(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2018-0043888
(43) 공개일자 2018년05월02일

(51) 국제특허분류(Int. Cl.)
G06F 11/10 (2006.01)
(52) CPC특허분류
G06F 11/1076 (2013.01)
G06F 2211/105 (2013.01)
(21) 출원번호 10-2016-0136613
(22) 출원일자 2016년10월20일
심사청구일자 2016년10월20일

(71) 출원인
인하대학교 산학협력단
인천광역시 남구 인하로 100, 인하대학교 (용현동)
(72) 발명자
김덕환
서울특별시 양천구 목동로 230, 103동 1004호 (목동, 아이파크아파트)
송태건
인천광역시 부평구 화랑북로 9-6 (산곡동)
(74) 대리인
특허법인 정안

전체 청구항 수 : 총 5 항

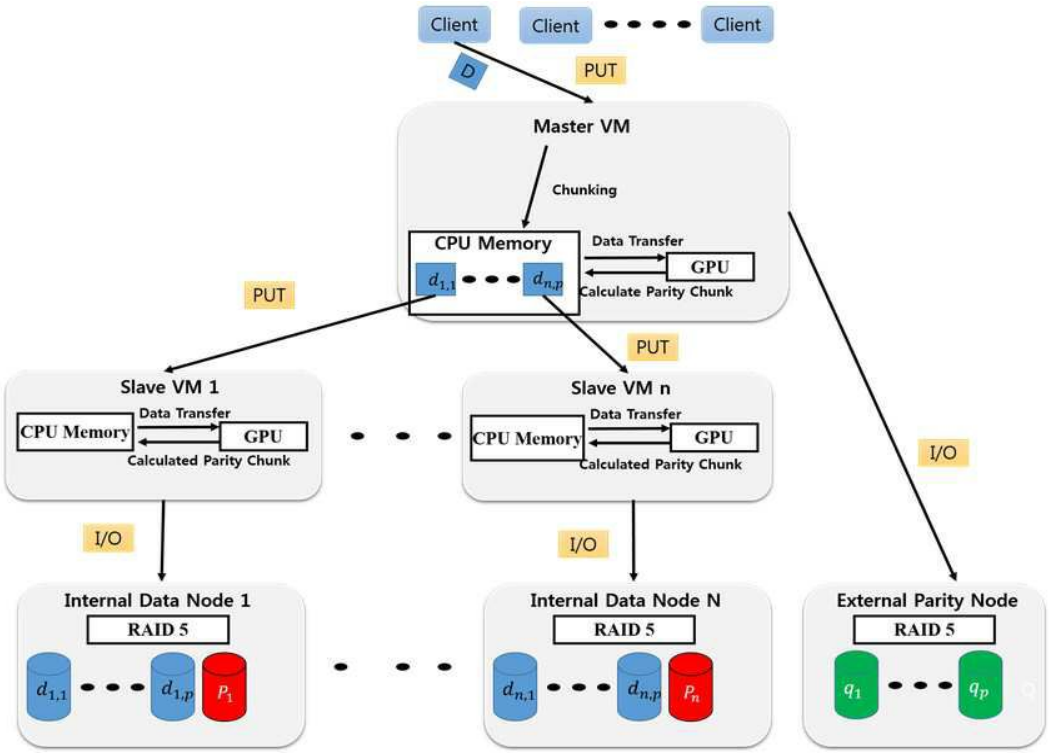
(54) 발명의 명칭 **다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템**

(57) 요 약

본 발명은 쓰기작업이 요청된 데이터의 패리티를 생성하여, 요청된 데이터와 함께 패리티를 저장하는, 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 관한 것으로서, CPU; 패스쓰루 GPU; 다수의 저장매체로 구성되어, 요청된 데이터 및, 패리티가 저장되는, 다수의 데이터 노드; 상기 요청된 데

(뒷면에 계속)

대 표 도 - 도3



이터를 다수의 데이터 청크로 청킹하되, 상기 요청된 데이터가 N개(N은 자연수)로 그룹화되어 분할되도록 청킹하고, 상기 데이터 청크 전체의 패리티(이하 외부 패리티)를 생성하여 저장하는, 마스터 가상머신; 및, 하나의 그룹에 속하는 데이터 청크의 패리티(이하 내부 패리티)를 생성하고, 해당 그룹의 데이터 청크 및, 내부 패리티를 하나의 데이터 노드에 저장하는, 슬레이브 가상머신을 포함하는 구성을 마련한다.

상기와 같은 시스템에 의하여, 다중 가상머신을 사용함으로써 데이터 카피 오버헤드를 줄이고, 고성능 인코딩할 수 있다.

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 이공학개인기초연구지원사업
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

명 세 서

청구범위

청구항 1

쓰기작업이 요청된 데이터의 패리티를 생성하여, 요청된 데이터와 함께 패리티를 저장하는, 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 있어서,

CPU;

패스쓰루 GPU;

다수의 저장매체로 구성되어, 요청된 데이터 및, 패리티가 저장되는, 다수의 데이터 노드;

상기 요청된 데이터를 다수의 데이터 청크로 청킹하되, 상기 요청된 데이터가 N개(N은 자연수)로 그룹화되어 분할되도록 청킹하고, 상기 데이터 청크 전체의 패리티(이하 외부 패리티)를 생성하여 저장하는, 마스터 가상머신; 및,

하나의 그룹에 속하는 데이터 청크의 패리티(이하 내부 패리티)를 생성하고, 해당 그룹의 데이터 청크 및, 내부 패리티를 하나의 데이터 노드에 저장하는, 슬레이브 가상머신을 포함하는 것을 특징으로 하는 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템.

청구항 2

제1항에 있어서,

상기 마스터 가상머신 및, 다수의 슬레이브 가상머신 각각은 상기 패스쓰루 GPU를 이용하여 독립적으로 데이터 청크를 병렬 연산을 통해 패리티를 생성하는 것을 특징으로 하는 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템.

청구항 3

제1항에 있어서,

상기 요청된 데이터는 상기 마스터 가상머신에서 n×p 데이터 청크 ($d_{1,1}$,…, $d_{n,p}$)로 청킹되고(단, n은 슬레이브 가상머신의 수이고, p는 상기 슬레이브 가상머신에 해당되는 데이터 노드의 저장매체의 개수임), i번째 슬레이브 가상머신은 데이터 청크 ($d_{i,1}$,…, $d_{i,p}$)에 대한 패리티를 생성하는 것을 특징으로 하는 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템.

청구항 4

제3항에 있어서,

상기 마스터 가상머신은 모든 청크 데이터의 각 행 ($d_{1,i}$,…, $d_{n,i}$)를 XOR하여 패리티 q_i를 생성하여, 전체적으로, p개의 외부 패리티 청크 (q_1 ,…, q_p)를 생성하고, 생성된 외부 패리티 청크를 하나의 데이터 노드에 저장하는 것을 특징으로 하는 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템.

청구항 5

제1항 내지 제3항 중 어느 한 항에 있어서,

상기 마스터 가상머신 또는 슬레이브 가상머신은 상기 데이터 청크를 XOR 연산하여, 패리티 데이터를 생성하는 것을 특징으로 하는 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템.

발명의 설명

기 술 분 야

[0001] 본 발명은 다중 가상머신 환경에서 패스쓰루(Pass-through) GPU를 사용하여 고성능으로 인코딩하는, 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 관한 것이다.

배 경 기 술

[0002] 최근 데이터의 양이 많아짐에 따라 적은 비용으로 스케일러블 스토리지(Scalable Storage)를 제공하기에 적합한 클라우드 스토리지 시스템(Cloud Storage System)에 관한 연구가 활발히 진행 중이다. 클라우드 스토리지 시스템(Cloud Storage System)에서는 가상머신을 사용하는 다수의 유저에게 고성능, 고효율의 스토리지를 제공하는 것과 데이터의 신뢰성을 보장해주는 것이 가장 큰 이슈이다.

[0003] NVIDIA 사에서 출시한 GPGPU(General-Purpose Graphic Processing)는 내장되어 있는 수백 개의 코어를 이용하여 대규모 병렬처리가 가능하며, 이를 이용한 고성능의 연구들이 진행 중이다[비특허문헌 1]. 클라우드 스토리지 시스템에서는 데이터의 신뢰성을 위해 디스크 페일(fail) 시 복구(reconstruction) 가능한 RAID(Redundant Array of Inexpensive Disk)형태로 구성된 스토리지를 사용한다.

[0004] 하지만 RAID의 경우 데이터의 신뢰성을 보장하는 반면, 에로 코드(Erasure Codes)를 이용한 패리티(Parity)의 생성은 많은 양의 XOR 연산을 필요로 한다. 이는 곧 CPU의 오버헤드(overhead)로 인해 데이터 병목현상 발생 및 클라우드 스토리지 시스템의 성능 감소를 일으킨다[비특허문헌 2]. 또한 클라우드 스토리지 시스템에서 자주 사용되는 RAID5, RAID6의 경우 RAID5는 최대 한 개, RAID6는 최대 두 개의 디스크 페일(disk fail) 시 복구(reconstruction) 가능 하다. 이로 인해 데이터(data)가 많아질수록 단일 RAID로 구성된 스토리지로는 신뢰성을 보장하는데 한계가 있다.

[0005] 최근 RAID 에서 이러한 CPU 오버헤드(overhead) 문제와 신뢰성 문제를 해결하기 위해 RAID51, RAID61 과 같은 계층 RAID를 구성하여 데이터(data)의 고 신뢰성을 위한 연구 및 RAID에 GPU를 적용한 고속 패리티(Parity) 생성에 관한 연구들이 진행 중이다[비특허문헌 3], [비특허문헌 4]. 하지만 기존의 계층 RAID에서 RAID1을 구성하는 매니저(Manager)의 경우 다수의 유저에 의해서 생겨난 모든 데이터(data)의 복제본을 만들어 저장하기 때문에 비효율적으로 스토리지를 사용하는 단점이 있다. 또한, 기존 계층 RAID에서 단일 GPU를 이용한 지속적인 패리티(Parity) 생성은 메인 메모리(Main Memory)와 GPU 메모리(Memory)로의 잦은 데이터(data) 전송으로 성능이 감소하는 단점이 있다[비특허문헌 5].

선행기술문헌

비특허문헌

[0006] (비특허문헌 0001) Feng, Xiaowen, et al. "A segment-based sparse matrix-vector multiplication on CUDA." Concurrency and Computation, 2014, pp,271-286

(비특허문헌 0002) Wryzykowski, Roman, LuKasz Kuczynski, and Marcin Wozniak. " Towards efficient execution of erasure codes on multicore architectures." Applied Parallel and Scientific Computing, Springer Berlin Heidelberg, 2012. pp.357-367

(비특허문헌 0003) Khasymski, Aleksandr, et al. "On the use of GPUs in realizing cost-effective distributed RAID." Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2012

(비특허문헌 0004) Thomasian, Alexander, and Yujie Tang. "Hierarchical RAID: Organization, Operation, Reliability and Performance." Proc. 7th IEEE Int'l Workshop on Storage Network Architecture and Parallel I/O (SNAPI2011). 2011.

(비특허문헌 0005) Wyrzykowski, Roman, Marcin WoŁźniak, and Lukasz Kuczynℓski. "Efficient Execution of Erasure Codes on AMD APU Architecture." Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2014. 613-621.

(비특허문헌 0006) M.Pirahandeh and D.H kim,"Energy-Aware GPU-RAID Scheduling for Reducing Energy Consumption in Cloud Storage System", In Computer Science and its Applications Spring Lecture Notes in Electrical Engineering, 330, pp.705-711, 2015

(비특허문헌 0007) Curry, Matthew L., et al. "Arbitrary dimension reed-solomon coding and decoding for extended raid on gpus." Petascale Data Storage Workshop, 2008. PDSW'08. 3rd. IEEE, 2008.

(비특허문헌 0008) 엔비디아 Pass-through GPU

발명의 내용

해결하려는 과제

[0007] 본 발명의 목적은 상술한 바와 같은 문제점을 해결하기 위한 것으로, 다중 가상머신 환경에서 패스쓰루(Pass-through) GPU를 사용하여 고성능으로 인코딩하는, 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템을 제공하는 것이다.

[0008]

과제의 해결 수단

[0009] 상기 목적을 달성하기 위해 본 발명은 쓰기작업이 요청된 데이터의 패리티를 생성하여, 요청된 데이터와 함께 패리티를 저장하는, 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 관한 것으로서, CPU; 패스쓰루 GPU; 다수의 저장매체로 구성되어, 요청된 데이터 및, 패리티가 저장되는, 다수의 데이터 노드; 상기 요청된 데이터를 다수의 데이터 청크로 청킹하되, 상기 요청된 데이터가 N개(N은 자연수)로 그룹화되어 분할되도록 청킹하고, 상기 데이터 청크 전체의 패리티(이하 외부 패리티)를 생성하여 저장하는, 마스터 가상머신; 및, 하나의 그룹에 속하는 데이터 청크의 패리티(이하 내부 패리티)를 생성하고, 해당 그룹의 데이터 청크 및, 내부 패리티를 하나의 데이터 노드에 저장하는, 슬레이브 가상머신을 포함하는 것을 특징으로 한다.

[0010] 또한, 본 발명은 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 있어서, 상기 마스터 가상머신 및, 다수의 슬레이브 가상머신 각각은 상기 패스쓰루 GPU를 이용하여 독립적으로 데이터 청크를 병렬 연산을 통해 패리티를 생성하는 것을 특징으로 한다.

[0011] 또한, 본 발명은 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 있어서, 상기 요청된 데이터는 상기 마스터 가상머신에서 n×p 데이터 청크 ($d_{1,1}$,…, $d_{n,p}$)로 청킹되고(단, n은 슬레이브 가상머신의 수이고, p는 상기 슬레이브 가상머신에 해당되는 데이터 노드의 저장매체의 개수임), i번째 슬레이브 가상머신은 데이터 청크 ($d_{i,1}$,…, $d_{i,p}$)에 대한 패리티를 생성하는 것을 특징으로 한다.

[0012] 또한, 본 발명은 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 있어서, 상기 마스터 가상머신은 모든 청크 데이터의 각 행 ($d_{1,i}$,…, $d_{n,i}$)를 XOR하여 패리티 q_i를 생성하여, 전체적으로, p개의 외부 패리티 청크 (q_1 ,…, q_p)를 생성하고, 생성된 외부 패리티 청크를 하나의 데이터 노드에 저장하는 것을 특징으로 한다.

[0013] 또한, 본 발명은 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생성 시스템에 있어서, 상기 마스터 가상머신 또는 슬레이브 가상머신은 상기 데이터 청크를 XOR 연산하여, 패리티 데이터를 생성하는 것을 특징으로 한다.

발명의 효과

[0014] 상술한 바와 같이, 본 발명에 따른 다중 가상머신 환경에서의 패스쓰루 GPU를 이용한 계층적 RAID의 패리티 생

성 시스템에 의하면, 다중 가상머신을 사용함으로써 데이터 카피 오버헤드를 줄이고, 고성능 인코딩할 수 있는 효과가 얻어진다.

도면의 간단한 설명

[0015] 도 1은 종래기술에 따른 GPU 기반 계층 RAID(Hierarchical RAID using GPU)의 구조를 나타낸 도면.

도 2는 본 발명의 일실시예에 따른 계층적 RAID의 패리티 생성 시스템의 구성을 나타낸 블록도.

도 3은 본 발명의 일실시예에 따른 계층적 RAID의 패리티 생성 방법을 나타낸 구조도.

도 4는 본 발명의 일실시예에 따른 SPC코드를 이용한 계층적 RAID55의 소거 코딩 스키마를 나타낸 도면.

도 5는 본 발명의 일실시예에 따른 패스쓰루 GPU를 이용한 마스터 가상머신의 패터리 생성 방법을 나타낸 슈도 코드.

도 6은 본 발명의 일실시예에 따른 패스쓰루 GPU를 이용한 슬레이브 가상머신의 패터리 생성 방법을 나타낸 슈도 코드.

도 7은 본 발명의 일실시예에 따른 코딩 매트릭스에 대한 예시도.

도 8은 본 발명의 실험에 따른 실험 환경을 나타낸 표.

도 9은 본 발명의 실험에 따른 가상머신의 개수에 따른 효과를 나타낸 그래프.

도 10는 본 발명의 실험에 따른 각 레이드(RAID)의 라이트 쓰루풋(write throughput)을 나타낸 그래프.

도 11은 본 발명의 실험에 따른 다중 가상머신 환경에서의 GPU를 이용한 패리티 생성 시간을 나타낸 표.

도 12은 본 발명의 실험에 따른 파일크기에 따른 종래기술과 본 발명의 전체 라이트 쓰루풋(write throughput)을 나타낸 표.

발명을 실시하기 위한 구체적인 내용

[0016] 이하, 본 발명의 실시를 위한 구체적인 내용을 도면에 따라서 설명한다.

[0017] 또한, 본 발명을 설명하는데 있어서 동일 부분은 동일 부호를 붙이고, 그 반복 설명은 생략한다.

[0018] 먼저, 본 발명에서 사용되는 패스쓰루(Pass-through) GPU에 대하여 설명한다.

[0019] 패스쓰루(Pass-through) GPU(Graphic processingr unit)는 물리적인 GPU를 가상머신에 1:1 연결을 해주는 방식이며, 하드웨어 자원(hardware resource)을 배타적으로 사용할 수 있도록 가상머신에 전용 장치를 제공해준다. 일반적인 가상화 GPU의 경우 가상머신에 GPU를 제공 시 하이퍼바이저(hypervisor)의 경유 및 스케줄링 과정으로 인해 성능의 저하가 일어난다. 하지만 패스쓰루(Pass-through) GPU의 경우 전용 GPU를 각 가상머신에 1:1 연결해줌으로써 전체 GPU의 성능 저하 없이 단일 가상머신에 1:1 할당 해줄 수 있다. 또한 다수의 가상머신에서 사용 시 스케줄링 과정으로써 성능이 하락하는 가상 GPU와는 다르게 패스쓰루(Pass-through) GPU의 경우 손실 없이 여러 가상머신에 할당이 가능하다[비특허문헌 8].

[0020] 즉, 패스쓰루 GPU는 물리적인 GPU를 가상머신에 1:1 연결 해주는 방식이다. 일반적인 가상화 GPU의 경우 가상머신들에게 GPU를 제공시 젠 하이퍼바이저(Xen hypervisor)에서 스케줄링 과정으로 인해 성능이 떨어지는 문제점이 존재한다. 하지만, 패스쓰루 GPU의 경우 단일 가상머신에 물리적인 GPU를 1대1로 매칭 시켜주는 방식이기 때문에 스케줄링과정이 없어 성능이 떨어지지 않는다.

[0021] 다음으로, 본 발명에서 사용하는 GPU 기반 계층 레이드(RAID)에 대하여 설명한다.

[0022] 도 1은 데이터(data)의 신뢰성과 고속 패리티(Parity) 생성을 위한 계층 RAID의 구조이다. 매니저(Manager)의 경우 신뢰성을 위해 RAID1으로 구성되었다. 데이터 노드(Data Node)에서는 고속의 패리티(Parity) 생성을 위해 GPU를 적용한 구조이다.

[0023] 프로세싱 과정은 다음과 같다. 수많은 유저에 의해서 생겨난 데이터(data)는 매니저(Manager)를 통해 신뢰성을 위해 모든 데이터(data)의 복제본을 저장한다. 또한, 데이터 노드(Data Node)에서는 GPU에서 패리티(Parity) 계산 후에 CPU로 데이터(data)를 보낸다. 이후 CPU에서는 RAID5로 구성된 스토리지에 데이터(data) 및 패리티(Parity)를 저장한다.

[0024] 도 1과 같은 기존 계층 RAID 구조의 경우 지속적으로 생겨나는 모든 데이터(data)의 복제본을 저장해야 함으로 비효율적으로 스토리지 공간을 사용하게 된다. 또한, GPU의 경우 메인 메모리(Main Memory)에서 GPU 메모리(Memory)로 데이터를 전송하는 시간과 연산 처리 결과 값을 GPU 메모리(Memory)에서 메인 메모리(Main Memory)로 전송시키는 오버헤드가 발생하는 단점이 있다.

[0025] 클라우드 스토리지 시스템은 다수의 유저에 의해 많은 데이터가 지속적으로 생겨난다. 이로 인해 기존과 같이 단일 GPU를 이용한 지속적인 패리티(Parity)의 생성은 메인 메모리(Main Memory)와 GPU 메모리의 잦은 데이터 전송으로 병목현상과 패리티(Parity) 생성 시 성능이 감소하게 된다. 기존의 계층 RAID51의 경우 데이터 미러링으로 인해 계층 RAID 55보다 더 많은 스토리지를 사용하는 단점이 존재한다. 또한, 기존 방법의 경우 디스크 장애시 복구는 가능하지만, 데이터 노드 전체의 장애시 복구가 불가능하다는 단점이 존재한다.

[0026] 다음으로, 본 발명의 일실시예에 따른 계층적 RAID의 패리티 생성 시스템의 구성에 대하여 도 2를 참조하여 설명한다.

[0027] 도 2에서 보는 바와 같이, 본 발명에 따른 계층적 RAID의 패리티 생성 시스템은 가상머신(30), CPU(10), 패스쓰루(Pass-through) GPU(20), 및, 저장매체(40)로 구성된다.

[0028] CPU(10)는 통상의 중앙처리장치(CPU)로서, 메인 메모리(11)를 구비한다. 메인 메모리(11)는 통상의 CPU와는 별도로 구성되나, 이하에서, 설명의 편의를 위하여, CPU(10)가 메인 메모리(11)를 구비하는 것으로 설명한다. 따라서 CPU(10)에서 GPU(20)로 데이터를 전송한다는 의미는, CPU(10)에서 처리하는 데이터를 GPU(20)에 직접 보낼 수도 있고, 메인 메모리(11)에 기록된 데이터를 GPU(20) 또는 GPU(20) 내의 GPU 메모리로 전송하는 것을 말한다.

[0029] 가상머신(30)은 CPU 등 하드웨어 장치 상에서 수행되는 프로그램 시스템으로서, 다중 가상머신으로 구성된다. 바람직하게는, 가상머신(30)은 마스터 가상머신(31) 및, 다수의 슬레이브 가상머신(32)으로 구성된다.

[0030] 즉, 마스터 가상머신(31) 및, 각 슬레이브 가상머신(32)는 CPU(11), 메인 메모리(10), 패스쓰루 GPU(20) 등 물리적인 장치를 공유하여 수행된다. 그러나 각 가상머신(31,32)은 하나의 독립적인 프로그램 시스템으로서 작동된다.

[0031] 또한, 앞서 설명한 바와 같이, 패스쓰루(Pass-through) GPU(20)는 물리적인 GPU를 가상머신(30)에 1:1 연결을 해주는 방식으로 구현된 그래픽 프로세서이다. 즉, 패스쓰루 GPU(20)는 GPU 코어나 GPU 메모리 등 하드웨어 자원(hardware resource)을 배타적으로 사용할 수 있도록, 각 가상머신(30)에 전용 장치로서 할당하여 제공해준다.

[0032] 저장매체(40)는 RAID(Redundant Array of Independent Disks)를 구현하기 위한 디스크 등 저장장치를 말한다. 이때, 데이터를 분산하기 위해 구성된, 하나의 물리적인 디스크(저장매체) 군을 노드로 표현한다. 따라서 도 2에서 보는 바와 같이, 저장매체(40)는 다수의 저장매체 노드(41)로 구성된다. 각 저장매체 노드(41)는 다시 RAID로 구성될 수 있다. 따라서 저장매체(40)는 계층적 구조의 RAID로 구성된다.

[0033] 또한, 각 저장매체 노드(41)는 각각 마스터 가상머신(31) 또는 슬레이브 가상머신(32)에 각각 할당된 노드이다. 즉, 각 가상머신(31,32)은 CPU(11)나 GPU(20)를 이용하여, 자신에게 해당되는 데이터 청크들에 대하여 패리티를 구하고, 구한 패리티 데이터 및, 청크 데이터를, 자신에게 할당된 저장매체 노드(41)에 저장한다.

[0034] 한편, 각 가상머신(31,32)은 통상적인 I/O를 이용하여, 메인 메모리 또는 GPU 메모리에 저장된 데이터를 저장매체(40)에 저장한다.

[0035] 또한, 마스터 가상머신(31)에서 사용하는 저장매체 노드(41)를 "내부 데이터 노드"라 부르기로 하고, 슬레이브 가상머신(32)에서 사용하는 저장매체 노드(41)를 "외부 패리티 노드"로 부르기로 한다.

[0036] 다음으로, 본 발명의 일실시예에 따른 계층적 RAID의 패리티 생성 시스템이 패리티(Parity)를 생성하는 방법을 도 3을 참조하여 설명한다. 도 3은 고속 패리티(Parity) 생성을 나타낸 구조로서, 패스쓰루(Pass-through) GPU를 적용한 다중(Multi) 가상머신(VM) 환경기반 계층 RAID에서 고성능 패리티(Parity) 생성 방법을 나타내고 있다. 즉, 도 3은 패스쓰루(Pass-through) GPU를 사용하는 계층 RAID55 의 고성능 인코딩을 보여준다.

[0037] 본 발명에 따른 방법은 데이터 노드 장애에도 복구 가능하며, 패스쓰루(Pass-through) GPU를 사용하여 각각의 가상머신이 독립적으로 청크(Chunking)된 데이터를 병렬연산을 통해 패리티를 생성할 수 있어서 GPU 오버헤드 감소를 통한 고성능 인코딩이 가능하다.

[0038] 도 3에서 보는 바와 같이, 패리티(Parity) 생성을 위한 구조는 크게 마스터 가상머신(Master vm)과 다수의 슬레이브 가상머신(Slave vm)들로 구성된다. 마스터 가상머신(Master VM)은 데이터의 신뢰성을 위해 패스쓰루(Pass-through) GPU를 이용해 모든 데이터의 패리티(Parity)인 외부 패리티(External Parity)를 생성 및 저장한다.

[0039] 이로 인해 기존의 데이터 복제본을 생성하여 저장하는 것보다 효율적인 스토리지 공간 활용이 가능하다. 슬레이브 가상머신(Slave VM)들은 마스터 가상머신(master VM)에서 청크(Chunking)된 데이터를 패스쓰루(Pass-through) GPU를 이용해 내부 패리티(Internal Parity)를 생성한다.

[0040] 전체 데이터를 청킹(Chunking)하여 각 슬레이브 가상머신(Slave VM)에서 다중 가상머신(Multi vm) 환경을 기반으로 패리티(Parity)를 생성함으로써 기존의 단일 GPU만을 이용한 패리티(Parity) 생성시의 잦은 데이터 전송으로 성능이 감소하는 부분의 해결이 가능하다.

[0041] 클라이언트에서 데이터 D가 쓰기(write)된 상황을 가정한다. 마스터 가상머신(vm)에 쓰기작업된 데이터는 n×p 데이터 청크(chunks) ($d_{1,1}$,…, $d_{n,p}$)로 청킹(Chunking)된다. 여기서, n은 슬레이브 가상머신(vm)의 수이고, p는 각 슬레이브 가상머신(vm)의 해당되는 디스크의 수이다.

[0042] 이때, n×p 데이터 청크의 p개($d_{i,1}$,…, $d_{i,p}$)를 그룹화하여, 하나의 그룹으로 형성하여, 각 그룹의 데이터 청크를 각 슬레이브 가상머신에 분산한다.

[0043] 마스터 가상머신(vm)은 모든 데이터 청크들을 그것에 해당하는 슬레이브 가상머신(vm)에 분산시킨다. 그런후 마스터 가상머신(vm)은 GPU를 이용하여 외부 패리티(external parity) (q_1 ,…, q_p)를 생성하여 저장한다.

[0044] 내부 데이터 노드(Internal data node)의 경우 패리티 디스크와 데이터 디스크로 구성된다. 슬레이브 가상머신(vm)에서는 청크(chunk)된 데이터 ($d_{i,1}$,…, $d_{i,p}$)를 GPU를 이용해 XOR 연산 함으로써 p_i 패리티 데이터를 생성하여 저장한다.

[0045] 다음으로, 본 발명의 일실시예에 따른 패스쓰루 GPU를 이용하여, 계층 RAID의 데이터를 인코딩하는 방법을 설명한다.

[0046] 도 4는 마스터 가상머신(vm)과 슬레이 가상머신(vm) 인코더에서 생성된 데이터와 패리티 데이터의 스키마(Scheme)를 보여준다.

[0047] 도 4에서 보는 바와 같이, 각 k번째 슬레이브 가상머신에서는 전체 데이터 청크 중에서 ($d_{k,1}$,…, $d_{k,p}$)의 청크 데이터를 XOR하여 패러터 p_k를 생성한다. 그리고 마스터 가상머신은 모든 청크 데이터의 행 ($d_{1,i}$,…, $d_{n,i}$)를 XOR하여 패리티 q_i를 생성한다.

[0048] 또한, 도 5의 테이블 1은 제안한 계층 RAID55에서의 마스터 가상머신(vm) 인코더 알고리즘을 보여준다. 라인(Line) 1의 경우와 같이, 먼저, 입력된 데이터는 n×p 데이터 청크(chunks) ($d_{1,1}$,…, $d_{n,p}$) 로 마스터 가상머신에서 청킹(chunking)된다. 다음으로, 라인(lines) 2-6과 같이, 마스터 가상머신의 인코더는 p개의 외부 패리티 청크(external parity chunk) (q_1 ,…, q_p)를 GPU를 이용해 병렬적으로 생성한다. 그리고 다음으로, 라인(line) 7과 같이, p개의 외부 패리티 청크(external parity chunk)는 CPU(또는 메인 메모리)에서 GPU 메모리로 전송되고 GPU 메모리에서 코딩 메트릭스 H를 이용해 패리티를 생성한다. GPU에서 생성된 패리티 데이터는 다시 CPU(또는 메인 메모리)로 전송되어 외부(External) 패리티 노드에 저장된다.

[0049] 코딩 메트릭스 H는 소거코드(Erasure Codes)를 이용해 패리티 데이터를 생성하기 위해 사용되는 메트릭스이다. 도 7에서의 H 부분이 이에 해당한다[비특허문헌 6]. 예를들어 알고리즘에 의해서 코딩 메트릭스 H의 첫 번째 행과 각각의 슬레이브 가상머신(Slave VM)에서 생성된 $d_{1,1}$....$d_{k,1}$,....,$d_{n,1}$을 XOR 연산해서 q_1을 생성하며 H의 p번째 행과 $d_{1,p}$,...,$d_{k,p}$,....$d_{n,p}$와의 XOR 연산을 해서 q_p 패리티를 생성하게 된다. $H_{j,i}$의 경우는 가장 안쪽의 for문을 기점으로 j가 n이 될 때까지 숫자가 증가하며, i의 경우는 j가 n까지 증가된 순간 1씩 증가되게 된다. 예를 들어, 만약 n을 10이라 가정했을 시 H 행렬의 가로크기는 10이며 $H_{1,1}$ $H_{2,1}$ $H_{3,1}$ ... $H_{10,2}$ ... $H_{1,2}$ $H_{2,2}$ $H_{3,2}$ //코딩메트릭스의 첫 번째 행을 뜻한다.

[0050] 다음으로, 도 6의 테이블 2는 제안한 계층 RAID55에서의 슬레이브 VM 인코더 알고리즘을 보여준다. 먼저, 라인(Line 1)과 같이, p 데이터 청크(data chunks) ($d_{k,1}$,…, $d_{k,p}$)는 슬레이브 가상머신 cpu에 전송받아진다. 마스

터 가상머신에 의해서. 라인(lines) 2-4와 같이, 슬레이브 가상머신의 인코더는 GPU에서 코딩 메트릭스 X를 이용해 P_k 내부(Internal) 패리티를 병렬적으로 생성한다. 다음으로, 라인 5와 같이, 내부(Internal) 패리티 청크는 GPU 메모리에서 CPU 메모리에 전송된다. 마지막으로 슬레이브 가상머신은 데이터 청크(data chunks) ($d_{k,1}$, …, $d_{k,p}$)와, 이로부터 생성된 내부(Internal) 패리티를 해당 내부 데이터 노드에 저장한다.

[0051] 코딩메트릭스 X는 코딩메트릭스 H와 같은 개념이며, H를 X로 표기하여 각각 다른 코딩 메트릭스를 사용한다는 것을 표현한 것이다. 결국 소거코드(Erasure Codes)를 이용해 패리티 데이터를 생성하기 위해 사용되는 메트릭스이다.

[0052] 코딩메트릭스 X의 첫 번째 행과 각 슬레이브 VM의 $d_{1,1}$ ... $d_{1,p}$를 XOR 연산해서 p_1을 생성하게 된다. X 코딩메트릭스 앞에 k는 슬레이브 해당 가상머신(vm)이며, i는 슬레이브 vm에 포함된 예를들어 $d_{k,1}$, $d_{k,p}$에서 p와 같은 데이터의 개수이다.

[0053] 다음으로, 본 발명의 효과를 위한 실험을 설명한다.

[0054] 먼저, 실험 환경(Experimental Environment)을 설명한다.

[0055] 본 발명에서의 실험은 Xen을 이용한 가상머신 환경에서 실험하였다. 실험을 위해 Samsung 840 Pro 128GB SSD를 이용하였다. 마스터 가상머신(master VM) 1개와 슬레이브 가상머신(Slave VM) 3개로 실험을 하였다. 또한 패스쓰루(Pass-through) GPU의 경우 Critx에서 제공되는 HDX 3D Pro를 이용해 마스터 가상머신(master VM), 슬레이브 가상머신(Slave VM) 모두에 적용하였다. 각 실험은 20회 반복해서 평균을 내어 측정하였다. 또한, 측정은 visual studio 에서 CUDA SDK 6.0을 이용하여 측정하였다. 제안하는 다중 가상머신(Multi VM) 환경에서 고속 패리티(Parity)생성을 위한 실험환경은 도 8의 표와 같다.

[0056] 다음으로, 실험결과(Experimental Result)에 대하여 설명한다.

[0057] 도 9은 가상머신(vm) 수와 파일크기(File size)에 따른 GPU의 패리티(Parity) 계산 성능을 나타낸다. 파일크기(File size)는 8M부터 512M 까지 다양하게 측정하였으며, 다중 가상머신(vm) 상황에서 멀티 패스쓰루(Pass-through) GPU 사용의 적합성을 판단하기 위해 측정하였다.

[0058] 측정은 각 가상머신에 1개부터 3개까지 GPU 벤치마크(Benchmark) 프로그램을 이용하여 GPU에 98%에 부하를 주었으며, 부하를 주지 않은 다른 1개의 가상머신(vm)에서 패리티(Parity) 계산 시간을 측정하였다. 실험에서 사용한 Grid k1 카드의 경우 패스쓰루(Pass-through) GPU를 최대 4개의 가상머신(vm)에 할당할 수 있기 때문에, 부하를 주는 가상머신(vm)의 개수를 1개부터 3개로 제한하였다. 실험결과 다수 가상머신(vm) 상황에서 평균 1.2% 정도의 미미한 성능 향상 및 하락이 있었다. 이는 패스쓰루(Pass-through) GPU를 적용한 Multi 가상머신(vm)환경에서 성능 감소 없이 패리티(Parity) 생성이 가능함을 보여준다.

[0059] 도 10는 single 가상머신(vm)에서 파일크기(File size)에 따른 RAID1, CPU 기반 RAID5, GPU 기반 RAID5, RAID0의 write throughput을 측정하였다. RAID0가 평균 557.02MB/s 로 가장 빠른 성능을 나타냈으며, GPU 기반 RAID5가 220.10MB/s, CPU 기반 RAID5가 133.44MB/s, RAID1 이 80.98MB/s의 성능을 보였다. 하지만 RAID1의 경우 빠른 속도를 보장하는 반면, 비효율적인 스토리지 사용과 데이터의 신뢰성을 보장할 수 없는 단점이 존재한다. 또한 CPU기반의 RAID5의 경우는 효율적인 스토리지 공간과 신뢰성을 보장하는 반면, 패리티(Parity) 계산으로 인한 cpu 오버헤드(overhead)로 GPU 기반의 RAID5보다 평균 26.9% 성능이 낮음을 보인다. GPU기반 RAID5의 경우 RAID0 보다 평균 56.02% 낮은 속도를 보이지만, RAID1과 CPU기반 RAID5 보다 빠른 write 속도를 보이며, RAID0 보다 데이터의 신뢰성과 스토리지의 효율 면에서 뛰어난 장점이 있다..

[0060] .

[0061] 다음 실험에서는Khasymski[비특허문헌 3] 방법과 본 발명에서 제안한 방법에서의 Write throughput 및 패리티(Parity) 계산시간을 비교하였다.

[0062] 도 11은 다중 가상머신 환경에서 Khasymski[비특허문헌 3] 방법의 패리티(Parity) 생성 시간과, 본 발명에서 제안한 방법의 패리티(Parity) 생성시간을 측정한 결과이다. 측정결과 제안한 방법이 평균 62.93% 패리티(Parity) 계산시 성능 향상을 보였다. 패리티(Parity) 계산 시 데이터가 커질수록 GPU 오버헤드(overhead)증가로 패리티(Parity) 계산 성능이 하락하는 Khasymski[비특허문헌 3] 방법과 달리 제안한 방법의 경우 데이터를 청킹(chunking)하여 세분화된 데이터(data)를 각 슬레이브 가상머신(Slave VM)에서 병렬계산함으로써, 계산 성능이

증가한 것을 볼 수 있다.

[0063] 도 12은 파일크기(File size)에 따른 Khasymski[비특허문헌 3] 방법과 본 발명에 따른 방법의 전체 write throughput을 보여준다. Khasymski[비특허문헌 3] 방법의 경우 data의 복제시간과 단일(single) 가상머신(vm)에서의 Parity 계산 시간 시 오버헤드(overhead)로 인해 제안한 방법에 비해 평균 50.11% 낮은 throughput을 보였다.

[0064] 본 발명에서는 패스쓰루(Pass-through) GPU를 적용한 다중 가상머신 환경에서 Multi GPU를 이용한 고속 패리티(Parity) 생성 및 효율적인 공간을 사용하는 계층 RAID 구조를 설명하였다. 본 발명에 따른 방법은 Multi GPU를 이용한 고속 패리티(Parity) 생성의 결과를 얻었다.

[0065] 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.

[0066]

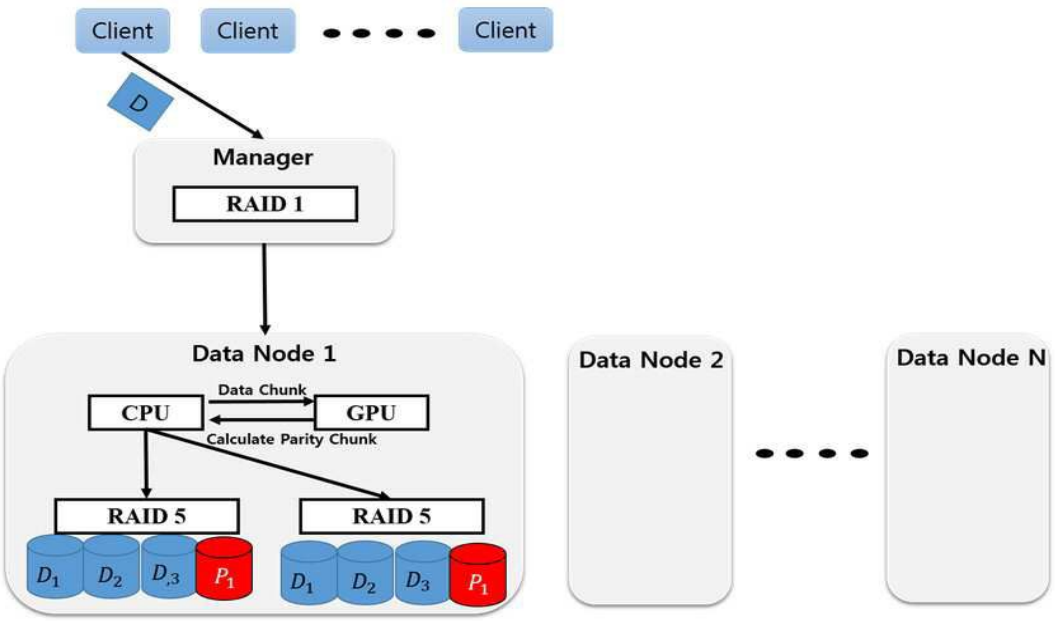
부호의 설명

[0067]

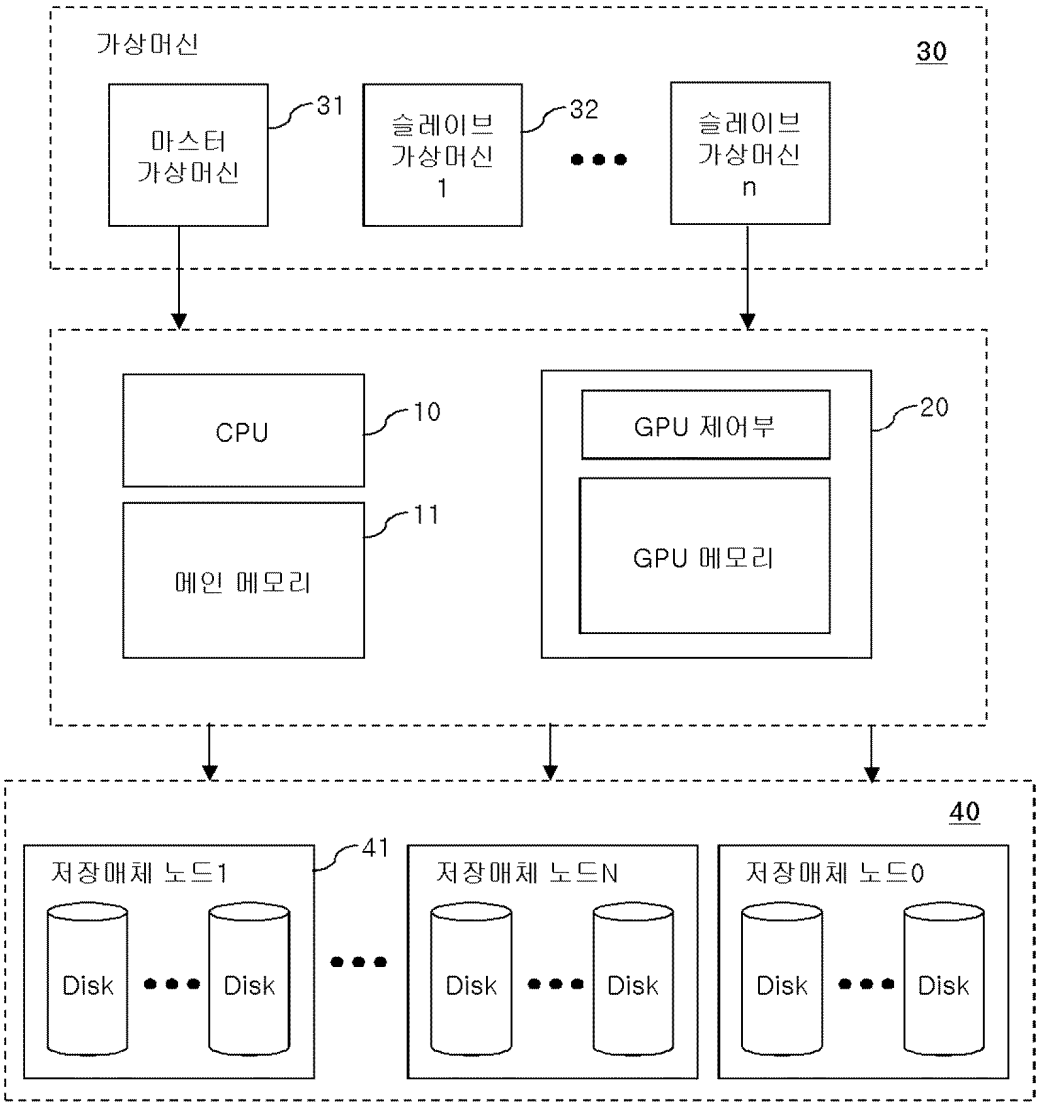
10 : CPU	11 : 메인 메모리
20 : GPU	30 : 가상머신
31 : 마스터 가상머신	32 : 슬레이브 가상머신
40 : 저장매체	41 : 저장매체 노드

도면

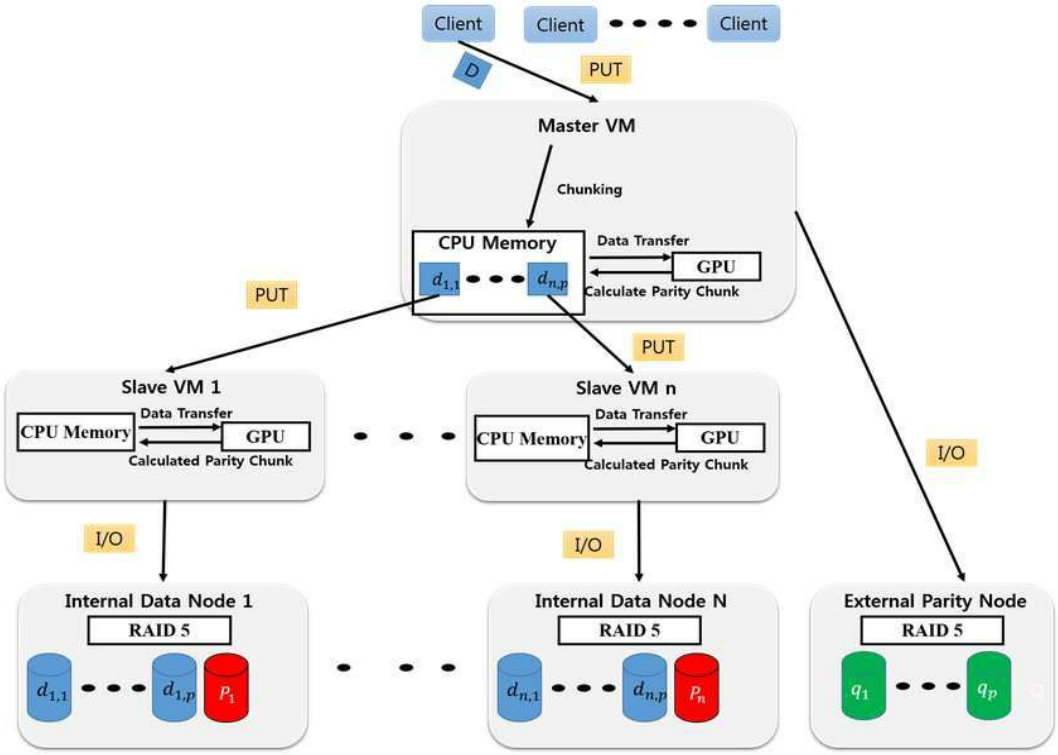
도면1



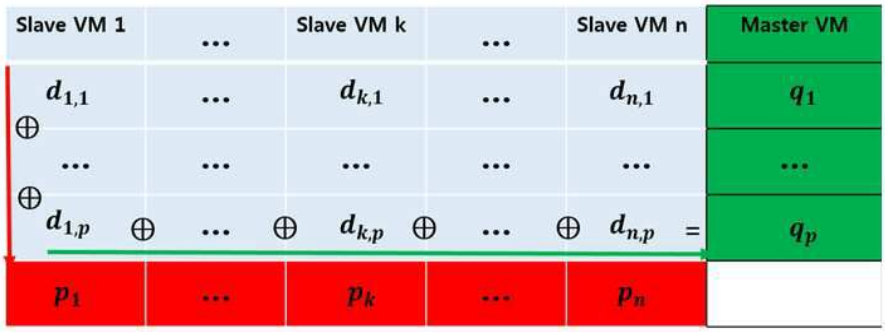
도면2


가상머신
30
마스터
가상머신
31
슬레이브
가상머신
1
32
슬레이브
가상머신
n
CPU
10
메인 메모리
11
GPU 제어부
20
GPU 메모리
40
저장매체 노드1
41
저장매체 노드N
저장매체 노드0
Disk
Disk
Disk
Disk
Disk
Disk

도면3


Client
Client
Client
D
PUT
Master VM
Chunking
CPU Memory
Data Transfer
GPU
Calculate Parity Chunk
PUT
PUT
Slave VM 1
CPU Memory
Data Transfer
GPU
Calculated Parity Chunk
Slave VM n
CPU Memory
Data Transfer
GPU
Calculated Parity Chunk
I/O
I/O
I/O
Internal Data Node 1
RAID 5
Internal Data Node N
RAID 5
External Parity Node
RAID 5

도면4


Slave VM 1
Slave VM k
Slave VM n
Master VM

도면5

Master_VM_Encoder($d[n][p]$, n, p)

Input:
$d[n][p]$: Two dimensional data chunks array.
n: the number of slave VMs.
p: the number of disk in each slave VM.
$q[p]$: One dimensional array of the external parity chunks

*/ Data chunking at master VM */
1. **read** data from host and **allocate** $d[n][p]$ chunks into CPU memory.
*/ External parity data generation using pass-through GPU */
2. **for** (i = 1; $i \leq p$; i++) do {
——— Master VM GPU kernel ———
3. **for** (j = 1; $j \leq n$; j++) do in parallel {
4. $q_i = q_i \oplus (\boldsymbol{H}_{j,i} \times \boldsymbol{d}_{j,i})$.
5. }
6. }
7. **transfer** external parities $q_{1,\ldots,}$ q_p to CPU memory;
8. **write** external parities $q_{1,\ldots,}$ q_p to external parity node.

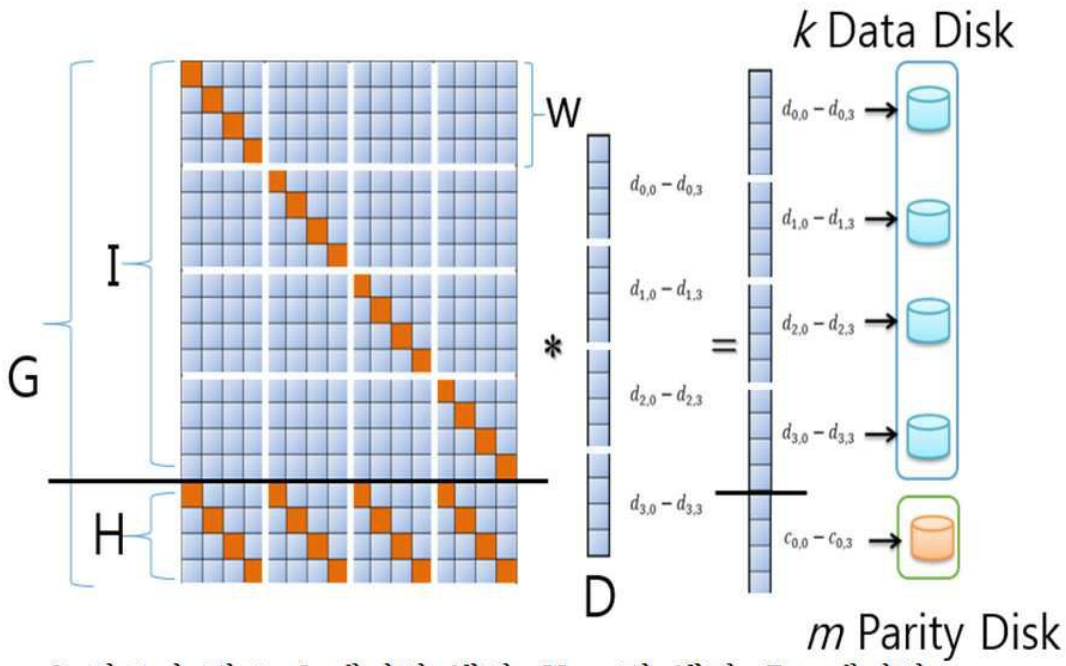
도면6

Slave_VM_Encoder($d[n][p]$, k, p)

Input:
$d[n][p]$: Two dimensional data chunks array.
k: the number of given slave VM.
p: the number of disk in each slave VM.
$P[k]$: An internal parity chunk

*/ Data transfer at slave VM */
1. **copy** $d[k][p]$ chunks from master VM to slave VM CPU memory.
*/ Internal parity data generation using pass-through GPU */
——— Slave VM GPU kernel ———
2. **for** (j = 1; $j \leq p$; j++) do in parallel {
3. $P_k = P_k \oplus (\boldsymbol{X}_{k,j} \times \boldsymbol{d}_{k,j})$.
4. }
}
5. **transfer** internal parity P_k to CPU memory;
6. **write** internal parity P_k to parity disk.
7. **write** data chunks $d_{k,1}$,…, $d_{k,p}$ to data disks.

도면7



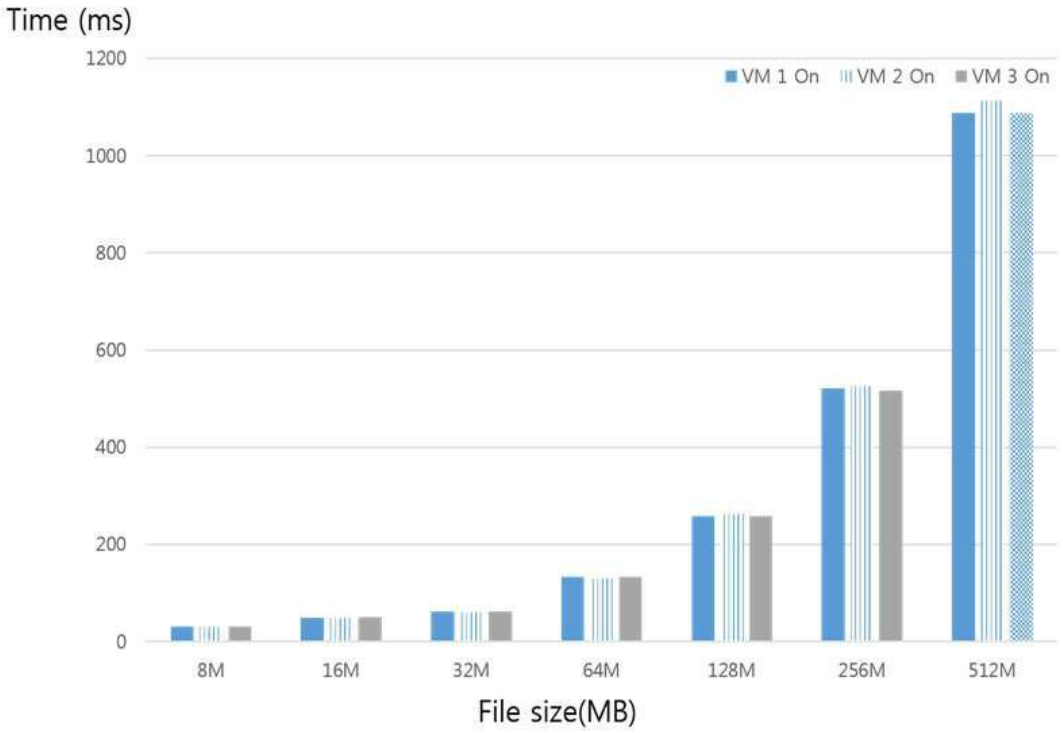
G=갈로아 필드; I=데이터 행렬; H=코딩 행렬; D= 데이터; w = word;

k= 데이터 디스크의 수; m=Parity 디스크의 수

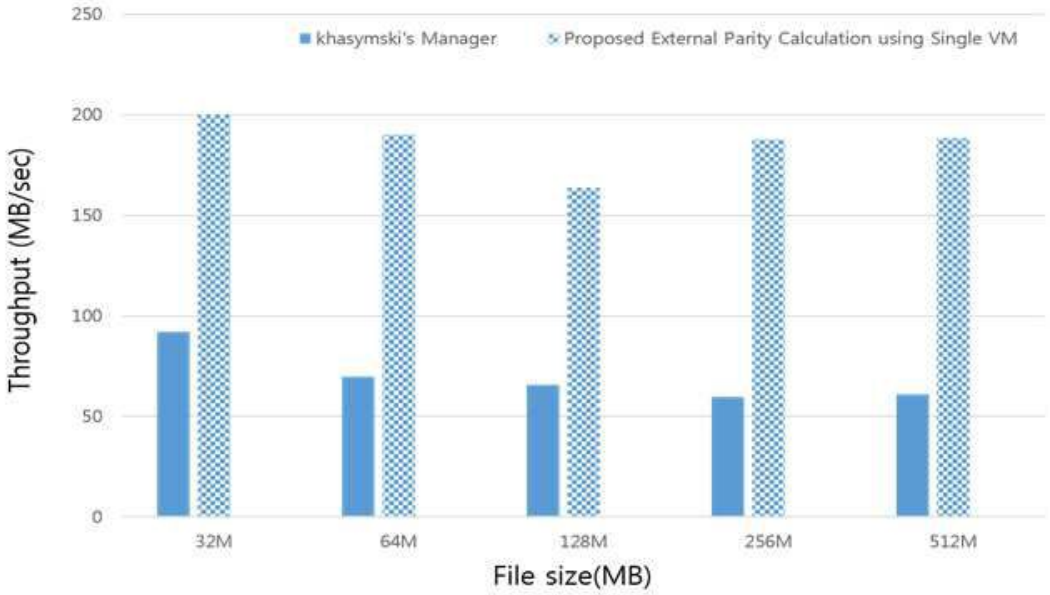
도면8

CPU	INTEL(R) XEON(TM) CPU E5-2620
MAIN MEMORY	DDR3 4GB
GPU	NVIDIA GRID K1
OS	WINDOWS SERVER 2008 R2
GPU MEMORY	4GB

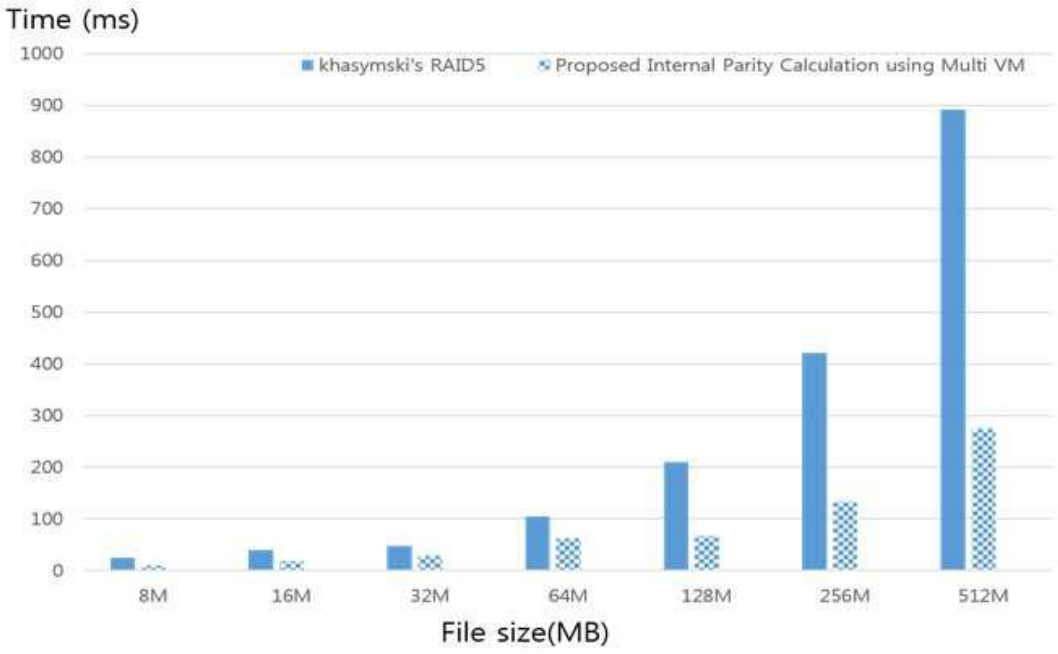
도면9


Time (ms)
1200
1000
800
600
400
200
0
VM 1 On
VM 2 On
VM 3 On
8M
16M
32M
64M
128M
256M
512M
File size(MB)

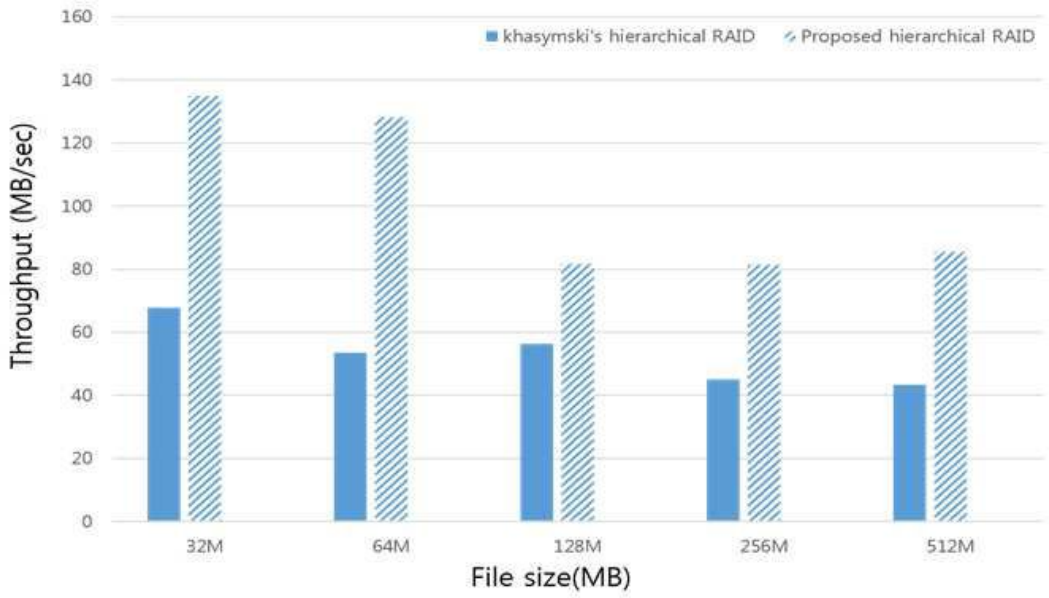
도면10


250
200
150
100
50
0
Throughput (MB/sec)
khasymski's Manager
Proposed External Parity Calculation using Single VM
32M
64M
128M
256M
512M
File size(MB)

도면11


Time (ms)
1000
900
800
700
600
500
400
300
200
100
0
khasymski's RAID5
Proposed Internal Parity Calculation using Multi VM
8M
16M
32M
64M
128M
256M
512M
File size(MB)

도면12


160
140
120
100
80
60
40
20
0
Throughput (MB/sec)
khasymski's hierarchical RAID
Proposed hierarchical RAID
32M
64M
128M
256M
512M
File size(MB)