉션하여 상기 제2 키-값 샤드가 상기 캐싱 요청을 처리하며,

이 후 상기 제1 키-값 샤드가 복구되면, 상기 리디렉션된 캐싱 요청을 복구된 상기 제1 키-값 샤드로 마이그레이션하여 상기 복구된 제1 키-값 샤드가 상기 캐싱 요청을 처리하는 것을 특징으로 하는 샤딩 기반의 키-값 캐싱 방법.

발명의 설명

기술 분야

[0001] 본 발명은 샤딩 기반의 키-값 캐싱 시스템 및 방법에 관한 것으로, 보다 상세하게는 대용량의 키-값 저장 시스템에서 대량의 키-값 쌍들을 캐싱하는 샤딩 기반의 키-값 캐싱 시스템 및 방법에 관한 것이다.

배경 기술

[0002] 올-플래시 어레이 (All-flash array, AFA) 서버는 수십 개의 낸드 플래시 기반 저장장치 (예, SSD)를 장착할 수 있다. 기존 AFA 기반 캐싱 시스템에서, 다수의 SSD는 RAID와 같은 기존 저장장치 통합 시스템을 통해 하나의 저장장치로 가상화 된다. 거대한 용량을 가지는 가상화된 저장장치에 데이터를 저장하게 되는데, 이 때 저장장치 내 키-값 쌍의 위치를 나타내는 메타데이터가 필요하다. 올-플래시 어레이 서버처럼 매우 거대한 용량을 지원하는 시스템에서는 매우 많은 키-값 쌍이 저장될 수 있다. 이 때 크게 세가지 문제점이 발생된다. 먼저 메타데이터의 크기가 매우 크다. 일반적으로 메타데이터는 서버 DRAM에 모두 적재되는데, AFA 서버 환경에서는 저장장치에 저장된 키-값 쌍이 매우 많으므로 모든 메타데이터를 DRAM에 적재할 수 없다. 따라서 기존 AFA 기반 시스템은 큰 메타데이터로 인해 일부 메타데이터만 DRAM에 적재할 수 있는 데, 저장장치에 존재하는 메타데이터를 접근할 경우 매우 낮은 성능을 보인다. 또한, 만료된 객체를 빠르게 소거하지 못할 경우 캐시 시스템의 유효 용량이 감소하여 낮은 캐시 적중률을 보인다. 마지막으로 다수의 저장장치를 사용하기 위한 기존 RAID 방식은 패리티 블록의 성능 부하와 용량 감소의 단점을 가진다.

발명의 내용

해결하려는 과제

[0003] 본 발명이 이루고자 하는 기술적 과제는 상기된 문제점들을 해소하는 것이다.

[0004] 본 발명의 일 실시예는 해시 충돌 무시 2단계 해시 테이블을 사용하여 메타데이터를 DRAM과 저장장치에 나누어 관리하며 종래 기술대비 낮은 입출력 부하로 키-값 캐싱 요청을 처리하는 샤딩 기반의 키-값 캐싱 시스템 및 방법을 제공하는 것을 목적으로 한다.

[0005] 본 발명의 일 실시예는 오브젝트들을 만료시간에 따라 배치여 만료된 오브젝트들을 효율적으로 선제적으로 제거함에 따라서 높은 캐시 적중률을 달성하는 샤딩 기반의 키-값 캐싱 시스템 및 방법을 제공하는 것을 목적으로 한다.

[0006] 본 발명의 일 실시예는 패리티 블록 등을 사용하지 않고 데이터를 기록하여 패리티 블록을 위한 입출력과 용량이 필요하지 않아 높은 성능 확장성을 보이는 샤딩 기반의 키-값 캐싱 시스템 및 방법을 제공하는 것을 목적으로 한다.

[0007] 본 발명이 이루고자 하는 기술적 과제는 이상에서 언급한 기술적 과제로 제한되지 않으며, 언급되지 않은 또 다른 기술적 과제들은 아래의 기재로부터 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0008] 상기 기술적 과제를 달성하기 위하여, 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템은 복수의 에스에스디들을 포함하는 키-값 저장소, 클라이언트로부터 수신되는 상기 키-값 저장소 내 키-값 쌍들의 캐싱 요청을 처리하는 키-값 디렉터 및 상기 키-값 디렉터가 수신한 상기 캐싱 요청을 전달받아 인덱싱하고 상기 키-값 저장소로부터 상기 키-값 쌍들의 입출력을 처리하는 키-값 샤드를 포함하고, 상기 키-값 샤드는 복수 개로 제공되고 복수의 키-값 샤드들 각각은 상기 복수의 에스에스디들 중 적어도 하나를 관리한다.

[0009] 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 방법은 샤딩 기반의 키-값 캐싱 시스템이 키-값 디렉터를 통해 클라이언트로부터 키-값 저장소 내 키-값 쌍들에 대한 캐싱 요청을 수신하고 키-값 샤드로 전달하는 단계 및 샤딩 기반의 키-값 캐싱 시스템이 상기 키-값 샤드를 통해 상기 캐싱 요청을 인덱싱하고 상기 키-값 저장소로부터 상기 키-값 쌍들의 입출력을 처리하는 단계를 포함하고, 상기 키-값 샤드는 복수 개로 제공되고, 독립적인 구조를 가지고 독립적으로 동작하는 상기 복수의 키-값 샤드들 각각은 상기 복수의 에스에스디들 중 적어도 하나를 관리한다.

발명의 효과

- [0010] 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템 및 방법은 해시 충돌 무시 2단계 해시 테이블을 사용하여 메타데이터를 DRAM과 저장장치에 나누어 관리하며 종래 기술대비 낮은 입출력 부하로 키-값 캐싱 요청을 처리할 수 있다.
- [0011] 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템 및 방법은 오브젝트들을 만료시간에 따라 배치여 만료된 오브젝트들을 효율적으로 선제적으로 제거함에 따라서 높은 캐시 적중률을 달성할 수 있다.
- [0012] 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템 및 방법은 패리티 블록 등을 사용하지 않고 데이터를 기록하여 패리티 블록을 위한 입출력과 용량이 필요하지 않아 높은 성능 확장성을 가질 수 있다.
- [0013] 본 발명의 효과는 상기한 효과로 한정되는 것은 아니며, 본 발명의 설명 또는 청구범위에 기재된 발명의 구성으로부터 추론 가능한 모든 효과를 포함하는 것으로 이해되어야 한다.

도면의 간단한 설명

- [0014] 도 1은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템을 보여주는 도면이다.
- 도 2는 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 블록도이다.
- 도 3은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템을 보여주는 예시도이다.
- 도 4는 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 해시 테이블을 보여주는 예시도이다.
- 도 5는 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 동작 과정을 보여주는 예시도이다.
- 도 6은 일 실시예에 따른 캐싱 시스템의 만료 객체의 즉각적 소거 방법을 보여주는 도면이다.
- 도 7은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 방법을 보여주는 흐름도이다.
- 도 8 내지 도 11은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 효과를 보여주는 그래프들이다.

발명을 실시하기 위한 구체적인 내용

- [0015] 이하에서는 첨부한 도면을 참조하여 본 발명을 설명하기로 한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며, 따라서 여기에서 설명하는 실시예로 한정되는 것은 아니다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.
- [0016] 명세서 전체에서, 어떤 부분이 다른 부분과 "연결(접속, 접촉, 결합)"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 부재를 사이에 두고 "간접적으로 연결"되어 있는 경우도 포함한다. 또한 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 구비할 수 있다는 것을 의미한다.
- [0017] 본 명세서에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 명세서상에 기재된 특징, 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.
- [0018] 본 명세서에서, "모듈"은 하드웨어, 소프트웨어 또는 펌웨어로 구성된 유닛을 포함하며, 예컨대 로직, 논리 블록, 부품, 또는 회로 등의 용어와 상호 호환적으로 사용될 수 있다. 모듈은 일체로 구성된 부품 또는 하나 또는 그 이상의 기능을 수행하는 최소 단위 또는 그 일부가 될 수 있다. 예컨대 모듈은 ASIC(application-specific integrated circuit)으로 구성될 수 있다.
- [0019] 이하, 첨부된 도면을 참고하여 본 발명의 실시예를 상세히 설명하기로 한다.
- [0020] 도 1은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템을 보여주는 도면이다.
- [0021] 도 1을 참조하면, 샤딩 기반의 키-값 캐싱 시스템(100)은 데이터베이스(200) 및 서버(300)와 네트워크를 통해 연결될 수 있다. 샤딩 기반의 키-값 캐싱 시스템(100)은 액세스 속도가 느린 백엔드의 데이터베이스(200)로부터

달리 획득되어야 할 정보를 바르게 액세스할 수 있다.

- [0022] 키-값 캐시(key-value cache)가 없다면, 웹 서버는 모든 읽기들과 쓰기들을 위해 서버(300) 및 데이터베이스(200)로 직접 액세스해야 한다. 데이터 베이스 서버(106)의 부하(load)를 줄이기 위해서, 키-값 저장소(key-value store) 기반의 램(랜덤 액세스 메모리)(RAM: Random Access Memory) 또는 플래시(Flash)가 데이터베이스(200)의 로드를 감소시키고 자주 사용되는 데이터를 위한 캐시(cache)로서 동작하기 위하여 제공될 수 있다. 웹 서버는 키-값 저장소를 포함하는 샤딩 기반의 키-값 캐싱 시스템(100)에 먼저 질의(query)할 수 있고, 키-값 저장소를 포함하는 샤딩 기반의 키-값 캐싱 시스템(100)이 요청된 데이터를 포함하지 않는 경우에 한하여 데이터베이스(200)로 액세스한다. 따라서, 데이터베이스(200) 및 서버(300)의 부하(load)가 상당히 감소될 수 있다.
- [0023] 일 실시예에서, 샤딩 기반의 키-값 캐싱 시스템(100) 및 데이터베이스(200)는 서버(300)에 포함될 수 있다. 즉, 서버(300)는 서버 내부에 샤딩 기반의 키-값 캐싱 시스템(100) 및 데이터베이스(200)를 포함할 수 있다. 서버(300)는 프로세서 및 메모리를 포함할 수 있다. 프로세서는 메모리와 연결될 수 있다. 메모리는 랜덤 액세스 메모리(RAM)를 포함할 수 있다. 메모리는 DRAM을 포함할 수 있다.
- [0024] 예를 들어, 서버(300)는 올-플래시 어레이(All-flash array, AFA) 서버일 수 있다. 서버는 복수 개의 낸드 플래시 기반 저장장치로서 예를 들어 에스에스디(Solid State Drive, SSD)를 포함할 수 있다. 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템(100)은 AFA(All-Flash Array)에서 대형 객체를 캐싱하도록 특별히 설계된 키-값 캐싱 시스템으로 제공될 수 있다.
- [0025] 본 명세서에서 이하 샤딩 기반의 키-값 캐싱 시스템(100)은 캐싱 시스템(BigKV)으로 지칭될 수 있다.
- [0026] 캐싱 시스템(BigKV)은 캐시(cache)의 고유한 속성을 중심으로 설계될 수 있다. 캐시는 데이터 복사본을 포함하기 때문에 캐시에 있는 항목의 정확한 기재 여부는 원본과 비교할 때 크리티컬하지 않다. 일 실시예에서, 캐싱 시스템(BigKV)은 해시 충돌을 무시하고 메타데이터 정보를 근사화하며 오류로 인한 데이터 손실을 허용함으로써 캐시 적중률을 크게 높이고 시스템에 더 유용한 객체를 유지할 수 있다.
- [0027] 도 2는 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 블록도이다.
- [0028] 도 2를 참조하면, 샤딩 기반의 키-값 캐싱 시스템(100)은 키-값 저장소(110), 키-값 디렉터(120), 키-값 샤드(130) 및 인터페이스부(140)를 포함할 수 있다.
- [0029] 키-값 저장소(110)는 복수의 에스에스디들(SSD)을 포함할 수 있다. 에스에스디(SSD)는 SSD로 지칭될 수 있다.
- [0030] 키-값 디렉터(120)는 클라이언트로부터 수신되는 키-값 저장소 내 키-값 쌍들의 캐싱 요청을 처리할 수 있다. 키-값 디렉터(120)는 복수 개로 제공되어 복수의 인터페이스부(140)들과 일대일로 연결될 수 있다.
- [0031] 키-값 샤드(130)는 키-값 디렉터가 수신한 캐싱 요청을 전달받아 인덱싱하고 키-값 저장소로부터 키-값 쌍들의 입출력을 처리할 수 있다. 일 실시예에서, 키-값 샤드(130)는 복수 개로 제공되고 복수의 키-값 샤드들(130) 각각은 상기 복수의 에스에스디들(SSD) 중 적어도 하나를 관리할 수 있다.
- [0032] 인터페이스부(140)는 클라이언트와 통신하여 키-값 디렉터에 캐싱 요청을 전달할 수 있다. 인터페이스부(140)는 NIC(Network interface card)를 포함할 수 있다.
- [0033] 도 3은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템을 보여주는 예시도이다. 도 2를 참조하여 이하 설명한다.
- [0034] 도 3에서, 최상단에는 클라이언트로부터 들어오는 키-값 쌍들에 대한 캐싱 요청을 처리하는 키-값 디렉터(120, KV-Director)가 배치될 수 있다. 키-값 디렉터(120)는 복수 개로 제공될 수 있다. 복수의 키-값 디렉터들(120) 각각은 인터페이스부(140)인 NIC를 하나씩 맡아 요청을 처리하며 키-값 캐싱 요청을 하단의 키-값 샤드(130, KV-Shard)에 전달할 수 있다. 키-값 디렉터(120)는 키-값 캐싱 요청의 키(key)를 이용하여 해시 처리하며 그 결과 해시값을 사용하여 어떤 키-값 샤드(130)에 캐싱 요청을 전달할지 결정할 수 있다. 키-값 디렉터(120) 각각은 하나의 코어에 고정되어 역할을 수행할 수 있다. 캐싱 시스템(BigKV)은 강력하지만 효율적인 128비트 해시 함수를 사용하여 키-값 샤드(130)에 객체(object)를 고르게 배포할 수 있다. 키-값 디렉터(120)는 또한 실패한 (또는 고장난) 키-값 샤드(130)를 감지 및 비활성화하고 이하 설명되는 반응형 내결함성 메커니즘을 통해 고장난 키-값 샤드(130)로부터 객체를 복구할 책임을 가진다.
- [0035] 키-값 샤드(130, KV-Shard)는 전달받은 키-값 쌍들에 대한 캐싱 요청을 인덱싱하고 에스에스디(SSD)를 포함하는 키-값 저장소(110)에 입출력을 담당하는 처리기일 수 있다. 키-값 샤드(130)는 복수 개로 제공될 수 있다. 키-

값 샤드들(130) 각각은 하나 혹은 몇 개의 에스에스디(SSD)를 관리하며 로그 구조(log-structured) 방식으로 데이터 및 메타데이터를 에스에스디(SSD)에 기록할 수 있다. 키-값 샤드(130)는 메타데이터의 전체를 SSD에 저장하며 일부 자주 접근되는 메타데이터를 서버 DRAM에 캐싱할 수 있다. 키-값 샤드(130) 각각은 독립적인 구조를 가지고 각기 다른 CPU 코어에서 독립적으로 동작하며, 다른 키-값 샤드(130)의 간섭 없이 하단의 에스에스디(SSD)의 성능을 최대한 활용할 수 있다. 키-값 샤드(130)는 2단계 해시 테이블을 유지 관리하고 키-값 디렉터(120)에서 전달된 캐싱 요청을 처리하며 기본적으로 SSD를 관리할 수 있다. 키-값 샤드(130)는 그것에 속한 SSD를 로그로 추상화할 수 있다.

[0036] 하나의 키-값 샤드(130)에 둘 이상의 SSD가 할당된 경우 RAID-0을 사용하여 통합할 수 있다. 키-값 샤드(130)는 객체 및 메타데이터에 대한 모든 변경 사항을 로그에 추가한다. 키-값 샤드(130)는 만료시간(time to live, TTL)을 고려하여 사용되지 않고 만료된 객체가 차지하는 공간을 회수하기 위해 정기적으로 청소를 수행할 수 있다.

[0037] 도 3의 캐싱 시스템(BigKV)은 새로운 올 플래시 어레이(AFA) 규모의 키-값 캐싱 시스템일 수 있다. 예를 들어, 캐싱 시스템(BigKV)은 최소한의 DRAM 사용으로 대규모 AFA에서 수십억 개의 대형 객체(object)를 인덱싱하고 효율적인 공간 관리를 통해 캐시 적중률을 극대화하도록 설계될 수 있다.

[0038] 캐싱 시스템(BigKV)은 이하의 충돌 무시하는 2단계 해시 테이블을 이용한 객체 인덱싱, 만료된 객체의 즉각적 소거, 및 반응형 내결함성 메커니즘을 포함하는 세 가지 핵심 기술을 제안할 수 있다.

[0039] 첫째로, 캐싱 시스템(BigKV)은 충돌을 무시하는 2단계 해시 테이블을 이용한 객체 인덱싱 방법을 제안할 수 있다.

[0040] 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템(100)은 연결 및 개방 주소 지정을 활용하고 단점을 극복하는 2단계 해시 테이블을 사용하여, 메타데이터를 DRAM과 저장장치에 나누어 관리하며 종래 기술대비 낮은 입출력 부하로 키-값 캐싱 요청을 처리할 수 있다.

[0041] 일 실시예에서, 캐싱 시스템(BigKV)에는 객체당 16B 메타데이터가 있을 수 있다. 캐싱 시스템(BigKV)은 DRAM 상 주 세트 연관 캐시에 핫 메타데이터만 유지할 수 있다. 이를 통해 캐싱 시스템(BigKV)은 포인터 오버헤드 없이 높은 적중률을 달성할 수 있다. 캐싱 시스템(BigKV)의 전체 메타데이터는 분할되어 플래시의 작은(16KB) 해시 테이블에 저장되며, 각 해시 테이블은 개방형 주소 지정을 사용하여 구축된다. 캐시 미스 시 캐싱 시스템(BigKV)은 16KB 해시 테이블을 읽고 이에 대해 해시 프로빙을 수행할 수 있다. 프로브 거리는 테이블 경계 내에서 제한될 수 있다. 따라서 캐시 미스 시 최대 하나의 추가 I/O가 필요하다. 캐싱 시스템(BigKV)은 지문(finger print) 충돌을 무시하므로 충돌한 객체가 손실될 수 있다. 대신, 캐싱 시스템(BigKV)은 충돌률을 극도로 낮춤(108)으로서 부정적인 영향을 최소화한다. 결과적으로, 캐싱 시스템(BigKV)은 쓰기 경로에서 충돌 해결 단계를 제거하여 쓰기 처리량을 향상시킬 수 있다.

[0042] 둘째로, 캐싱 시스템(BigKV)은 만료된 객체의 즉각적 소거 방법을 제안할 수 있다. 즉, 캐싱 시스템(BigKV)은 데이터를 SSD에 저장할 때 오브젝트의 만료 시간이 비슷한 것들끼리 묶어 저장함으로써, 만료되는 오브젝트들을 효과적으로 소거할 수 있다. 따라서, 캐싱 시스템(BigKV)은 같은 용량을 사용하는 타 종래기술에 비해 더 큰 유효 용량을 제공하여 높은 캐시 적중률을 달성할 수 있다.

[0043] 캐싱 시스템(BigKV)은 만료된 객체를 빠르게 제거할 수 있다. 캐싱 시스템(BigKV)은 5비트를 사용하여 객체의 만료 시간(TTL)을 근사화한다. 이는 작지만 만료된 객체를 정확하게 식별하기에 충분하다. 효율적인 공간 재확보를 위해 키-값 쌍의 객체를 작성할 때, 캐싱 시스템(BigKV)은 동일한 할당 그룹에 유사한 TTL을 가진 객체를 할당하여 함께 추출 및 제거되도록 할 수 있다. 이러한 큰 할당 그룹을 추적하는데 메모리가 거의 필요하지 않다.

[0044] 마지막으로, 캐싱 시스템(BigKV)은 반응형 내결함성 메커니즘을 제안할 수 있다. 캐싱 시스템(BigKV)은 RAID의 낮은 성능 확장성 한계를 극복하기 위해 샤딩(sharding) 기반의 구조를 띄고 있다. 캐싱 시스템(BigKV)은 패리티 블록 등을 두지 않고 데이터를 저장하므로 패리티 블록에 따른 연산 및 입출력 부하가 존재하지 않는다. 캐싱 시스템(BigKV)에서 저장장치 고장 시 해당 SSD에 저장된 데이터를 전부 잃어버린다는 단점이 있지만, 캐시 시스템의 특성 상 원본 데이터는 데이터베이스 서버에 이미 기록되어 있으므로 데이터 일관성에 문제가 되지 않는다. 캐싱 시스템(BigKV)은 오히려 높은 SSD의 성능을 최대한 활용할 수 있게 한다.

[0045] 일 실시예에 따른 캐싱 시스템(BigKV)은 저장장치 또는 에스에스디(SSD)의 고장 시 고장난 에스에스디를 관리하는 키-값 샤드(130)를 중지시키고 새로운 에스에스디가 삽입될 때까지의 시간 동안 고장나지 않은 남은 키-값

샤드들(130)이 키-값 캐싱 요청을 처리하도록 할 수 있다. 새로운 에스에스디 또는 저장장치가 모두 성공적으로 삽입되면, 삽입된 에스에스디를 가진 키-값 샤드가 다시 활성화 되어 키-값 캐싱 요청을 처리할 수 있다. 캐싱 시스템(BigKV)은 이러한 방식을 통해 높은 SSD의 성능을 효과적으로 활용할 수 있다.

[0046] 일 실시예에서, 캐싱 시스템(BigKV)은 샤딩(sharding) 기반으로 설계되어 패리티 기반 기술을 사용하지 않지만 반응형 내결함성 메커니즘을 통해 우수한 내결함성을 제공할 수 있다. 캐싱 시스템(BigKV)은 키-값 샤드(130) 내에서 하나 또는 몇 개의 SSD를 분리할 수 있다. 캐싱 시스템(BigKV)은 장애 또는 결함이 발생한 SSD가 감지되면, 장애가 발생한 SSD와 연결된 키-값 샤드(130)를 비활성화하고 연결된 객체(또는 키-값 객체)를 삭제할 수 있다. 개별 키-값 샤드(130)는 독립적으로 작동하기 때문에, 문제가 있는 키-값 샤드(130)를 작동 중지시키더라도(고장난 SSD가 복구될 때까지) 정상적인 키-값 샤드들(130)에는 영향을 미치지 않는다. 따라서, SSD 고장 시 캐시 용량을 줄일 수 있다.

[0047] 도 4는 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 해시 테이블을 보여주는 예시도이다. 도 5는 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 동작 과정을 보여주는 예시도이다.

[0048] 도 4는 메타데이터를 사용하여 객체를 인덱싱하기 위한 2단계 해시 테이블을 보여준다. 도 4는 캐싱 시스템(BigKV)이 인메모리 및 인플래시 데이터 구조를 통해 객체를 인덱싱하고 LOOKUP 및 SET 요청을 처리하는 방법을 보여준다.

[0049] 일 실시예에서, 캐싱 시스템(BigKV)은 자주 접근되는 메타데이터(핫 메타데이터)를 DRAM 내 저장할 수 있다. 예를 들어, 캐싱 시스템(BigKV)은 단위 시간에 일정 기준 이상의 횟수로 접근되는 일부 메타데이터들을 DRAM 내에 저장할 수 있다. 단위 시간 및 일정 기준은 미리 설정될 수 있다.

[0050] 캐싱 시스템(BigKV)에서 우선적으로 모든 메타데이터는 SSD에 저장된다. SSD에 저장되는 해시 테이블은 낸드 플래시의 페이지 크기(16KB)로 쪼개어져 존재할 수 있다. 쪼개진 각각의 해시 테이블(HTable)은 세트 연관(Set-associative) 캐시 방식으로 DRAM에 캐싱될 수 있다(SetCache). 캐싱 시스템(BigKV)은 이러한 구조를 통해 DRAM에 존재하지 않는 메타데이터를 찾을 때 한 번의 SSD 입출력을 통해 메타데이터를 찾을 수 있기 때문에 효율적이다.

[0051] 도 4는 2-레벨 메타데이터의 구성을 예시한다. 캐싱 시스템(BigKV)은 플래시 공간을 데이터 영역(111)과 메타데이터 영역(112)의 두 영역으로 나눈다. 속성정보(예를 들어, 전체 길이의 키(key) 및 정확한 만료시간(TTL)를 포함하는 키-값 객체 및 해당 헤더는 데이터 영역(111)에 저장될 수 있다.

[0052] 캐싱 시스템(BigKV)은 고정 크기 단위(기본적으로 4KB)를 사용하여 키-값 객체에 대한 공간을 할당할 수 있다. 이 고정 크기 할당은 내부 단편화로 이어지지만 4KB는 큰 객체를 수용하기에 충분히 작기 때문에 부정적인 영향은 미미할 수 있다.

[0053] 캐싱 시스템(BigKV)은 메타데이터 영역(112)에 키-값 객체를 인덱싱하는 플래시 상주 메타데이터를 포함할 수 있다. SSD에 저장되는 해시 테이블은 복수 개의 해시 테이블(HTables)로 분할되며, 각 해시 테이블은 선형 프로빙 방식으로 해싱을 개방하여 관리할 수 있다. 해시 테이블 각각의 크기는 16KB로 일반적인 NAND 페이지 크기와 일치할 수 있다. 각 해시 테이블 항목에는 해당 메타데이터가 가리키는 키-값 쌍의 객체와 연결된 객체 별 메타데이터가 16B 있을 수 있다. 객체 별 메타데이터에는 키-값 객체의 64비트 지문, 데이터 영역의 객체에 대한 38비트 포인터, 18비트 객체 크기 및 기타 플래그(8비트)와 같은 필수 정보가 들어 있다. 16B 테이블 헤더를 포함하여 각 해시 테이블은 1,023개의 해시 항목을 보유할 수 있다.

[0054] 캐싱 시스템(BigKV)은 DRAM에 있는 핫 객체의 핫 메타데이터를 보유하는 해시 항목을 캐싱할 수 있다. 키-값 객체(데이터 및 헤더)는 DRAM에 캐시되지 않는다. 몇 개의 큰 객체보다 DRAM에 많은 작은 해시 항목을 유지하는 것이 일관된 객체 액세스를 제공하는 데 더 좋다. DRAM 상주 메타데이터는 b버킷으로 나누어진 세트 연관 캐시로 구성될 수 있다. 각 버킷은 64B CPU 캐시 라인에 맞는 4개의 16B 해시 항목이 있다. k연속 버킷 그룹은 해시 테이블(HTable) 전용인 세트 캐시(SetCache)를 형성할 수 있다. 즉, 각 해시 테이블에는 k버킷이 있는 자체 세트 캐시(SetCache)가 있다. 각 세트 캐시(SetCache)에는 최신 해시 테이블(HTable)을 추적하기 위한 포인터가 있으며 나중에 메타데이터 정리에 사용될 수 있다.

[0055] 캐싱 시스템(BigKV)은 처음에 (가장 작은) 4KB 객체인 경우에도 모든 키-값 객체를 인덱싱하기 위해 많은 수의 해시 테이블(HTable)을 만들 수 있다.

[0056] 해시 테이블(HTable)은 전체 올 플래시 어레이(AFA) 공간의 0.4%를 차지하므로 심각한 캐시 공간 낭비로 이어지

지 않는다. 키-값 샤드(KV-Shard)에 대한 장치 용량이 주어지면 해시 테이블(HTable)의 수 h 는 정적으로 결정될 수 있다(예를 들어, 장치 용량이 1TB인 경우 $h = 218$). 마찬가지로, 키-값 샤드(KV-Shard)의 DRAM 크기가 주어지면 세트 캐시(SetCache)의 버킷 수는 정적으로 얻어질 수 있다(예를 들어, DRAM이 128MB인 경우 $b = 221$). 위의 예에서, $k = 23$, 8개의 버킷(= 32개의 해시 엔트리)이 동일한 해시 테이블(HTable)을 캐싱하는 데 사용될 수 있다.

[0057] 예를 들어, 각 해시 테이블(HTable)에 1,023개의 해시 항목이 있음을 고려하면 상위 3.128%의 핫 항목을 DRAM에 캐시할 수 있다. 이 속성을 통해 시스템 설계자는 워크로드에 대해 낙관적인 DRAM 대 SSD 비율을 구성할 수 있다.

[0058] 도 5는 충돌 무시 해싱을 사용한 객체 업데이트 및 세트 처리 방법을 보여준다.

[0059] 도 5에서, 캐싱 시스템(BigKV)은 키-값 쌍을 인덱싱할 때, 해시 충돌을 허용할 수 있다. 즉, 캐싱 시스템(BigKV)은 서로 다른 키-값 쌍이 같은 해시 값을 가질 때, 최근에 쓰인 키-값 쌍만을 시스템에 유지하고 기존 오래된 키-값 쌍은 추적하지 않을 수 있다. 따라서, 캐싱 시스템(BigKV)은 쓰기 연산(SET) 과정에서 같은 해시 값을 가지는 기존 데이터를 읽기 위한 종래 기술에 존재하는 과정을 생략함으로써 높은 SET 성능을 제공할 수 있다.

[0060] 일 실시예에서, 도 5는 캐싱 시스템(BigKV)이 세트(SET) 요청을 처리하는 방법을 보여준다. 쓰기 객체가 주어지면 키-값 디렉터(KV-Director)는 해시 함수를 사용하여 객체 키에서 128비트 해시 값을 계산할 수 있다. 해시 값의 가장 높은 $\log_2(s)$ 비트는 샤드 번호로 사용되며, 여기에서 s 는 시스템의 샤드 수에 해당할 수 있다. 이후, 캐싱 요청은 지정된 키-값 샤드(KV-Shard)로 전달될 수 있다. 키-값 샤드(KV-Shard)는 객체 헤더와 데이터를 먼저 데이터 영역에 쓸 수 있다. 그런 다음 키-값 샤드(KV-Shard)는 해시 값의 다음 $\log_2(b)$ 비트를 버킷 번호로 사용하여 새 항목이 추가될 대상 버킷을 선택할 수 있다. 해시 값의 가장 낮은 64비트가 객체의 지문으로 사용된다. 동일한 지문(FP)이 있는 해시 엔트리(또는 해시 항목)가 버킷에 있는 경우 캐싱 시스템(BigKV)은 데이터 영역의 최신 객체를 가리키도록 기존 항목을 업데이트할 수 있다.

[0061] 새 항목과 이전 항목이 전체 키가 다른 객체를 가리키는 경우 지문 충돌이 발생할 가능성이 있다(도 5의 Obj A 및 Obj B). 캐싱 시스템(BigKV)은 지문 충돌 가능성을 단순히 무시하여 새 항목이 이전 항목을 대체할 수 있도록 한다. 일치하는 항목이 없으면 캐싱 시스템(BigKV)은 버킷의 여유 슬롯에 새 해시 항목을 추가할 수 있다. 객체에 대한 이전 해시 항목이 해당 해시 테이블(HTable)에 있을 수 있다. 버킷이 테이블로 제거되면 이전 항목(또는 엔트리)은 결국 가장 최근의 새 항목으로 대체될 수 있다. 마지막으로 캐싱 시스템(BigKV)은 항목에 1비트 더티 비트를 설정하고 클라이언트에게 완료를 알릴 수 있다.

[0062] 해시 충돌을 무시함으로써 캐싱 시스템(BigKV)은 SET을 제공하는 동안 메타데이터 I/O를 포함하지 않는다. 그러나 이것은 필연적으로 땀글링 객체(dangling object)를 생성할 수 있다. 해시 충돌이 발생하면 이전(이전에 작성된) 해시 항목을 새 항목으로 덮어쓰므로 데이터 영역의 키-값 객체는 유효하지 않은 객체인 땀글링 객체가 될 수 있다(도 5의 Obj B). 땀글링 객체는 캐시 공간을 쓸데없이 차지할 뿐만 아니라 캐시 누락을 유발할 수 있다. 땀글링 객체가 핫 객체라고 가정하면 해당 객체를 대상으로 하는 미래의 LOOKUP이 항상 누락될 수 있다.

[0063] 그러나, 땀글링 객체의 부정적인 영향은 크지 않다. 작은 지문(예를 들어, Depot의 8비트)에 의존하는 다른 키-값 디자인과 달리, 큰 64비트 지문을 사용하는 본 발명의 2단계 해시 구성은 충돌률이 매우 낮다. 일 실시예에서, 캐싱 시스템(BigKV)에서 지문 충돌은 (i) 동일한 샤드, (ii) 동일한 세트 캐시(SetCache), (iii) 동일한 지문에 서로 다른 객체가 할당된 경우에만 발생할 수 있다. 예를 들어, 충돌률은 약 10⁻⁸일 수 있다. 4억 1400만 캐싱 요청 중 5개 캐시 누락만 충돌로 인해 발생할 수 있다.

[0064] 땀글링 객체는 거의 생성되지 않지만 시간이 지남에 따라 SSD에 축적되지 않는다. 땀글링 객체는 청소 과정에서 정기적으로 제거되기 때문이다. 성능을 위해 캐싱 시스템(BigKV)은 최신 해시 항목을 해시 테이블(HTable)에 즉시 플러시하는 대신 세트 캐시(SetCache)에 캐시할 수 있다. 따라서, 해시 테이블(HTable)의 이전 엔트리는 여전히 땀글링 객체를 가리킬 수 있다. 시스템 오류가 발생하면 캐싱 시스템(BigKV)은 최신 객체를 가리키는 DRAM의 최신 해시 항목을 잃을 수 있다. 캐싱 시스템(BigKV)은 오류로부터 메타데이터 지속성을 보장하는 로그 구조 설계를 사용하여 이 문제를 해결한다.

[0065] 도 6은 일 실시예에 따른 캐싱 시스템의 만료 객체의 즉각적 소거 방법을 보여주는 도면이다.

[0066] 캐싱 시스템(BigKV)은 상기 키-값 쌍의 속성정보에 포함된 만료시간 정보가 유사한 키-값 쌍들끼리 묶어서 키-

값 저장소에 저장할 수 있다.

- [0067] 일 실시예에서, 캐싱 시스템(BigKV)은 올 플래시 어레이(AFA) 공간을 관리하고 객체를 저장하고 정리하는 과정에서, 만료시간 정보(TTL)가 유사한 키-값 쌍들끼리 묶어서(동일 세그먼트 할당) 저장할 수 있다. 캐싱 시스템(BigKV)은 메타데이터와 데이터 영역을 각각 장치 공간 할당 및 정리 단위인 2MB 세그먼트로 분할할 수 있다. 캐싱 시스템(BigKV)은 메타데이터와 객체 데이터를 SSD에 쓰기 위해 로그 구조 접근 방식을 취한다. 캐싱 시스템(BigKV)은 별도의 2MB 쓰기 버퍼에 보류 중인 해시 테이블(HTable) 및 키-값 객체를 유지할 수 있다.
- [0068] 캐싱 시스템(BigKV)은 버퍼가 가득 차면 지정된 영역으로 플러시(flush)하여 메타데이터 영역(해시 테이블의) 또는 데이터 영역(키-값 객체 및 헤더)에 새로운 2MB 세그먼트를 형성할 수 있다. 이전 해시 테이블(HTable) 및 키-값 객체는 다른 세그먼트에서 변경되지 않은 상태로 남아 있을 수 있다. 사용 가능한 여유 공간이 없으면 캐싱 시스템(BigKV)은 청소를 트리거하여 이전 해시 테이블(HTable) 및 객체가 차지한 공간을 회수한다. 청소 비용을 최소화하고 만료된 키-값 객체를 사전에 제거하기 위해 캐싱 시스템(BigKV)은 만료시간 정보(TTL) 기반의 저장(또는 할당) 및 청소(또는 정리) 정책을 사용할 수 있다.
- [0069] 메타데이터 세그먼트의 할당 및 정리가 더 간단하기 때문에 캐싱 시스템(BigKV)이 메타데이터 세그먼트를 관리하는 것을 설명한 다음 데이터 세그먼트를 관리하는 방법을 설명한다.
- [0070] 메타데이터 세그먼트의 경우, 캐싱 시스템(BigKV)은 FIFO(First In First Out) 대기열을 사용하여 메타데이터 세그먼트를 추적한다. 캐싱 시스템(BigKV)은 보류 중인 해시 테이블(HTable)을 새 메타데이터 세그먼트에 쓴 후, 해당 세그먼트 번호를 대기열(queue)에 푸시(push)한다. 공간이 필요할 때, 캐싱 시스템(BigKV)은 대기열(queue)에서 가장 오래된 세그먼트를 정리 대상으로 팝(pop)할 수 있다. 오래된(오래 전에 작성된) 해시 테이블(HTable)이 최근 것보다 더 많은 유효하지 않은 항목을 가질 가능성이 높기 때문이다. 캐싱 시스템(BigKV)은 정리 대상 세그먼트의 모든 해시 테이블(HTable)을 읽을 수 있다. 2MB 메타데이터 세그먼트에는 64개의 해시 테이블(HTable)이 포함될 수 있다. 각 해시 테이블(HTable)에 대해 캐싱 시스템(BigKV)은 세트 캐시(SetCache)가 정리 대상 세그먼트의 테이블을 가리키는지 여부를 확인한다. 만약 그렇다면, 해당 테이블에는 최신 항목이 포함되어 있으므로 다른 곳으로 이동될 수 있다. 그렇지 않다면, 해당 해시 테이블(HTable)은 오래되어 폐기될 수 있다.
- [0071] 데이터 세그먼트의 경우, 캐싱 시스템(BigKV)은 정리 비용을 최소화하기 위해 SSD에 수명이 비슷한 데이터를 함께 배치할 수 있다. 보다 구체적으로 캐싱 시스템(BigKV)은 객체의 속성정보를 고려할 수 있다. 캐싱 시스템(BigKV)은 특히 만료시간 정보(TTL)을 고려하여 만료시간(TTL)이 비슷한 객체를 함께 배치할 수 있다. 객체의 만료시간(TTL)은 객체가 더 이상 사용되지 않을 때를 알려준다. 캐싱 시스템(BigKV)은 유사한 만료시간(TTL)을 가진 키-값 객체를 동일한 세그먼트에 배치하여, 유사한 시간에 만료되어 함께 제거되도록 할 수 있다.
- [0072] 일 실시예에서, 캐싱 시스템(BigKV)은 데이터 세그먼트를 32가지 유형으로 분류하며, 각 유형은 서로 다른 범위의 만료시간 정보(TTL, 예를 들어, [0, 60s), [60s, 120s), ..., [5,184,000s, ∞])를 포함할 수 있다. 캐싱 시스템(BigKV)은 각 세그먼트 유형에 대해 별도의 2MB 쓰기 버퍼를 유지할 수 있다. 각각의 쓰기 버퍼는 지정된 세그먼트 유형의 만료시간(TTL) 범위에 맞는 만료시간(TTL)을 가진 펜딩중인 키-값 객체를 버퍼링한다. 버퍼가 가득 차면 캐싱 시스템(BigKV)은 펜딩중인 중인 객체를 배출하고 해당 유형에 대한 새로운 데이터 세그먼트를 생성할 수 있다.
- [0073] 데이터 세그먼트를 관리하기 위해, 캐싱 시스템(BigKV)은 두 가지 유형의 FIFO 대기열을 사용한다. 첫 번째는 전역 FIFO 대기열(global FIFO queue)이다. 새로 생성된 데이터 세그먼트는 만료시간(TTL) 유형에 관계없이 전역 대기열로 푸시될 수 있다. 두 번째는 32개의 만료시간(TTL) 대기열로, 각 대기열은 동일한 만료시간(TTL)을 가지는 데이터 세그먼트의 목록을 가진다. 각 만료시간(TTL) 대기열의 헤더에는 처리할 수 있는 최대 만료시간(T_{max} TTL)이 있습니다. 예를 들어, 도 6에서 만료시간(TTL) 큐(대기열)는 만료시간(TTL)이 60초보다 길지만 120초를 초과하지 않는 객체를 저장하는 데이터 세그먼트를 포함한다. 만료시간(TTL) 큐 헤더는 각 세그먼트의 생성 시간을 기록할 수 있다(즉, 세그먼트가 기록된 시간). 세그먼트의 키-값 객체는 모두 최대 만료시간(T_{max} TTL) 전에 만료되므로 만료 시간은 $tc + T_{max}$ TTL로 추정할 수 있다.
- [0074] 일 실시예에서, 데이터 세그먼트 정리의 경우, 캐싱 시스템(BigKV)은 만료시간(TTL) 대기열에서 첫 번째(가장 오래된) 세그먼트를 스캔하고 예상 만료 시간이 현재 시간보다 짧은 만료된 세그먼트가 있는지 확인한다. 그렇다면 캐싱 시스템(BigKV)은 해당 세그먼트를 정리 대상 세그먼트로 선택할 수 있다. 정리 대상 세그먼트의 모든 객체가 만료되었으므로 폐기되고 정리 대상 세그먼트는 자유 데이터 세그먼트 목록에 배치될 수 있다.

- [0075] 청소를 위해 추가 I/O가 필요하지 않습니다. 만료된 세그먼트가 없으면 캐싱 시스템(BigKV)은 전역 대기열에서 가장 오래된 세그먼트를 선택하여 콜드 객체를 추출한다. 핫 객체가 제거되는 것을 방지하기 위해 캐싱 시스템(BigKV)은 객체 헤더의 전체 키를 읽고 세트 캐시(SetCache)가 해당 메타데이터를 캐시하는지를 확인할 수 있다. 만약 그렇다면, 해당 핫 객체는 삭제되지 않고 캐싱 시스템(BigKV)으로 다시 작성될 수 있다. 만약 그렇지 않으면, 해당 객체는 삭제될 수 있다. 모든 핫 오브젝트를 이동시킨 후, 정리 대상 세그먼트는 폐기되고 자유 목록에 배치될 수 있다.
- [0076] 캐싱 시스템(BigKV)은 해시 테이블(HTable)에서 추출된 객체의 해시 항목을 제거하지 않고 정리 프로세스를 완료한다. 이를 제거하려면 캐싱 시스템(BigKV)은 많은 해시 테이블(HTable)의 업데이트를 거쳐야 한다. 2MB 세그먼트에는 최대 512개의 4KB 객체가 포함되며 해당 해시 항목은 해시 테이블(HTable)에 고르게 분산된다. 최악의 경우 512개의 16KB 해시 테이블(HTable)을 업데이트해야 하며 여기에는 최대 8MB(= 512×16KB) 데이터 읽기 및 쓰기가 포함된다. 세그먼트 정리 후 2MB의 여유 공간이 회수된다는 점을 고려하면 비용이 너무 많이 든다. 일 실시예에서, 캐싱 시스템(BigKV)은 해시 테이블(HTable)의 업데이트를 건너뛰지만 데이터 세그먼트에서 제거된 객체의 위치를 가리키는 좀비 해시 항목을 해시 테이블(HTable)에 남긴다.
- [0077] 데이터 세그먼트 정리의 결과로 불가피하게 생긴 소위 좀비 항목은 시간이 지남에 따라 누적되어 결국 해시 테이블(HTable)의 공간을 차지한다. 캐싱 시스템(BigKV)은 데이터 세그먼트에서 만료된 키-값 객체를 즉시 제거할 수 있다. 그러나, 해시 테이블(HTable)의 해시 항목(또는 해시 엔트리)은 최근에 참조되었기 때문에 여전히 유효한 것(hot)으로 간주될 수 있다. 따라서, 핫 키-값 객체를 가리키는 다른 해시 항목이 LRU 정책에 의해 선택되고 제거될 가능성이 높다.
- [0078] 좀비 해시 항목을 빠르게 제거하기 위해 캐싱 시스템(BigKV)은 만료시간(TTL)을 활용할 수 있다. 객체별 메타데이터를 압축하기 위해 캐싱 시스템(BigKV)은 5비트 만료시간(TTL) 필드만 예약하여 해시 항목(키-값 객체가 아니라)의 만료 시간을 추정할 수 있다. 5비트 만료시간(TTL)은 상기 32가지 TTL 유형 중 하나를 나타낼 수 있다. 메타데이터 세그먼트를 정리하거나 세트 캐시(SetCache)에서 해시 테이블(HTable)로 더티(dirty) 항목을 제거할 때, 캐싱 시스템(BigKV)은 플래시에서 해시 테이블(HTable)을 읽는다. 이러한 프로세스와 결합하여 캐싱 시스템(BigKV)은 만료시간(TTL) 필드를 확인하고 만료된 항목을 제거한다. 따라서, 추가 I/O 없이 좀비를 정리할 수 있다.
- [0079] 도 7은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 방법을 보여주는 흐름도이다. 도 7은 샤딩 기반의 키-값 캐싱 시스템이 캐싱 요청에 따라 키-값 쌍들의 입출력을 처리하는 단계에서, 특정 에스에스디(SSD)가 고장난 경우의 샤딩 기반의 키-값 캐싱 방법을 보여준다. 도 3을 참조하여 설명한다.
- [0080] 도 7에서, 캐싱 시스템(BigKV)은 클라이언트의 캐싱 요청을 수신하면 복수의 키-값 샤드들(130) 중 제1 키-값 샤드로 디렉션하여 처리할 수 있다(단계 S100). 즉, 키-값 디렉터(120)는 제1 키-값 샤드에 키-값 객체 요청을 할당할 수 있다(단계 S110).
- [0081] 캐싱 시스템(BigKV)은 제1 키-값 샤드의 정상 동작 여부를 확인할 수 있다(단계 S120) 캐싱 시스템(BigKV)은 제1 키-값 샤드가 고장이 아닌 경우, 제1 키-값 샤드에서 캐싱 요청을 처리하고(단계 S130), 상기 제1 키-값 샤드가 고장인 경우, 정상 상태인 제2 키-값 샤드로 상기 캐싱 요청을 리디렉션하여(단계 S140) 상기 제2 키-값 샤드가 상기 캐싱 요청을 처리할 수 있다(단계 S160).
- [0082] 이 후, 캐싱 시스템(BigKV)은 제1 키-값 샤드의 복구 여부를 확인할 수 있다(단계 S150). 제1 키-값 샤드가 복구되면, 캐싱 시스템(BigKV)은 리디렉션된 캐싱 요청을 복구된 제1 키-값 샤드로 마이그레이션할 수 있다(단계 S170). 복구된 제1 키-값 샤드에 캐싱 요청이 다시 할당되면 제1 키-값 샤드는 해당 캐싱 요청을 처리할 수 있다.
- [0083] 보다 구체적으로, 캐싱 시스템(BigKV)은 에스에스디(SSD)의 오류가 발생하는 경우 문제가 있는 키-값 샤드(KVShard)를 비활성화하고 연결된 키-값 객체를 포기할 수 있다.
- [0084] 본 발명의 캐싱 시스템(BigKV)은 내결함성을 갖추기 위해 데이터의 손실을 허용할 수 있다. 캐싱 시스템(BigKV)은 들어오는 키-값 요청을 실패 시 정상 키-값 샤드로 리디렉션할 수 있다. 키-값 샤드의 번호는 객체 키로 해시 함수에 의해 정적으로 결정된다. 실패한 키-값 샤드로 향하는 키-값 객체는 더 이상 캐시될 수 없으며 클라이언트는 실패한 SSD가 복구될 때까지 반복적인 캐시 누락으로 고통받는다. 캐싱 시스템(BigKV)은 추가 해시 함수를 사용하여 이 문제를 해결한다. 실패한 키-값 샤드에 매핑된 키-값 객체의 경우 캐싱 시스템(BigKV)은 다른 해시 함수를 사용하여 키-값 샤드의 번호를 계산하고 정상 키-값 샤드로 리디렉션 할 수 있다. 리디렉션된

키-값 객체를 읽는 것은 새로운 해시 함수를 통해 이루어질 수 있다.

[0085] 캐싱 시스템(BigKV)은 리더렉션된 키-값 객체를 효율적으로 마이그레이션할 수 있다. 실패한 키-값 샤드가 복구된 후 캐싱 시스템(BigKV)은 다른 정상 키-값 샤드로 리더렉션된 키-값 객체를 복구된 키-값 샤드로 마이그레이션할 수 있다. 일 실시예에서, 마이그레이션은 완전한 풀-마이그레이션 및 마이그레이션 없는 노-마이그레이션의 두 가지 극단적인 솔루션을 포함할 수 있다. 풀-마이그레이션은 모든 정상 키-값 샤드를 스캔하고 리더렉션된 객체를 복구된 키-값 샤드로 마이그레이션한다. 캐싱 시스템(BigKV)은 모든 키-값 객체가 마이그레이션된 후 복구된 키-값 샤드를 활성화하여 클라이언트의 캐싱 요청을 처리할 수 있다. 기존의 RAID와 비교하여 캐싱 시스템(BigKV)의 풀-마이그레이션은 시스템의 모든 SSD를 스캔하고 실패한 블록을 조립할 필요가 없기 때문에 더 작은 I/O가 필요하다. 그러나 풀-마이그레이션은 여전히 중요하지 않은 I/O가 발생하여 활성화까지 오랜 시간이 걸릴 수 있다. 노-마이그레이션은 정상 키-값 샤드에 임시로 저장된 모든 키-값 객체를 버리고 즉시 복구된 키-값 샤드를 활성화할 수 있다. 이를 위해서는 마이그레이션을 위한 I/O가 필요하지 않지만 복구된 키-값 샤드가 충분히 워밍업될 때까지 높은 실패율이 발생할 수 있다.

[0086] 일 실시예에서, 캐싱 시스템(BigKV)은 실패한 키-값 샤드(KV-Shard)가 복구되면 처음에 이를 비활성화 상태로 유지하고 들어오는 키-값 캐싱 요청은 정상적인 키-값 샤드에서 처리할 수 있다. 캐싱 시스템(BigKV)은 클라이언트가 읽은 키-값 객체가 복구된 키-값 객체에 속하는 경우 이를 원래 키-값 샤드로 전달할 수 있다. 그런 다음 해당 키-값 객체는 복구된 키-값 샤드에서 직접 제공될 수 있다. 캐싱 시스템(BigKV)은 원래 키-값 샤드에 핫 객체에 대한 거의 모든 요청을 처리하는 충분한 수의 키-값 객체가 포함된 후 복구된 키-값 샤드를 활성화하고 다른 키-값 샤드의 나머지 키-값 객체를 버린다. 따라서, 캐싱 시스템(BigKV)은 마이그레이션 비용을 최소화할 뿐만 아니라 복구된 키-값 샤드를 빠르게 워밍업할 수 있다.

[0087] 캐싱 시스템(BigKV)은 해시 테이블(HTable)로 플러시될 때까지 세트 캐시(SetCache)에 더티 해시 항목을 유지할 수 있다. 갑작스러운 시스템이나 정전이 발생하면 더티 항목은 필연적으로 손실될 수 있다. 캐싱 시스템(BigKV)은 로그 구조 설계 덕분에 뛰어난 메타데이터 지속성을 달성할 수 있다. 캐싱 시스템(BigKV)은 전체 속성이 포함된 헤더와 함께 데이터 세그먼트에 키-값 객체를 추가할 수 있다. 모든 데이터 세그먼트는 서로 연결되어 거대한 로그를 형성한다. 시스템 장애가 발생하더라도 캐싱 시스템(BigKV)은 로그에 기록된 객체 헤더를 스캔하여 최신 해시 테이블(HTable)을 복구할 수 있다. 로그가 너무 길어지는 것을 방지하기 위해 캐싱 시스템(BigKV)은 다른 로그 구조 시스템의 체크포인트와 유사하게 해시 테이블(HTable)에 대한 더티 항목을 정기적으로 플러시할 수 있다.

[0088] 도 8 내지 도 11은 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템의 효과를 보여주는 그래프들이다. 도 8 및 도 9는 본 발명의 캐싱 시스템(BigKV)의 성능을 YCSB(Yahoo Cloud Serving Benchmark)를 사용하여 기존의 캐싱 시스템인 uDepot 및 SlickCache와 비교한 그래프를 나타낸다.

[0089] 도 8 및 도 9는 다양한 오브젝트 크기에 대하여 높은 접근 지역성을 보이는 YCSB 벤치마크를 수행하여 캐싱 시스템들을 평가한 결과를 나타낸다. 도 7 및 도 9에서, uDepot-OPT와 BigKV-OPT는 결과 분석을 위한 설정으로 모든 메타데이터를 DRAM에 적재하는 경우이고, 나머지 uDepot-Cache와 SlickCache, BigKV는 메타데이터의 일부분을 DRAM에 캐싱하는 경우를 나타낸다. 본 발명의 캐싱 시스템(BigKV)은 해시 충돌 무시 2단계 해시 테이블 기법의 효과에 따라 uDepot-Cache 대비 2배, SlickCache 대비 8.8배 높은 성능을 보이며, 훨씬 짧은 응답시간을 보장하는 것을 알 수 있다.

[0090] 보다 구체적으로, 도 8은 고도로 지역화된 YCSB 작업 부하(99:1)에서 본 발명의 일 실시예에 따른 샤딩 기반의 키-값 캐싱 시스템(또는, 캐싱 시스템(BigKV))의 처리량을 보여준다. 평균 개체 크기가 4KB 및 32KB일 때 캐싱 시스템(BigKV)은 uDepot-OPT에 필적하는 처리량을 나타내고, uDepot-Cache 및 SlickCache보다 각각 평균 2.0배 및 8.8배 능가한다. 캐싱 시스템(BigKV)은 고도로 지역화된 개체 액세스 패턴을 활용하여 99.3%의 메타데이터 적중률을 달성할 수 있다. 캐싱 시스템(BigKV)은 세트 캐시(SetCache)에 있는 핫 항목의 상위 3.128%를 캐시하며, 이는 고도로 지역화된 워크로드의 1% 핫 항목을 보유하기에 충분히 크다. 높은 적중률 덕분에 캐싱 시스템(BigKV)은 BigKV-OPT와 유사한 처리량을 달성할 수 있다.

[0091] uDepot-Cache는 개체의 높은 지역성을 활용하지 못하여 낮은 메타데이터 적중률(8.9%)을 보여준다. 이는 uDepot-Cache가 사용자 요청을 처리하기 전에 인플레이시 해시 테이블을 조회해야 하므로 SSD에 대한 I/O 트래픽이 거의 두 배가 되기 때문이다. SlickCache는 또한 DRAM의 핫 항목을 캐시하여 높은 적중률을 달성한다. 그럼에도 불구하고 SlickCache는 모든 워크로드에서 심각한 처리량 저하를 겪고 있다. 이는 캐시 미스가 발생할 때 높은 패널티 때문이다. 캐시 미스 시 SlickCache는 플래시 내 해시 테이블을 통과해야 하므로 많은 I/O가 발생

한다. SlickCache는 또한 정기적으로 병합 작업을 수행해야 한다. 성능을 위해 SlickCache는 SSD(웜 계층)의 로그에 많은 더티 항목을 한 번에 플러시한 다음 나중에 플래시 내 해시 테이블(콜드 계층)과 병합한다. 그러나 웜 및 콜드 계층을 병합하면 많은 I/O가 발생하여 SlickCache가 느려진다.

[0092] 도 8에서, YCSB-A(도 8(a)) 및 F(도 8(e))는 캐싱 시스템(BigKV)이 uDepot-OPT보다 성능이 우수하다는 것을 보여준다. 캐싱 시스템(BigKV)은 지문 충돌을 무시하지만 uDepot은 개체 헤더를 읽고 모든 SET에 대해 충돌이 발생하는지 확인해야 한다. 그 결과 중간 정도의 성능 저하가 발생한다. 평균 개체 크기가 128KB 이상이면 모든 시스템이 유사한 처리량을 나타낸다. 두 가지 이유 때문이다. 첫째, 객체의 크기가 커질수록 인덱싱할 객체의 수가 줄어든다. 동일한 DRAM이 주어지면 해시 테이블의 더 많은 부분을 캐시할 수 있으므로 메타데이터 적중률이 높아진다. uDepot-Cache조차도 해시 테이블의 많은 부분을 DRAM으로 가져오므로 처리량이 향상된다. 대부분의 더티 항목이 DRAM에 캐시되고 SSD로 제거되지 않기 때문에 SlickCache의 병합 비용도 낮아진다. 이렇게 하면 개체 크기가 32KB 이하일 때보다 SlickCache가 더 빨라진다. 둘째, 메타데이터 누락의 부정적인 영향이 감소하기 때문이다. 대신 SSD에서 큰 개체(128KB 및 256KB)를 읽는 것이 지배적인 부분이 된다.

[0093] 도 9는 LOOKUP이 우세한(95%) YCSB-B 조회 대기 시간의 CDF를 보여주고 다른 것들은 유사한 대기 시간 경향을 가지고 있는 것을 보여준다. 캐싱 시스템(BigKV)은 높은 메타데이터 적중률 덕분에 uDepot-OPT와 거의 동일한 조회 대기 시간을 제공한다. SlickCache도 높은 적중률을 달성하지만 대기 시간이 길다. 내부 I/O(인플레이스 테이블을 스캔하고 웜 및 콜드 계층을 병합하는 데 필요)가 클라이언트 요청을 자주 지연시키기 때문이다. 개체 크기가 128KB 및 256KB로 증가하면 이러한 간섭이 사라진다. uDepot-Cache는 메타데이터 미스 비율이 높기 때문에 캐싱 시스템(BigKV) 및 SlickCache보다 지연 시간이 더 길다.

[0094] 도 10은 SSD 용량에 따른 캐시 시스템의 캐시 적중률을 보여준다. 캐싱 시스템(BigKV)은 즉각적인 만료 오브젝트 소거 기술을 사용하여, 기존의 단순 만료시간을 고려하지 않고 용량이 가득찬 경우에 오브젝트들을 소거하는 방식(uDepot) 대비 높은 캐시 적중률을 보인다. 캐싱 시스템(BigKV)은 uDepot과 같은 수준의 캐시 적중률을 달성하기 위해 2배 더 작은 용량을 필요로 한다.

[0095] 도 10에서, 만료시간(TTL) 인식 기반의 공간 관리가 적중률에 미치는 영향을 분석한다. 도 10은 uDepot, BigKV-HOT 및 본 발명의 캐싱 시스템(BigKV)의 세 가지 기술의 적중률을 비교한다. uDepot은 LRU를 사용하여 피해자 객체를 선택하고 AFA 캐시에서 모든 객체를 삭제한다. BigKV-HOT은 전역 대기열에서 콜드 세그먼트를 선택하고 핫 오브젝트만 자유 세그먼트로 이동한다. 만료시간(TTL) 만료 객체를 명시적으로 제거하지 않는다. 핫 객체 마이그레이션 외에도 캐싱 시스템(BigKV)은 TTL 근사치를 통해 만료된 객체를 사전에 제거할 수 있다.

[0096] 도 10에서, 올 플래시 어레이(AFA) 용량을 768GB에서 2TB로 늘리면서 캐시 적중률을 표시한다. AFA 용량이 1TB 일 때 캐싱 시스템(BigKV)은 추적에서 달성할 수 있는 최대 적중률인 92.1%를 달성한다. BigKV-HOT과 uDepot 모두 AFA 용량을 2TB로 두 배로 늘리면 이 적중률을 제공한다. 프로덕션 환경에서 모든 캐싱 시스템은 목표 미스 비율을 충족하기에 충분한 캐시 공간을 가져야 한다. 도 10은 캐싱 시스템(BigKV)이 더 작은 AFA 용량으로 지정된 미스 비율을 달성할 수 있음을 보여준다.

[0097] 도 11은 복수 배열 독립 디스크(RAID, Redundant Array of Independent Disks)를 사용하여 다수의 SSD를 관리하는 경우(BigKV-RAID)와 본 발명에 따라 샤딩 방식으로 저장장치를 관리하는 캐싱 시스템(BigKV)의 성능 확장성 평가 결과를 나타낸다. 캐싱 시스템(BigKV)은 8개의 SSD를 사용했을 때 BigKV-RAID 대비 5.4배 높은 읽기 성능과 5.8배 높은 쓰기 성능을 보이는 것을 알 수 있다.

[0098] 캐싱 시스템(BigKV)은 SSD 장애 시 BigKV-RAID 비해 안정적으로 작동한다. 예를 들어, 8개의 SSD가 있을 때, 캐싱 시스템(BigKV)과 BigKV-RAID의 두 시스템을 비교한다. 캐싱 시스템(BigKV)은 각각 하나의 SSD와 결합된 8개의 키-값 샤드(KV-Shard)를 실행하고 반응적 내결함성 메커니즘을 사용하여 관리할 수 있다. BigKV-RAID는 RAID-6으로 8개의 SSD를 그룹화하여 생성된 볼륨에서 8개의 키-값 샤드(KV-Shard)를 실행한다. BigKV-RAID는 패리티 블록에 용량의 25%를 사용한다. 캐싱 시스템(BigKV)은 캐시 용량이 더 크기 때문에 BigKV-RAID(79%)보다 적중률(83%)이 더 높다. 캐싱 시스템(BigKV)의 처리량은 BigKV-RAID의 처리량보다 3배 높다.

[0099] 장치 오류의 경우 키-값 디렉터(KV-Director)가 들어오는 키-값 객체를 다른 키-값 샤드(KV-Shard)로 리디렉션하기 때문에 적중률과 처리량이 점차 증가한다. 캐싱 시스템(BigKV)은 백그라운드에서 복구된 키-값 샤드로 핫 객체를 전송하는 반면 다른 키-값 샤드는 요청을 계속 처리하므로 복구 중에 처리량 저하가 발생하지 않는다.

[0100] 캐싱 시스템(BigKV)과 달리 BigKV-RAID의 적중률은 용량 손실이 없기 때문에 SSD 장애의 영향을 받지 않는다. 그러나 실패한 SSD가 감지된 후 BigKV-RAID는 성능 저하 모드에 들어가고 패리티를 사용하여 손상된 데이터의

온라인 재구성으로 인해 36%의 성능 저하를 경험한다. 고장난 SSD를 복구한 후에도 BigKV-RAID는 RAID 재구성 프로세스가 완료될 때까지 300분 이상의 시간이 소요될 때까지 느리게 수행한다.

[0101] 도 11은 SSD 수를 늘리면서 캐싱 시스템(BigKV)과 BigKV-RAID의 처리량을 비교한 것이다. 캐싱 시스템(BigKV)은 8개의 SSD가 있는 BigKV-RAID보다 5.4배 및 5.8배 더 높은 처리량을 제공하는 것을 알 수 있다. 캐싱 시스템(BigKV)의 성능은 SSD 수에 따라 선형으로 확장된다. 캐싱 시스템(BigKV)의 키-값 디렉터(120, 도 2 참조)는 들어오는 요청을 키-값 샤드(130, 도 2 참조) 전체에 고르게 분배하는 반면, 각 키-값 샤드(130)는 이를 동시에 처리할 수 있다. BigKV-RAID는 패리티 블록에 대한 추가 I/O 및 계산으로 인해 확장 가능한 성능을 제공하지 못한다. RAID-6에는 최소 4개의 SSD가 필요하기 때문에 2개의 SSD가 있는 BigKV-RAID는 작동하지 않는다.

[0102] 본 발명의 실시예에 따른 구성요소, 예컨대 모듈 또는 프로그램 각각은 단수 또는 복수의 서버 구성요소로 구성될 수 있으며, 이러한 서버 구성요소들 중 일부 서버 구성요소가 생략되거나, 또는 다른 서버 구성요소가 더 포함될 수 있다. 일부 구성요소들(모듈 또는 프로그램)은 하나의 객체로 통합되어, 통합되기 이전의 각각의 해당 구성요소에 의해 수행되는 기능을 동일 또는 유사하게 수행할 수 있다. 본 발명의 실시예에 따른 모듈, 프로그램 또는 다른 구성요소에 의해 수행되는 동작들은 순차적, 병렬적, 반복적 또는 휴리스틱하게 실행되거나, 적어도 일부 동작이 다른 순서로 실행되거나, 생략되거나, 또는 다른 동작이 추가될 수 있다.

[0103] 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

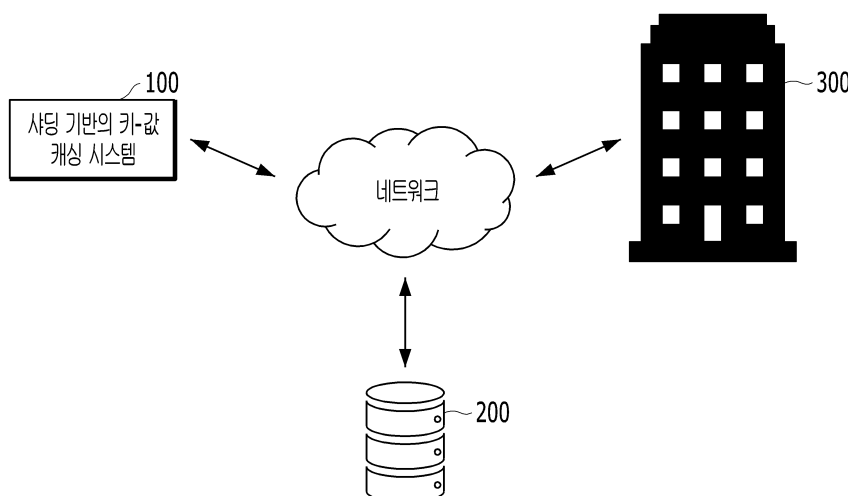
[0104] 본 발명의 범위는 후술하는 청구범위에 의하여 나타내어지며, 청구범위의 의미 및 범위 그리고 그 균등 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

부호의 설명

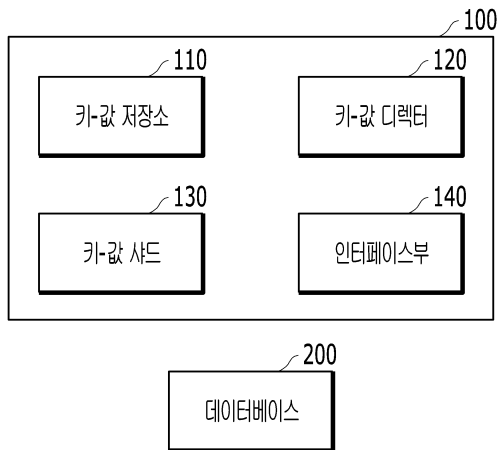
[0105] 100: 샤딩 기반의 키-값 캐싱 시스템
200: 데이터 베이스
300: 서버

도면

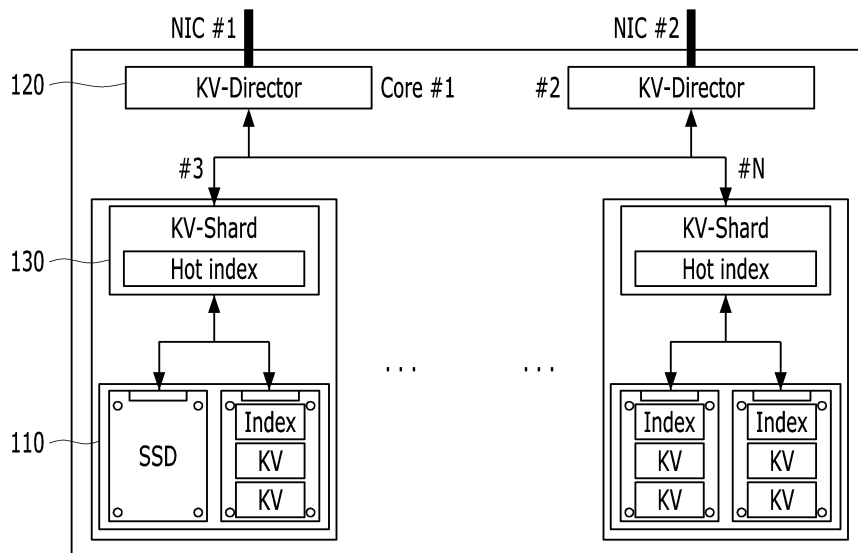
도면1



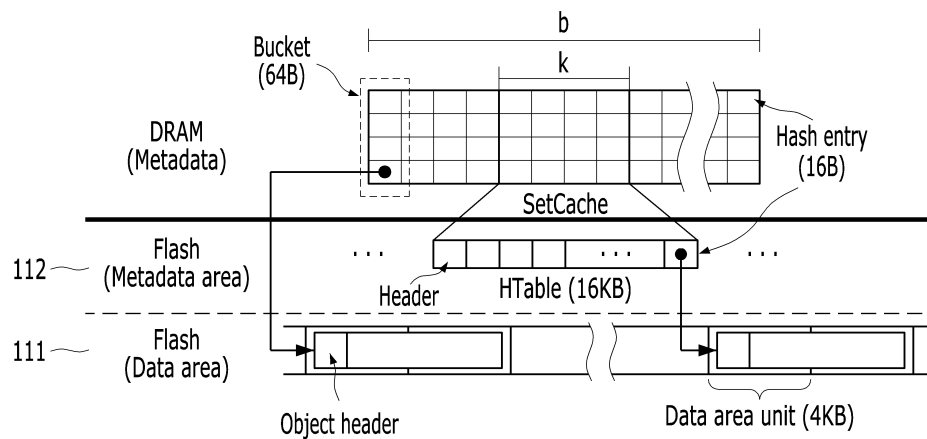
도면2



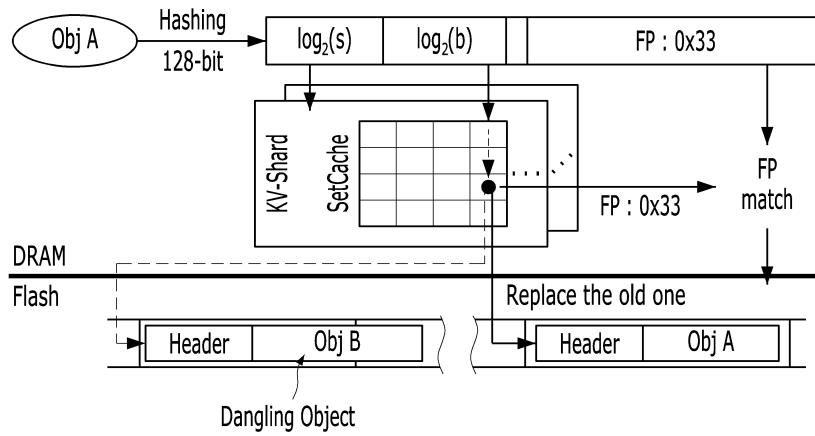
도면3



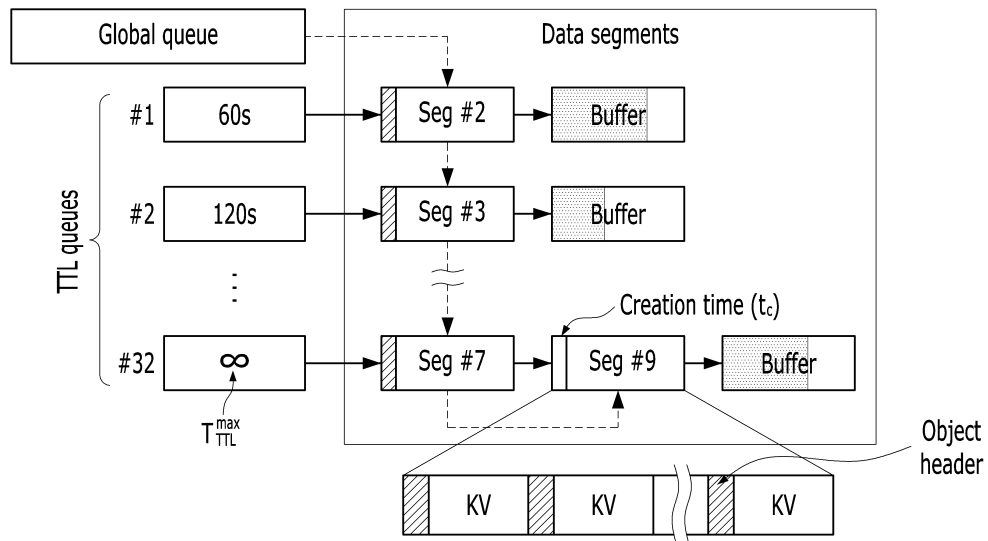
도면4



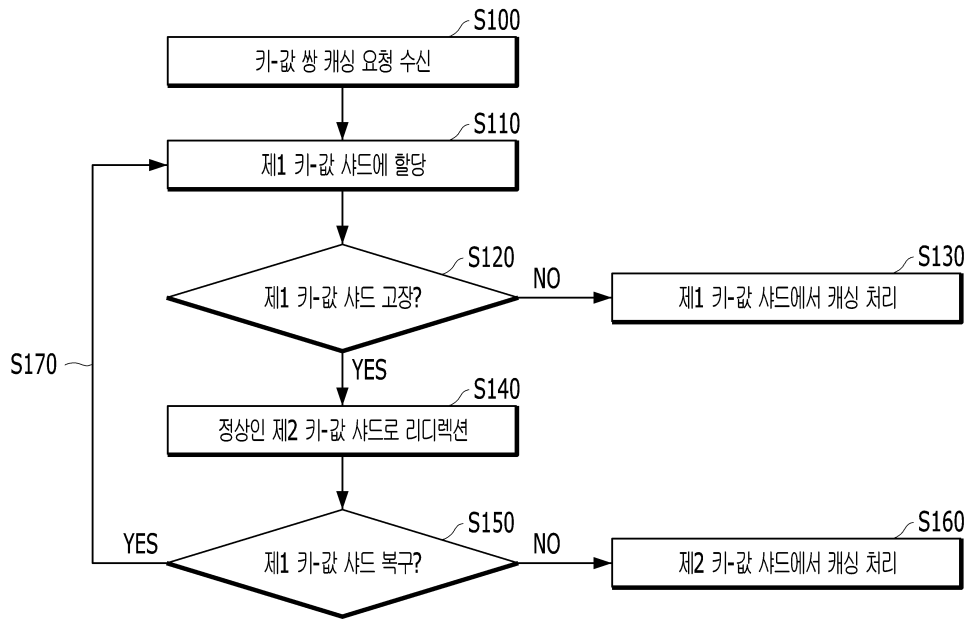
도면5



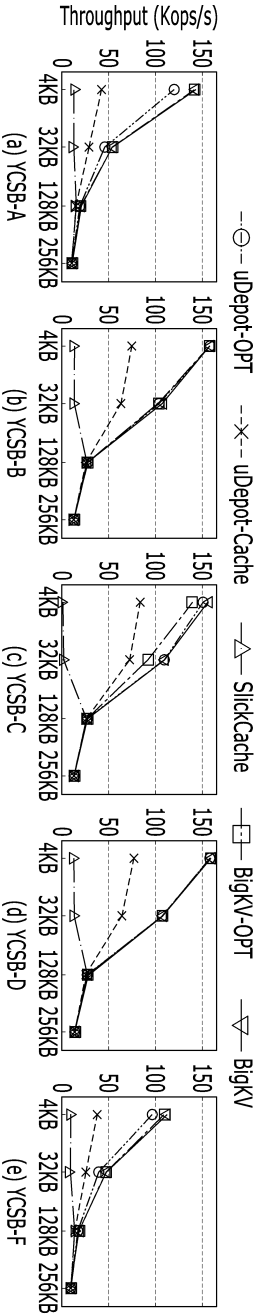
도면6



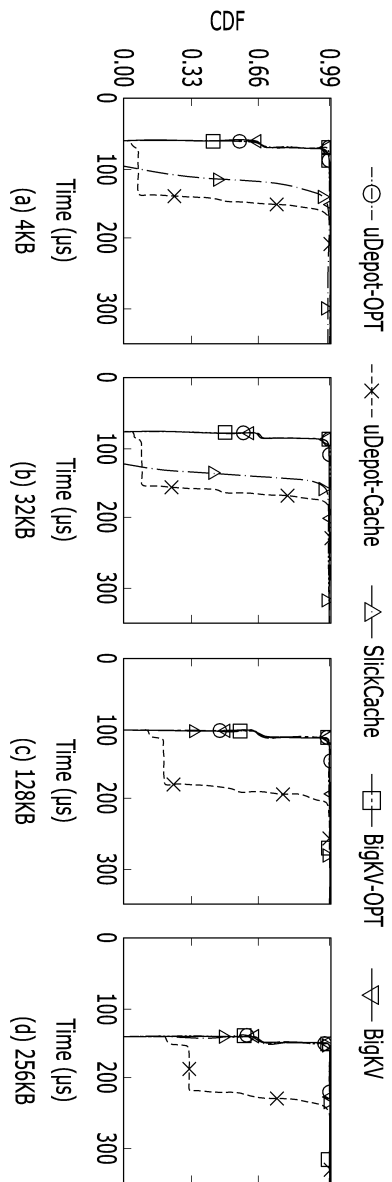
도면7



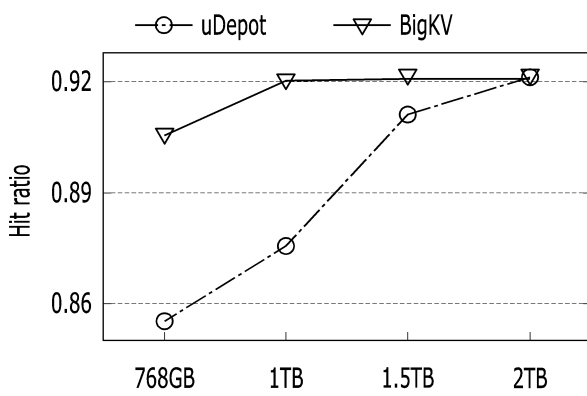
도면8



도면9



도면10



도면11

