



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2018-0059197
(43) 공개일자 2018년06월04일

(51) 국제특허분류(Int. Cl.)
G06F 11/10 (2006.01) G06F 3/06 (2006.01)
(52) CPC특허분류
G06F 11/1076 (2013.01)
G06F 3/0641 (2013.01)
(21) 출원번호 10-2016-0158480
(22) 출원일자 2016년11월25일
심사청구일자 2016년11월25일

(71) 출원인
인하대학교 산학협력단
인천광역시 남구 인하로 100, 인하대학교 (용현동)
(72) 발명자
김덕환
서울특별시 양천구 목동로 230, 103동 1004호 (목동, 아이파크아파트)
허희성
인천광역시 연수구 솔샘로 20, 101동 1408호 (청학동, 삼용아파트)
(74) 대리인
특허법인 정안

전체 청구항 수 : 총 5 항

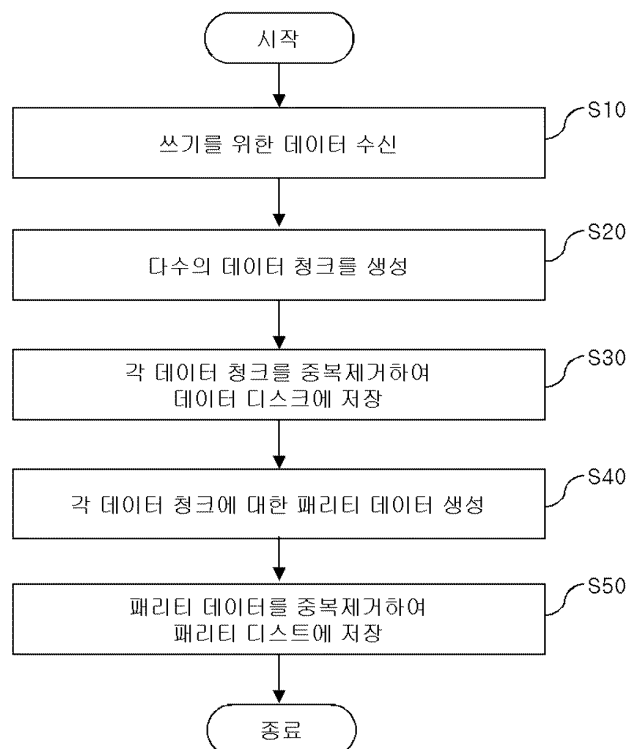
(54) 발명의 명칭 **충 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법**

(57) 요약

본 발명은 오픈스택 클라우드 서비스를 위한 블록 스토리지 또는 신더(Cinder)에서 SSD로 구성된 RAID 기반 충 플래시 어레이(All Flash Array) 디스크에 패리티 데이터 중복제거를 적용하는, 충 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 관한 것으로서, (a) 쓰기를 위한 데이터를 입력받

(뒷면에 계속)

대표도 - 도2



는 단계; (b) 상기 데이터를 체크 단위로 분할하여 다수의 데이터 체크를 생성하는 단계; (c) 각 데이터 체크에 대하여 중복제거를 수행하여, 데이터 디스크에 저장하는 단계; (d) 상기 각 데이터 체크에 대한 패리티 데이터를 생성하는 단계; 및, (e) 상기 패리티 데이터에 대하여 중복제거를 수행하여, 패리티 디스크에 저장하는 단계를 포함하는 구성을 마련한다.

상기와 같은 방법에 의하여, 중복되는 데이터를 갖는 워크로드 처리 시에 효율적으로 패리티 쓰기 연산 횟수를 감소시킬 수 있다.

(52) CPC특허분류

G06F 2211/1057 (2013.01)

G06F 2212/262 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호 1711042413

부처명 미래창조과학부

연구관리전문기관 정보통신기술 진흥센터(IITP)

연구사업명 SW컴퓨팅산업원천기술개발

연구과제명 차세대 헬스케어 서비스를 위한 고사용성, 고가용성, 고성능 클라우드 스토리지 기술연구

기 여 율 1/2

주관기관 인하대학교 산학협력단

연구기간 2016.06.01 ~ 2016.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호 1345241699

부처명 교육부

연구관리전문기관 한국연구재단

연구사업명 이공학개인지초연구지원사업

연구과제명 클라우드 기반 software Defined storage의 고성능, 고신뢰성, 고가용성 및 공간효율성 향상을 위한 시스템 소프트웨어

기 여 율 1/2

주관기관 인하대학교 산학협력단

연구기간 2015.11.01 ~ 2016.10.31

명세서

청구범위

청구항 1

오픈스택 프레임워크에 의한 신더(Cinder)가 다수의 SSD(Solid-State Drive)를 이용하여 블록 스토리지를 가상화하여 제공하는, 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 있어서,

- (a) 쓰기를 위한 데이터를 수신하는 단계;
- (b) 수신된 데이터를 청크 단위로 분할하여 다수의 데이터 청크를 생성하는 단계;
- (c) 각 데이터 청크에 대하여 중복제거를 수행하여, 데이터 디스크에 저장하는 단계;
- (d) 상기 각 데이터 청크에 대한 패리티 데이터를 생성하는 단계; 및,
- (e) 상기 패리티 데이터에 대하여 중복제거를 수행하여, 패리티 디스크에 저장하는 단계를 포함하는 것을 특징으로 하는 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법.

청구항 2

제1항에 있어서,

상기 (c)단계에서, 상기 패리티 데이터의 중복 여부를 판단하기 위하여 해시 테이블(hash table)을 이용하되, 상기 데이터 청크의 중복 여부를 판단하기 위해 사용되는 해쉬 테이블을 같이 이용하는 것을 특징으로 하는 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법.

청구항 3

제1항에 있어서,

상기 (c)단계에서, 상기 패리티 데이터의 중복 여부를 판단하기 위하여 해시 테이블(hash table)을 이용하되, 상기 데이터 청크의 중복 여부를 판단하기 위해 사용되는 해쉬 테이블을 같이 이용하는 것을 특징으로 하는 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법.

청구항 4

제1항에 있어서,

상기 블록 스토리지는 다수의 SSD로 구성되고, 상기 다수의 SSD를 레이드(RAID)-6에 의해 데이터 디스크와 패리티 디스크로 분할하여, 데이터 청크 또는 패리티 데이터를 저장하는 것을 특징으로 하는 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법.

청구항 5

제1항 내지 제4항 중 어느 한 항의 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법을 수행하는 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

발명의 설명

기술 분야

[0001] 본 발명은 오픈스택 클라우드 서비스를 위한 블록 스토리지 또는 신더(Cinder)에서 SSD로 구성된 RAID 기반 총 플래시 어레이(All Flash Array) 디스크에 패리티 데이터 중복제거를 적용하는, 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 관한 것이다.

배경 기술

[0002] 최근 인터넷에서 생성되는 데이터의 양이 급격하게 증가하였으며, 많은 양의 데이터를 신속하게 처리하기 위해 고성능 클라우드 스토리지 시스템을 구축하여 활용하는 경우가 많아지고 있다. 이러한 클라우드 스토리지 시스템의 성능 및 신뢰성을 향상시키기 위한 방법 중 하나는 스토리지에 RAID를 구성하여 사용하는 것이다.

[0003] 고성능 클라우드 스토리지를 구성하기 위해서는 주로 NAND 플래시 메모리(Flash memory) 기반의 SSD를 사용하고 있는데, SSD는 HDD에 비해 높은 성능을 갖지만 단점으로는 용량 당 가격이 HDD에 비해 높으며, 수명이 존재하여 일정 횟수 이상 쓰기/지우기 연산을 수행하면 사용이 불가능하다는 단점이 있다. SSD를 사용하여 RAID 기반 고성능 클라우드 스토리지를 구성해야 하는 경우는 이러한 단점들이 문제가 될 수 있으며, RAID가 적용된 디스크에 작은 크기의 쓰기 연산이 발생한 경우에는 디스크에 쓰여진 데이터 뿐 아니라 패리티 데이터도 지속적으로 갱신해야 하므로 SSD의 마모는 더욱 빠르게 진행된다.

[0004] 위와 같은 문제의 해결을 위해서 스토리지에 데이터 중복제거 기법을 적용할 수 있다. SSD들로 이루어진 RAID 디스크에 중복제거를 적용하면 중복되는 데이터에 대한 쓰기 연산을 제거할 수 있으므로 중복제거를 적용하지 않았을 때에 비해 SSD의 수명을 증가시킬 수 있다는 장점이 있다. 그러나 중복제거를 전체 디스크에 적용할 때는 디스크로 들어오는 모든 데이터에 대한 핑거프린트를 생성해야 하므로 연산량이 많아지며 해시테이블이 차지하는 DRAM 용량도 증가하게 된다.

선행기술문헌

비특허문헌

[0005] (비특허문헌 0001) Voras, Ivan, et al., "Evaluating open-source cloud computing solutions", Proc. of the 34th Intl. Convention on MIPRO 2011, pp.209-214, May. 2011.

(비특허문헌 0002) Wen, Xiaolong, et al., "Comparison of Open-Source Cloud Management Platforms: OpenStack and OpenNebula", Proc. of the 9th Intl. Conference on Fuzzy Systems and Knowledge Discovery(FSKD), pp.2457-2461, May. 2012.

(비특허문헌 0003) Hui-Seong Heo, Mehdi Pirahandeh, Kwang-Soo Lee, Deok-Hwan Kim, "All Flash Array Storage Virtualisation using SCST", KIISE Transactions on Computing Practices, Vol.20, No.10, pp.525-533, Oct. 2014.

(비특허문헌 0004) Nil Mistry, et al., "Throughput Studies on an InfiniBand Interconnect via All-to-All Communications", Proc. of the 23rd High Performance Computing Symposium, Apr. 2015

(비특허문헌 0005) Liu, Jiuxing, Jiesheng Wu and Dhabaleswar K. Panda., "High performance RDMA-based MPI implementation over InfiniBand", Intl. Journal of Parallel Programming, Vol.32, No.3, Jun. 2004

발명의 내용

해결하려는 과제

[0006] 본 발명의 목적은 상술한 바와 같은 문제점을 해결하기 위한 것으로, 오픈스택 클라우드 서비스를 위한 블록 스토리지 또는 신더(Cinder)에서 SSD로 구성된 RAID 기반 총 플래시 어레이(All Flash Array) 디스크에 패리티 데이터 중복제거를 적용하는, 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법을 제공하는 것이다.

과제의 해결 수단

- [0007] 상기 목적을 달성하기 위해 본 발명은 오픈스택 프레임워크에 의한 신더(Cinder)가 다수의 SSD(Solid-State Drive)를 이용하여 블록 스토리지를 가상화하여 제공하는, 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 관한 것으로서, (a) 쓰기를 위한 데이터를 입력받는 단계; (b) 상기 데이터를 청크 단위로 분할하여 다수의 데이터 청크를 생성하는 단계; (c) 각 데이터 청크에 대하여 중복제거를 수행하여, 데이터 디스크에 저장하는 단계; (d) 상기 각 데이터 청크에 대한 패리티 데이터를 생성하는 단계; 및, (e) 상기 패리티 데이터에 대하여 중복제거를 수행하여, 패리티 디스크에 저장하는 단계를 포함하는 것을 특징으로 한다.
- [0008] 또한, 본 발명은 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 있어서, 상기 (c)단계에서, 상기 패리티 데이터의 중복 여부를 판단하기 위하여 해시 테이블(hash table)을 이용하되, 상기 데이터 청크의 중복 여부를 판단하기 위해 사용되는 해쉬 테이블을 같이 이용하는 것을 특징으로 한다.
- [0009] 또한, 본 발명은 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 있어서, 상기 (c)단계에서, 상기 패리티 데이터의 중복 여부를 판단하기 위하여 해시 테이블(hash table)을 이용하되, 상기 데이터 청크의 중복 여부를 판단하기 위해 사용되는 해쉬 테이블을 같이 이용하는 것을 특징으로 한다.
- [0010] 또한, 본 발명은 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 있어서, 상기 블록 스토리지는 다수의 SSD로 구성되고, 상기 다수의 SSD를 레이드(RAID)-6 에 의해 데이터 디스크와 패리티 디스크로 분할하여, 데이터 청크 또는 패리티 데이터를 저장하는 것을 특징으로 한다.
- [0011] 또한, 본 발명은 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법을 수행하는 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것이다.

발명의 효과

- [0012] 상술한 바와 같이, 본 발명에 따른 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법에 의하면, 중복되는 데이터를 갖는 워크로드 처리 시에 효율적으로 패리티 쓰기 연산 횟수를 감소시킬 수 있는 효과가 얻어진다.

도면의 간단한 설명

- [0013] 도 1은 본 발명에서 사용하는 오픈스택 클라우드 시스템에 대한 구성의 예시도.
- 도 2는 본 발명을 실시하기 위한 전체 시스템에 대한 구성도.
- 도 3은 본 발명의 일실시예에 따른 사전 패리티 방식을 사용하는 RAID-6 스토리지에 대한 구성도.
- 도 4는 본 발명의 일실시예에 따른 패리티 데이터(Parity data) 중복제거 방법을 설명하는 슈도코드.
- 도 5는 본 발명의 일실시예에 따른 패리티 디스크에 대한 중복제거 방법을 나타낸 블록도.
- 도 6은 본 발명의 일실시예에 따른 클라우드 블록 스토리지 환경에서의 패리티 디스크 중복제거 방법을 설명하는 예시도.
- 도 7은 본 발명의 실험에 따른 기존의 사전 패리티를 사용하는 방법과 패리티 디스크에 중복제거를 사용한 방법의 입출력 성능을 비교한 결과 그래프.
- 도 8은 본 발명의 실험에 따른 기존의 사전 패리티를 사용하는 방법과 패리티 디스크에 중복제거를 사용한 방법의 쓰기 횟수를 비교한 결과 그래프.
- 도 9는 본 발명의 실험에 따른 해시테이블을 위한 DRAM 사용량을 측정한 결과를 나타낸 그래프.
- 도 10은 본 발명의 실험에 따른 데이터를 복구하기 위한 디코딩 시간을 측정한 결과를 나타낸 그래프.

발명을 실시하기 위한 구체적인 내용

- [0014] 이하, 본 발명의 실시를 위한 구체적인 내용을 도면에 따라서 설명한다.
- [0015] 또한, 본 발명을 설명하는데 있어서 동일 부분은 동일 부호를 붙이고, 그 반복 설명은 생략한다.

- [0016] 먼저, 본 발명에서 사용되는 레이드(RAID)에 대하여 설명한다.
- [0017] 레이드(RAID)는 여러 개의 디스크를 사용하여 고성능 또는 고신뢰성을 갖는 디스크를 구성하는 방법이다. 일반적으로 클라우드 스토리지 시스템을 구축하는 경우에는 다수의 디스크를 사용하는 경우가 많으므로, 디스크의 고장에 대비하여 한 개의 디스크가 손상되어도 데이터의 복원이 가능한 RAID-5 또는 두 개의 디스크 고장까지 견딜 수 있는 RAID-6를 사용한다. 여기서 RAID-6의 경우 데이터 디스크가 N개 일 때 2개의 패리티(parity) 디스크를 추가적으로 사용해야 하므로 N+2개의 디스크를 사용하게 되며, 두 개의 패리티 데이터를 생성하는 복잡한 연산으로 인해 RAID-5에 비해 성능 면에서는 조금 떨어지지만 RAID-5에 비해 높은 신뢰성을 가지는 장점이 있다. 따라서, RAID-6는 다수의 디스크를 사용하는 환경에서 고성능과 고신뢰성을 갖는 디스크를 구현하기 위해 사용된다.
- [0018] 다음으로, 본 발명에서 사용되는 오픈스택 블록 스토리지에 대하여 도 1을 참조하여 설명한다. 도 1은 본 발명에서 사용되는 오픈스택 프레임워크에 대한 구성도이다.
- [0019] 도 1에서 보는 바와 같이, 오픈스택은 컴퓨팅 서비스를 제공하는 노바(Nova), 오브젝트 스토리지 서비스를 제공하는 스윙프트(Swift), 이미지 서비스를 제공하는 글랜스(Glance), 블록 스토리지 서비스를 제공하는 신더(Cinder) 등의 서비스 등으로 구성되어 있으며, 운영 네트워크(Management network)를 통해 서비스 컴포넌트들을 연동하여 클라우드 서비스를 제공한다.
- [0020] 컴퓨팅 서비스를 제공하는 노바(Nova)는 데이터의 입출력이 많은 스토리지 시스템과 빠른 처리속도를 요구하는 서버 등의 고성능 컴퓨팅 시스템 구축에 사용할 수 있다. 스토리지 시스템에서 노바(Nova)는 가상화된 스토리지 자원을 스케줄러를 통해 관리하며, 노바(Nova) 컴퓨트 노드에서 하이퍼바이저를 통해 서버를 가상화하고 가상화된 자원에 대한 컴퓨팅 작업을 진행한다.
- [0021] 또한 블록 스토리지 서비스를 제공하기 위한 신더(Cinder)는 데이터베이스 구성을 위한 스토리지, 블록 레벨 디스크로의 직접 접근을 제공하는 파일서버 등 입출력 성능에 민감한 시스템에 적합하다. 신더(Cinder)는 LVM(Logical Volume Manager)을 통해 여러 개의 물리 디스크들을 하나의 볼륨 그룹으로 묶어 블록 스토리지로써 사용하고, 이 블록 스토리지들은 가상화되어 사용자에게 제공된다.
- [0022] 다음으로, 본 발명에서 사용되는 총 플래시 메모리(All Flash Array)에 대하여 설명한다.
- [0023] 총 플래시 메모리(All Flash Array)는 플래시 메모리를 사용하는 SSD (Solid-State Drive)만으로 구성된 스토리지이다. SSD는 자기 디스크를 사용하는 하드디스크와는 달리 반도체를 이용하여 정보를 저장한다.[비특허문헌 3] 이는 순수 전자식으로 동작하기 때문에 기계적으로 헤드를 이동시켜 데이터를 읽어야 하는 하드디스크에 비해 탐색 시간이 매우 빠르다. 입/출력 속도 또한 순차 읽기/쓰기의 경우는 하드디스크의 몇 배, 랜덤 읽기/쓰기의 경우는 수십 배 이상 빠르다는 장점이 있다. 그러나, SSD는 하드디스크와는 달리 디스크를 구성하는 셀에 마모도가 존재하여 제한된 횟수만 쓸 수 있다는 단점이 있다.[비특허문헌]
- [0024] 다음으로, 본 발명에서 사용되는 데이터 중복제거 방법에 대하여 설명한다.
- [0025] 중복제거 방법은 입력되는 데이터에 해시 값을 부여하고 기존의 데이터와 비교하여 해시 값이 같으면 동일 데이터로 인식한다. 따라서 동일한 데이터는 디스크에 저장하기 전에 제거되므로, 공간 활용도를 높일 수 있다는 장점이 있다.
- [0026] 여기서 데이터의 중복 제거를 수행하는 방식에 따라 인라인 방식과 포스트 프로세스 방식으로 분류할 수 있다. 인라인 방식의 경우 데이터가 저장되기 전에 중복제거를 수행하는 방식으로 디스크 용량을 줄일 수 있다는 장점이 있지만, 데이터 입력이 많은 경우 해시 연산의 오버헤드로 인한 병목 현상이 발생하는 단점이 있다. 포스트 프로세스 방식의 경우 임시 디스크에 저장 후, 유휴(idle) 시간에 중복된 데이터를 찾아 제거해주는 방식이며, 공간 활용 면에서 단점을 가진다. SSD를 이용한 시스템의 경우 디스크에 저장되기 전에 제거하는 인라인 방식을 사용하여 쓰기 횟수를 감소시킬 수 있다.
- [0027] 다음으로, 본 발명의 일실시예에 따른 총 플래시 어레이 기반 오픈스택 클라우드 블록 스토리지에서의 패리티 데이터 중복제거 방법을 도 2를 참조하여 설명한다.
- [0028] 도 2에서 보는 바와 같이, 본 발명의 패리티 데이터 중복제거 방법은 (a) 데이터 수신 단계(S10), (b) 데이터 청크를 생성하는 단계(S20), (c) 데이터 청크에 대하여 중복제거를 하여 저장하는 단계(S30), (d) 각 데이터 청크에 대한 패리티 데이터를 생성하는 단계(S40), 및, (e) 패리티 데이터를 중복제거하여 저장하는 단계(S50)로

구성된다.

- [0029] 본 발명에서는 오픈스택의 신더(Cinder)에서 할당한 SSD들에 RAID를 적용하고, 이 중 패리티가 저장되는 디스크에 대해 중복제거를 적용한다.
- [0030] 앞서 설명한 바와 같이, 본 발명을 실시하기 위한 전체 시스템은 오픈스택 프레임워크에 의한 시스템으로서, 신더(Cinder)가 다수의 SSD 스토리지(또는 총플래시 어레이)를 이용하여 저장장치(또는 스토리지)를 구성한다. 그리고 이하에서 설명하듯이, RAID-6 방식을 적용하여, SSD 스토리지를 데이터 디스크와 패리티 디스크로 구분하여, 사전 패리티 방식으로 구성한다.
- [0031] 즉, 소거 코딩(Erasure Codes)을 이용하여 패리티 정보를 저장하는 경우, 청크 단위로 패리티 데이터를 생성하고, 청크 단위로 데이터 및 패리티 데이터를 저장한다. 도 3에서 보는 바와 같이, 데이터 청크와 패리티 청크(또는 패리티 데이터)는 스트라이프 방식으로 각각의 데이터 노드(또는 SSD 디스크)에 저장된다. 바람직하게는, 청크 크기의 범위는 32~128KB이다. RAID에서 스트라이프 방식은 데이터를 청크 사이즈 단위로 나누고 RAID 수만큼의 청크를 하나의 세트로 구성하여 저장하는 방법이다. SSD 디스크의 손상 등에 의해 데이터가 손실되는 경우, 다른 데이터 노드(또는 SSD 디스크)에 복제된 데이터 청크를 이용하거나, 패리티 데이터를 이용하여 데이터를 복구할 수 있다.
- [0032] 먼저, 패리티 데이터 중복제거에 대하여 보다 구체적으로 설명한다.
- [0033] RAID가 적용된 스토리지에 쓰기 연산 발생 시 블록에 대한 쓰기 작업이 일어나고 추가적으로 패리티 디스크에 대한 쓰기 작업이 발생한다. 이는 읽기(Read)-수정(Modify)-쓰기(Write) 연산을 수행하여 패리티 데이터를 갱신하게 되는데, 작은 데이터의 쓰기 연산 시에도 패리티 데이터의 갱신이 같이 이루어져야 하므로 패리티 디스크가 전체 RAID가 적용된 디스크에 대한 병목지점이 된다.
- [0034] 따라서 기존의 RAID-6 방식에서는 지속적인 패리티 갱신으로 인해 패리티 디스크에 쓰기 연산이 집중된다. 이를 방지하기 위해 도 3과 같이 사전 패리티를 이용한 방식을 사용한다. 이는 패리티 데이터를 여러 디스크에 걸쳐 저장하므로, 쓰기 연산 시 갱신해야 하는 패리티 데이터로 인한 디스크의 마모를 RAID가 적용된 전체의 디스크에 분산시킬 수 있다는 장점이 있다. 그러나 이는 디스크에 대한 쓰기횟수를 감소시키지는 못하므로 디스크 전체에 대한 마모도는 패리티를 저장하기 위한 독립적인 디스크를 사용하는 방법과 같다.
- [0035] 따라서 디스크에 대한 쓰기횟수를 줄이기 위하여 데이터 중복제거를 적용해야 하는데, 기존의 RAID에서는 각각의 디스크가 데이터와 패리티를 저장하기 위해 사용되므로 RAID로 사용될 디스크 전체에 대해 중복제거를 적용해야 한다. 이는 디스크로 들어오는 모든 데이터에 대해 해시합수를 적용하여 핑거프린트를 생성해야 하므로 많은 연산을 필요로 하고, 해시테이블이 차지하는 DRAM 용량이 증가하게 된다.
- [0036] 또한 모든 디스크에 대해 중복제거를 적용해야 하므로 중복제거를 적용한 디스크의 수에 비례하여 CPU, DRAM 자원을 필요로 한다는 단점이 있으며, 클라우드 환경에서는 사용자가 요청하는 만큼의 디스크를 바로 제공하기 위해 대규모의 스토리지 시스템을 구축하기 때문에 이러한 단점이 더욱 부각된다.
- [0037] 따라서, 이보다 효율적으로 RAID 디스크에 대한 쓰기 횟수를 감소시키기 위해 패리티 디스크에 대한 중복제거를 적용할 수 있다. 도 4의 표는 RAID 디스크에서의 패리티 데이터 중복제거의 알고리즘을 나타내고, 도 5는 RAID가 적용된 스토리지들 중에서 패리티 디스크에 대해 중복제거가 적용된 모습을 나타낸다.
- [0038] 도 4의 표에서 Generate_parity() 함수는 RAID 디스크에서 패리티 데이터(Reed solomon, even odd, etc.)를 생성하는 함수이고, Generate_hash_value() 함수는 중복제거를 위해 해시 값(SHA-1, murmur3, etc.)을 생성하는 함수이다.
- [0039] 패리티 디스크에 대해 중복제거를 적용하지 않았을 때는, 도 5와 같은 구조의 RAID 디스크에서 패리티 데이터 저장 시 패리티 디스크가 전체 RAID가 적용된 디스크에 대한 병목지점이 된다. 그러나 패리티 디스크에 대해 중복제거를 적용한 경우에 중복된 데이터가 많은 데이터에 대한 쓰기 연산 시에는 중복되는 패리티 데이터에 대한 쓰기연산을 수행하지 않는다. 따라서 이때, 패리티 디스크에 쓰기연산이 집중적으로 수행되는 것을 방지하므로, RAID 디스크에서의 패리티 데이터 생성으로 인한 병목지점을 제거할 수 있다. 또한, RAID 디스크 전체에 대해 중복제거를 적용하는 방법에 비해 해시 값 계산을 위한 CPU 연산량과 해시테이블을 위한 DRAM 사용량을 효율적으로 감소시킬 수 있다. 또한 RAID를 구성하는 디스크의 스트라이프 크기에 비해 중복제거를 수행하기 위한 청크의 크기를 작게 함으로써, 갱신된 패리티 청크의 변경된 부분이 거의 없을 때 디스크에 실제로 저장되는 패리티 데이터를 최소화할 수 있게 된다.

- [0040] 다음으로, 본 발명의 일실시예에 따른 패리티 데이터 중복제거 방법이 적용된 오픈스택 클라우드 블록 스토리지 시스템을 설명한다.
- [0041] 오픈스택의 신더(Cinder)는 노바(Nova)에서 생성하는 가상머신에 블록 스토리지를 제공하는 역할을 한다. 이는 SCSI 이니시에이터-타겟 구조로 연결되는데, 신더(Cinder) 블록 스토리지 노드가 스토리지를 제공하는 타겟 시스템이 되고 노바(Nova)에서 생성한 가상머신이 이니시에이터 시스템이 되어 신더(Cinder)에서 제공하는 스토리지를 받아 사용하게 된다. 플래시 메모리 기반의 SSD를 사용하는 오픈스택 블록 스토리지 시스템에 제안하는 패리티 데이터 중복제거를 적용하는 경우에는 SSD로의 쓰기 연산을 줄여 내부 셀의 마모를 효율적으로 감소시킬 수 있으며, 컴퓨트 노드에서 블록 스토리지 노드로 전송되는 데이터의 양이 감소하게 된다.
- [0042] 본 발명에서 제안하는 패리티 데이터 중복제거를 적용한 오픈스택 클라우드 블록 스토리지 시스템의 구조는 도 6과 같다.
- [0043] 오픈스택 클라우드 블록 스토리지 노드인 신더(Cinder)에서는 SSD를 가상화하여 노바(Nova) 컴퓨트 노드에서 생성된 가상머신에 제공하고, 노바(Nova) 컴퓨트 노드에서 생성한 가상머신에서는 제공받은 가상의 디스크에 대해 입출력 작업을 수행할 때 패리티 데이터의 중복제거를 실시한다.
- [0044] 다음으로, 본 발명의 효과를 실험을 통해 설명한다.
- [0045] 본 발명에서는 1개의 블록 스토리지 노드에서 총 6개의 SSD를 가상화하여 컴퓨트 노드에서 생성한 가상머신에 마운트하고, Jersasure 프로그램을 사용하여 RAID-6 디스크를 시뮬레이션한다. 여기서 기존의 방법인 사선 패리티를 이용하는 방법과 사선 패리티 방법을 사용하는 RAID-6 디스크에 데이터 중복제거를 적용한 방법, 패리티 디스크에만 데이터 중복제거를 적용한 경우 각각에 대하여 쓰기성능 및 디스크의 쓰기횟수, 디코딩 성능, DRAM 사용량을 측정하였다. 실험에 사용한 Workload는 웹에서[www.kernel.org] 다운로드 하였으며, 리눅스 커널 2.6.0 버전 ~ 2.6.39 버전에 해당하는 총 40개 버전의 소스 코드를 묶은 2.19GB 크기의 파일이며, 해시 값을 생성하기 위한 체크 사이즈는 4KB로 설정하였다.
- [0046] 도 7은 기존의 사선 패리티를 사용하는 방법과 패리티 디스크에 중복제거를 사용한 방법의 입출력 성능을 5회씩 반복하고, 평균을 계산하여 비교한 결과이다. 여기서 RAID-6 디스크의 스트라이프 크기는 각각 32KB, 1024KB로 설정하였다.
- [0047] 또한 도 8은 쓰기 횟수를 비교한 결과를 보인다.
- [0048] 기존의 사선 패리티를 사용한 RAID 디스크에 중복제거를 적용한 결과는 중복제거를 적용하지 않은 디스크에 비해 32KB, 1024KB 체크 사이즈에서 각각 75%, 89%의 성능만을 나타내었지만, 제안하는 패리티 중복제거를 적용한 결과는 32KB 체크 사이즈에서 95%, 1024KB에서 96%의 성능을 보였다. 이는 기존의 사선 패리티를 사용한 RAID 디스크에 중복제거를 적용한 경우 제안하는 패리티 디스크에 대한 중복제거 적용 시 보다 계산해야 할 해시 값이 많아 중복제거로 인한 오버헤드가 더 높기 때문이다.
- [0049] 기존의 사선 패리티를 사용한 RAID 디스크에 중복제거를 적용한 결과는 중복제거를 적용하지 않은 디스크에 비해 32KB, 1024KB 체크 사이즈에서 각각 75%, 89%의 성능만을 나타내었지만, 제안하는 패리티 중복제거를 적용한 결과는 32KB 체크 사이즈에서 95%, 1024KB에서 96%의 성능을 보였다. 이는 기존의 사선 패리티를 사용한 RAID 디스크에 중복제거를 적용한 경우 제안하는 패리티 디스크에 대한 중복제거 적용 시 보다 계산해야 할 해시 값이 많아 중복제거로 인한 오버헤드가 더 높기 때문이다.
- [0050] 또한 패리티 디스크로의 쓰기 횟수를 비교하면, 기존의 RAID 디스크에 중복제거를 적용한 경우 쓰기 횟수가 중복제거를 적용하지 않은 경우에 비해 32KB, 1024KB의 체크 사이즈에서 각각 8%, 13%만큼 낮았지만, 제안하는 패리티 중복제거를 적용한 방법은 쓰기 횟수가 각각 22%, 19%만큼 낮았다. 이는 제안하는 기법을 사용하는 경우 패리티 데이터의 중복만을 제거하지만, 기존의 중복제거 방식은 데이터와 패리티가 여러 디스크에 저장되기 때문에 디스크 당 중복제거의 효율이 떨어지기 때문이다.
- [0051] 도 9와 도 10은 각각 해시테이블을 위한 DRAM 사용량을 측정한 결과와, 데이터를 복구하기 위한 디코딩 시간을 측정한 결과이다.
- [0052] 해시테이블을 위한 DRAM 사용량은 제안하는 패리티 중복제거를 적용한 경우가 기존의 중복제거를 적용한 경우에 비해 32KB, 1024KB 체크 사이즈에서 각각 72%, 69%만큼 적었다. 이는 제안하는 방법이 RAID 디스크 중 갱신이 자주 일어나는 패리티 데이터의 중복만 제거하지만, 기존의 방법은 일반 데이터와 패리티 데이터에 대한 해시 값을 동시에 해시테이블에 유지하기 때문이다. 또한 디코딩 성능의 경우에는 중복제거를 적용하지 않은 RAID 디

스크가 가장 높았으며, 기존의 중복제거 방법을 적용한 RAID 디스크의 경우 중복제거를 적용하지 않은 RAID 디스크보다 32KB, 1024KB 청크 사이즈에서 각각 4%, 5% 많은 시간을 필요로 하였으며 제안하는 패리티 중복제거를 적용한 RAID 디스크의 경우 7%, 11% 더 많은 시간을 소모하였다. 이는 제안하는 중복제거 방법이 더 많은 중복된 데이터를 제거하였기 때문에, 데이터 복원 시 참조해야 할 데이터가 더 많기 때문이다.

[0053] 이상, 본 발명자에 의해서 이루어진 발명을 실시 예에 따라 구체적으로 설명하였지만, 본 발명은 실시 예에 한정되는 것은 아니고, 그 요지를 이탈하지 않는 범위에서 여러 가지로 변경 가능한 것은 물론이다.

[0054]

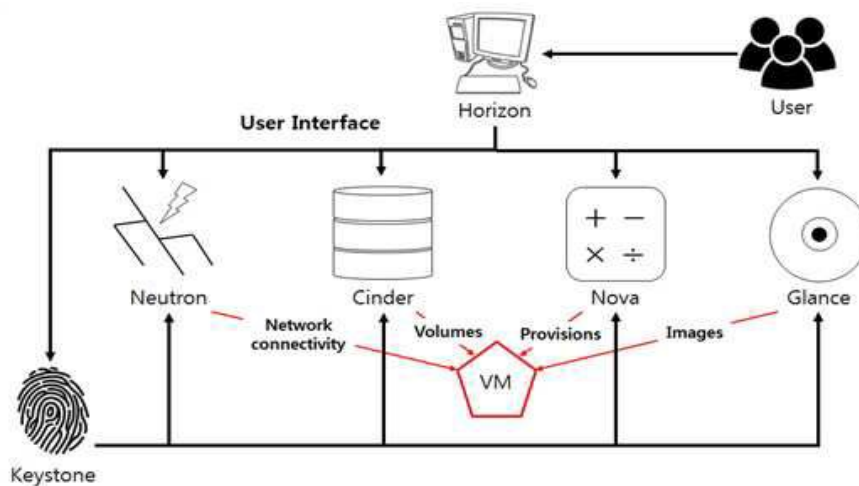
부호의 설명

[0055]

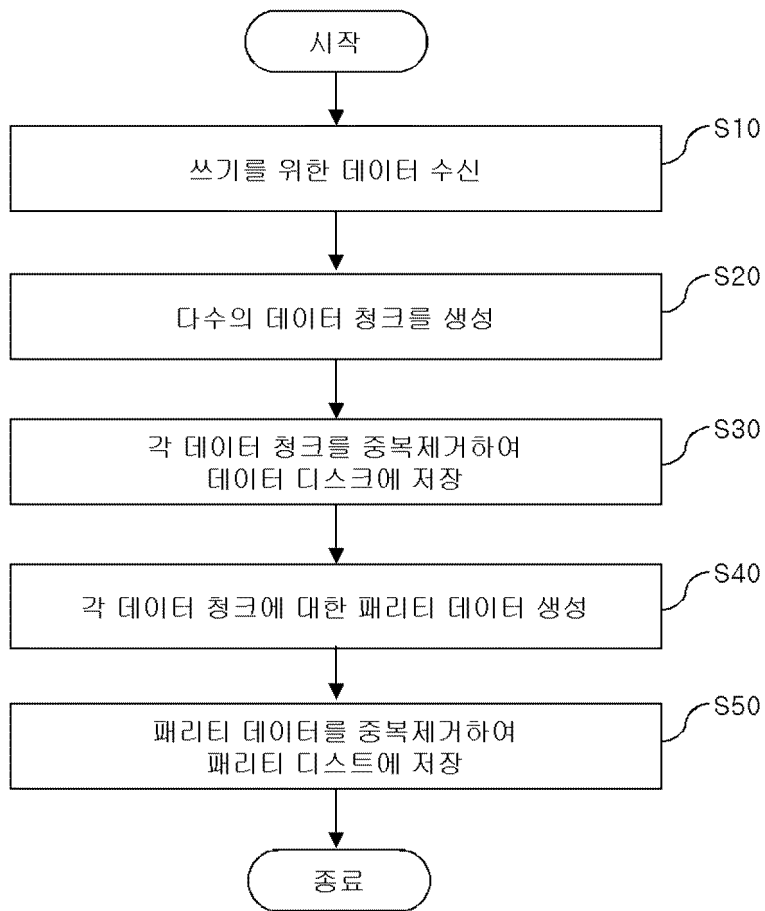
10 : 스토리지폴 관리부	11 : 가상머신 스토리지
12 : 총플래시 어레이	20 : 논리구조 설정부
21 : RAID 설정부	22 : 논리볼륨 설정부
23 : 파일시스템 설정부	30 : 성능 모니터부
40 : 파라미터 매니저	

도면

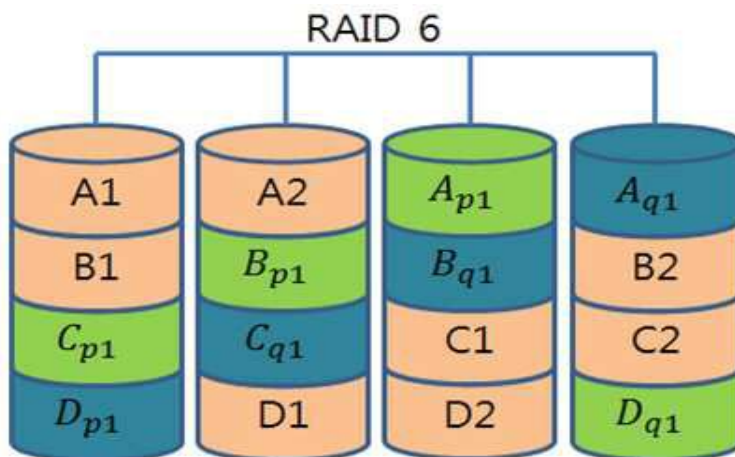
도면1



도면2



도면3



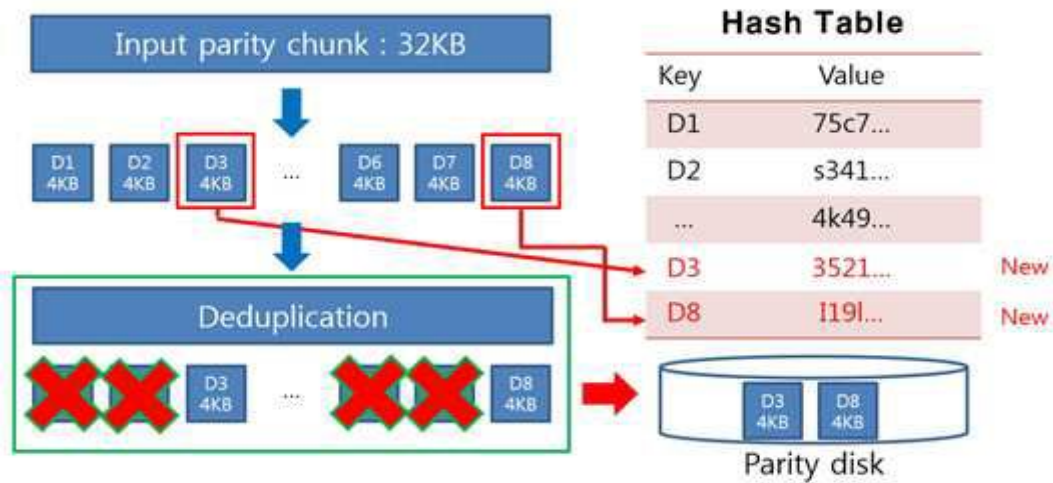
도면4

k : The number of Data disks
 m : The number of Parity disks
 F : Workload size
 n : RAID chunk size
 c : Chunk size for generating hash value
 $\beta = [F / (k * n)] + 1$: The number of iteration
 CH_new : New data array
 CH_old : Old data array
 Parity : Parity data array

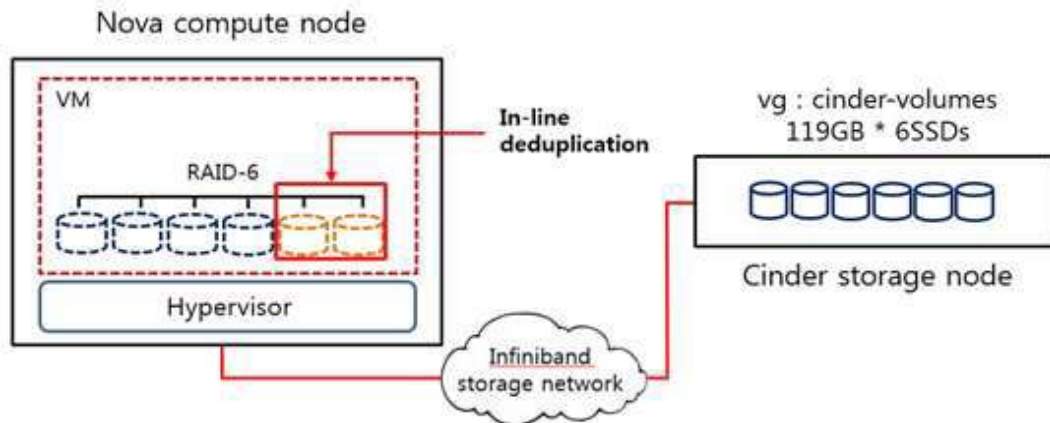
```

    /* Read-Modify-Write operation */
1.  for (j=1; j ≤ β; j++)
2.    for (i=1; i ≤ k; i++)
3.      read New data from host and allocate it into CH_new[i][j]
4.      read Old data from disks and allocate it into CH_old[i][j]
5.    end for
6.    for (i=1; i ≤ m; i++)
7.      read parity data from disks and allocate it into Parity[i][j]
8.    end for
9.    parity[i][j] = Generate_parity(CH_new[i][j], CH_old[i][j],
                                   Parity[i][j])
10. end for
    /* Write new data into data disks */
11. for (j=1; j ≤ β; j++)
12.   for (i=1; i ≤ k; i++)
13.     Write CH_new[i][j] into data disks
14.   end for
15. end for
    /* Parity data de-duplication */
16. for(j=1; j ≤ β; j++)
17.   for(i=1; i ≤ m; i++)
18.     for(l=1; l ≤ n/c; l++)
19.       hashvalue =Generate_hash_value(Parity[i][j][l])
20.       if ( hashvalue isn't exist in hashtable )
21.         add hashvalue into hashtable
22.         write Parity[i][j][l] into parity disks
23.       end if
24.     end for
25.   end for
26. end for
  
```

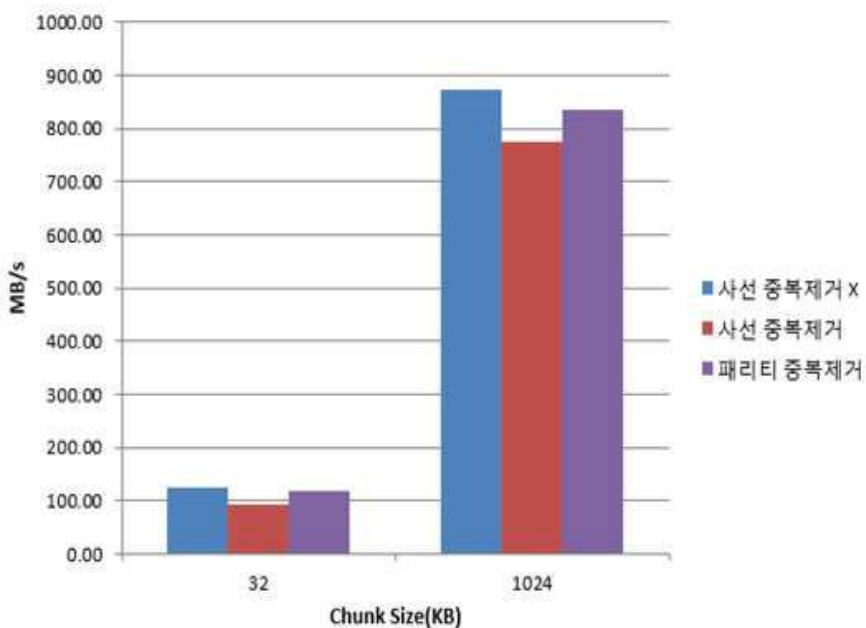
도면5



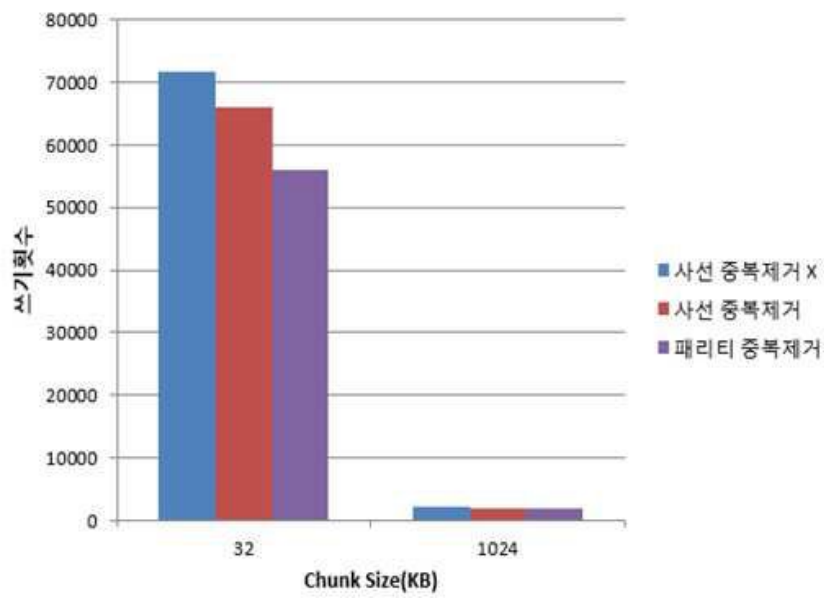
도면6



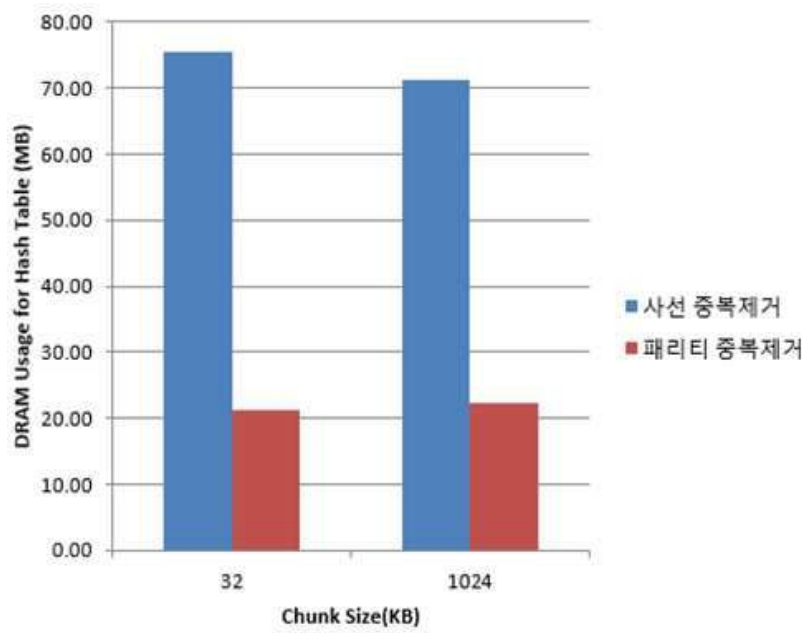
도면7



도면8



도면9



도면10

