



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2023-0077220
(43) 공개일자 2023년06월01일

(51) 국제특허분류(Int. Cl.)
G06F 11/10 (2006.01) H03M 13/09 (2015.01)
(52) CPC특허분류
G06F 11/1076 (2013.01)
G06F 11/1012 (2013.01)
(21) 출원번호 10-2021-0164251
(22) 출원일자 2021년11월25일
심사청구일자 2021년11월25일

(71) 출원인
인하대학교 산학협력단
인천광역시 미추홀구 인하로 100(용현동, 인하대학교)
(72) 발명자
김덕환
서울특별시 양천구 목동로 230, 103동 1004호(목동, 아이파크아파트)
메히디
인천광역시 미추홀구 경인남길 102(용현동)
(74) 대리인
양성보

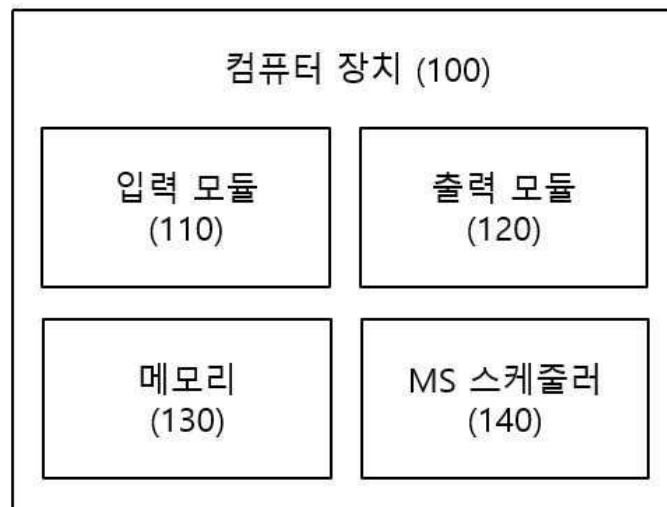
전체 청구항 수 : 총 10 항

(54) 발명의 명칭 클라우드 스토리지 시스템을 위한 고성능 회귀 AVX-2 패리티 인코딩 디코딩의 컴퓨터 장치, 및 그의 방법

(57) 요약

다양한 실시예들은 클라우드 스토리지 시스템을 위한 고성능 회귀 AVX-2 패리티 인코딩 디코딩의 컴퓨터 장치, 및 그의 방법에 관한 것으로, 컴퓨터 장치는 다중 섹터(multisector; MS) 스케줄러를 포함하고, 다중 섹터 스케줄러는 노드들, 디스크들, 및 섹터들의 조합으로 구현되는 클라우드 스토리지 시스템에 대해, 노드들, 디스크들, 및 섹터들의 조합을 기반으로, 0이 아닌 비트들이 대각으로 배열되는 회귀 코딩 행렬을 사용하여, 패리티 계산을 수행하도록 구성된다.

대표도 - 도8



(52) CPC특허분류

G06F 11/1048 (2013.01)

H03M 13/09 (2019.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711136767
과제번호	2021R1F1A1050750
부처명	과학기술정보통신부
과제관리(전문)기관명	한국연구재단
연구사업명	기본연구
연구과제명	[Ezbaro] IoT 데이터를 위한 연합 심층학습을 통한 블록체인지향 분산 엣지 컴퓨팅
솔루션에 대한 연구	
기 여 율	1/2
과제수행기관명	인하대학교산학협력단
연구기간	2021.06.01 ~ 2022.02.28

이 발명을 지원한 국가연구개발사업

과제고유번호	1345332336
과제번호	2018R1D1A1B07042602
부처명	교육부
과제관리(전문)기관명	한국연구재단
연구사업명	창의도전연구기반지원
연구과제명	[Ezbaro] 고가용성, 고신뢰성 클라우드 환경을위해 딥러닝과 이레이저 코딩 기술을
적용한 Semantic 기반의 스토리지 시스템	
기 여 율	1/2
과제수행기관명	인하대학교산학협력단
연구기간	2021.03.01 ~ 2022.02.28

명세서

청구범위

청구항 1

컴퓨터 장치에 있어서,

메모리; 및

상기 메모리와 연결되고, 상기 메모리에 저장된 적어도 하나의 명령을 실행하도록 구성되는 다중 섹터 (multisector; MS) 스케줄러

를 포함하고,

상기 다중 섹터 스케줄러는,

노드들, 디스크들, 및 섹터들의 조합으로 구현되는 클라우드 스토리지 시스템에 대해, 상기 노드들, 상기 디스크들, 및 상기 섹터들의 조합을 기반으로 패리티 계산을 수행하도록 구성되는,

컴퓨터 장치.

청구항 2

제 1 항에 있어서,

상기 다중 섹터 스케줄러는,

0이 아닌 비트들이 대각으로 배열되는 회귀 코딩 행렬을 사용하도록 구현되는,

컴퓨터 장치.

청구항 3

제 2 항에 있어서,

상기 다중 섹터 스케줄러가 MS 인코딩 스케줄러로 동작하는 경우, 상기 다중 섹터 스케줄러는,

상기 노드들, 상기 디스크들, 및 상기 섹터들의 조합을 기반으로, 상기 회귀 코딩 행렬을 사용하여 패리티 청크들을 생성하도록 구성되는,

컴퓨터 장치.

청구항 4

제 3 항에 있어서,

상기 다중 섹터 스케줄러는,

입력 데이터를 복수의 데이터 청크들에 할당하고,

상기 데이터 청크들과 상기 패리티 청크들을 패리티 노드들과 패리티 디스크들에 병렬로 기록하도록 구성되는,

컴퓨터 장치.

청구항 5

제 4 항에 있어서,
상기 다중 섹터 스케줄러는,
상기 패리티 디스크들을 사용하여, 상기 클라우드 스토리지 시스템에서 상기 디스크들 중 적어도 하나 또는 상기 섹터들 중 적어도 하나의 장애를 복구하도록 구성되는,
컴퓨터 장치.

청구항 6

제 4 항에 있어서,
상기 다중 섹터 스케줄러는,
상기 패리티 청크들을 생성하기 위해, 상기 데이터 청크들과 상기 회귀 코딩 행렬에 액세스하도록 구성되는,
컴퓨터 장치.

청구항 7

제 4 항에 있어서,
상기 다중 섹터 스케줄러가 MS 디코딩 스케줄러로 동작하는 경우, MS 디코더는,
상기 데이터 청크들을 읽으면서 장애 데이터와 생존 데이터를 검출하고,
상기 장애 데이터에 해당하는 열들을 상기 노드 계층 코딩 행렬로부터 삭제하여 회귀 역코딩 행렬을 도출하고,
상기 회귀 역코딩 행렬, 상기 생존 데이터, 및 상기 패리티 청크들을 사용하여, 상기 장애 데이터를 복구하는,
컴퓨터 장치.

청구항 8

제 7 항에 있어서,
상기 복구된 데이터와 상기 생존 데이터는 병합되어 상기 호스트로 전송되는,
컴퓨터 장치.

청구항 9

제 1 항에 있어서,
각 노드가 복수의 디스크들로 구성되고, 각 디스크가 복수의 섹터들로 구성되는,
컴퓨터 장치.

청구항 10

제 1 항 내지 제 9 항 중 어느 한 항에 기재된 컴퓨터 장치에 의해 수행되는 방법.

발명의 설명

기술 분야

[0001] 본 개시는 클라우드 스토리지 시스템을 위한 고성능 회귀 AVX(advanced vector extensions)-2 패리티 인코딩 디코딩의 컴퓨터 장치, 및 그의 방법에 관한 것이다.

배경 기술

[0002] 기존의 대규모 스토리지는 복제 기술(replication technique)을 사용하여 스토리지 시스템을 구축할 수 있는 가능성을 보여주었다. 복제 기술은 기존의 이레이저 코딩 방식(eraser coding schemes)에 비해 저장 효율성이 낮다. 데이터 디스크의 두 섹터와 중복 디스크의 다른 섹터가 모두 동시에 실패하는 경우, 섹터 수준 및 디스크 수준 장애들을 모두 처리하는, 부분 최대 거리 분리 가능(partial maximum distance separable; PMDS) 코드, STAIR 코드, 및 섹터-디스크(sector-disk; SD) 코드와 같은 기존 이레이저 코드를 사용하여 복구할 수 있다. 또한 다차원 스토리지에서 내부 패리티 스케줄러(parity scheduler; PS)는, 데이터가 섹터 및 노드 수준에 대해 메모리에 할당되는 방식에 대한 인식이 부족하기 때문에, 패리티 노드에서 패리티 디스크를 생성한다.

발명의 내용

해결하려는 과제

[0003] 본 개시는 클라우드 스토리지 시스템을 위한 고성능 회귀 AVX-2 패리티 인코딩 디코딩의 컴퓨터 장치, 및 그의 방법을 제공한다.

과제의 해결 수단

[0004] 본 개시에 따른 컴퓨터 장치는, 메모리, 및 상기 메모리와 연결되고, 상기 메모리에 저장된 적어도 하나의 명령을 실행하도록 구성되는 다중 섹터(multisector; MS) 스케줄러를 포함하고, 상기 다중 섹터 스케줄러는, 노드들, 디스크들, 및 섹터들의 조합으로 구현되는 클라우드 스토리지 시스템에 대해, 상기 노드들, 상기 디스크들, 및 상기 섹터들의 조합을 기반으로 패리티 계산을 수행하도록 구성된다. 다양한 실시예들에 따르면, 상기 다중 섹터 스케줄러는, 0이 아닌 비트들이 대각으로 배열되는 회귀 코딩 행렬을 사용하도록 구현된다.

발명의 효과

[0005] 본 개시에 따르면, SD 및 STAIR 스케줄러와 비교하여, MS 스케줄러는 평균 인코딩 및 디코딩 성능을 향상시킨다. MS 스케줄러는 고급 벡터 확장(advanced vector extensions; AVX)-2, OpenMP 스레드 병렬 처리 및 회귀 패리티 계산 소프트웨어 구현을 사용하여 다수의 패리티 디스크들 및 섹터들에 대해 패리티 계산을 수행할 수 있다. 결과적으로, MS 스케줄러의 대각 구조(diagonal structure) 때문에 패리티 청크들을 계산하기 위해 수행되는 XOR 연산이 더 적다. MS 스케줄러의 회귀 AVX-2/-512 대각 이레이저 코딩 구현은, MS 스케줄러가 많은 XOR 및 곱셈 연산을 건너뛰어 다중 레벨 장애로부터의 복구와 호환될 수 있다.

도면의 간단한 설명

[0006] 도 1은 기존의 클라우드 스토리지 시스템의 실질적인 스토리지 구조를 도시하는 도면이다.
 도 2는 도 1의 인코딩 및 디코딩 프로세스를 도시하는 도면이다.
 도 3은 도 1의 패리티 스케줄러의 알고리즘을 나타내는 도면이다.
 도 4, 도 5, 및 도 6은 도 1에 있는 인코딩 스케줄러들(CES1, CES2, CES3)을 각각 이용하는 마스터 노드의 알고리즘을 나타내는 도면들이다.
 도 7은 도 1과 관련하여, 스마트 벡터 밀집 패리티 스케줄러의 알고리즘을 나타내는 도면이다.
 도 8은 본 개시에 따른 컴퓨터 장치를 도시하는 도면이다.
 도 9는 도 8의 MS 스케줄러를 도시하는 도면이다.
 도 10은 도 9의 MS 인코딩 스케줄러의 알고리즘을 나타내는 도면이다.
 도 11은 도 9의 MS 디코딩 스케줄러의 알고리즘을 나타내는 도면이다.

도 12는 도 9의 MS 회귀 패리티 스케줄러에서의 MS 코딩 행렬 G의 회귀 행렬 개요를 나타내는 도면이다.

도 13 및 도 14는 도 9의 MS 회귀 패리티 스케줄러의 당면 과제에 대한 솔루션과 관련된 알고리즘들을 나타내는 도면들이다.

발명을 실시하기 위한 구체적인 내용

[0007] 최근 이레이저 코드에는 PMDS 코드, STAIR 코드, 및 SD 코드가 포함되며, 최대 3개의 레벨 장애들을 처리한다. 본 개시는, 전체 노드 또는 디스크를 내결함성 전용으로 지정하지 않고, 패리티 노드들, 디스크들 및 섹터들의 조합이 중복 전용으로 사용되는 새로운 인코딩 스케줄러로서, 다중 섹터(MS) 스케줄러를 제안한다. MS 스케줄러는 노드들, 디스크들 및 섹터들의 수가 증가하더라도 계산 효율성이 높다. 이하에서, 새로운 코드 구성을 사용하여 MS 스케줄러가 SD 코드 및 STAIR 코드의 기존 2차원 및 3차원 이레이저 코딩 기술보다 더 나은 계산 효율성을 제공함을 입증한다.

[0008] MS 스케줄러는 다중 레벨 장애 복구로 확장 가능하다. MS 스케줄러는 각 노드의 디스크들 및 섹터들의 수와 같은 각 노드의 내부 구조를 알고 있다. SD 및 STAIR 스케줄러와 비교하여, MS 스케줄러는 평균 인코딩 및 디코딩 성능을 향상시킨다. MS 스케줄러는 고급 벡터 확장(AVX)-2, OpenMP 스레드 병렬 처리 및 회귀 패리티 계산 소프트웨어 구현을 사용하여 다수의 패리티 디스크들 및 섹터들에 대해 패리티 계산을 수행할 수 있다. 결과적으로, MS 스케줄러의 대각 구조 때문에 패리티 청크들을 계산하기 위해 수행되는 XOR 연산이 더 적다. MS 스케줄러의 회귀 AVX-2/-512 대각 이레이저 코딩 구현은, MS 스케줄러가 많은 XOR 및 곱셈 연산을 건너뛰어 다중 레벨 장애로부터의 복구와 호환될 수 있다.

[0009] 하기 [표 1]에는 본 문서에서 사용된 매개변수와 그 기호, 정의 및 범위가 나와 있다. 이제, 인코딩 및 디코딩 프로세스, 실제 클라우드 스토리지 구조 및 이레이저 코드와 같은 클라우드 스토리지 시스템의 패리티 스케줄링과 관련된 몇 가지 주요 개념이 설명된다.

표 1

기호	정의	범위
K	데이터 섹터들의 총 수	$(1 \leq K \leq K)$
I	스토리지 계층들의 수	$i \in [1, \text{Max}]$
$k_i, k(i)$	주어진 계층 내 데이터 유닛들의 총 수	$(1 \leq i \leq 3)$
M	패리티 섹터들의 총 수	$(1 \leq K \leq M)$
$m_i, m(i)$	패리티 노드들의 총 수	$(1 \leq i \leq 3)$
G_i	주어진 계층 내 패리티 유닛들을 계산하기 위한 코딩 행렬	$(1 \leq i \leq 3)$
G_{23}	디스크 및 섹터 패리티를 계산하기 위한 코딩 행렬	-
$d[K \omega]$	데이터 청크	-
$p[M \omega]$	M 패리티 섹터들	-
LW	논리적 기록 작업 지연	-
CH_i	주어진 계층의 메인 메모리에 청크를 할당하는 데 따른 지연	$(1 \leq i \leq 3)$
XOR_i	주어진 계층의 패리티 스케줄링 수행으로 인한 지연	$(1 \leq i \leq 3)$
RW_i	주어진 계층의 디스크에 대한 물리적 파이프라인으로 인한 지연	$(1 \leq i \leq 3)$

[0011] 하기 [표 2]와 같이, 제안된 클라우드 스토리지 애플리케이션(cloud storage application; CSA)은 용도(usage), 패리티 스케줄링 기법(scheduling techniques), 내결함성(fault tolerance; FT) 특징, 패리티 오버헤드 패리티(parity of parity overhead; PoPO), CES 유형, 및 내부 청킹 오버헤드(internal chunking overhead; ICO) 측면에서 기존 CSA와 비교된다.

표 2

[0012]

	용도	스케줄링 기법	FT 특징	PoPO	CES 유형	ICO
제 1 기존 CSA (Lue et al., Plank et al. 및 Günther et al.)	인코딩, 디코딩	스마트 벡터	3 레벨	높음	CES1/2/3	있음
제 2 기존 CSA (Chen and Xu)	디코딩	벡터	디스크 레벨	중간	CES1/2/3	있음
제 3 기존 CSA (Zhou and Tian)	인코딩, 디코딩	벡터	디스크 레벨	높음	CES1	있음
제 4 기존 CSA (Gribaudo et al.)	인코딩	정규화	노드 레벨	중간	CES1	있음
제 5 기존 CSA (Pei et al.)	디코딩	정규화	디스크 레벨	높은	CES1	있음
제안된 CSA	인코딩, 디코딩	희귀 대각 벡터	다중 레벨	없음	PCES	없음

[0013]

CSA 용도는 인코딩과 디코딩 둘 다로 구성될 수 있지만, 이에 제한되지 않는다. 제 1 기존 CSA의 인코딩 및 디코딩 스케줄러는 SMID SSE 스마트 벡터 패리티 스케줄링 기술을 사용하며, SMID는 3 레벨 노드, 디스크 및 섹터 스토리지 계층들에서 구현될 수 있다. 또한, 이 CSA는 높은 PoPO 및 ICO를 갖는다. STAIR 코드 및 SD 코드는 클라우드 스토리지 환경에서 패리티 디스크 및 섹터를 계산하기 위해 SMID 갈르와(Galois) 필드 계산 코드를 사용한다. 제 2 기존 CSA의 디코딩 스케줄러는 스토리지 계층의 디스크 레벨에서 구현할 수 있는 간단한 행렬-벡터 패리티 스케줄링 기술을 사용한다. 결과적으로, 이 CSA는 중간 규모의 PoPO와 ICO를 갖는다. 제 3 기존 CSA의 인코딩 및 디코딩 스케줄러는 높은 PoPO와 ICO를 갖는 간단한 행렬-벡터 패리티 스케줄링 기법을 사용한다. 반면에, 제 4 CSA의 인코딩 스케줄러는 정규화된 패리티 스케줄링 기법을 적용하는데, 이는 중간 PoPO와 ICO로 디스크 레벨에서 구현될 수 있다. 제 5 CSA의 디코딩 스케줄러는 정규화된 패리티 스케줄링 기법을 사용하며, 이것은 높은 PoPO와 ICO로 디스크 레벨에서 구현될 수 있다. 마지막으로, 제안된 CSA의 인코딩 및 디코딩 스케줄러도 희귀 패리티 스케줄링 기법을 사용하지만, 새로 제안된 클라우드 인코딩 스케줄러(proposed cloud encoding scheduler; PCES)를 사용하기 때문에 PoPO나 ICO가 없다.공정한 비교를 위해, 제안된 CSA를 SD 코드 및 STEAR 코드를 사용하는 제 1 기존 CSA의 인코딩 및 디코딩 스케줄러와 비교한다. 제 1 기존 CSA는 제안된 CSA와 가장 유사하다. LRC 코드는 확장 가능한 코딩 스키마가 아니며 섹터 장애를 자연스럽게 제공하지 않기 때문에 제외했다. CSA들에 대한 추가 조사를 수행한 결과 기존 클라우드 스토리지 시스템에 세 가지 패턴들의 인코딩 스케줄러들(CES1, CES2, CES3)이 있는 것으로 나타났다.

[0015]

이하, 기존의 클라우드 스토리지 시스템의 패리티 스케줄러가 첨부된 도면을 참조하여 설명된다.

[0016]

도 1은 기존의 클라우드 스토리지 시스템의 실질적인 스토리지 구조를 도시하는 도면이다. 도 2는 도 1의 인코딩 및 디코딩 프로세스를 도시하는 도면이다. 도 3은 도 1의 패리티 스케줄러의 알고리즘을 나타내는 도면이다. 도 4, 도 5, 및 도 6은 도 1에 있는 인코딩 스케줄러들(CES1, CES2, CES3)을 각각 이용하는 마스터 노드의 알고리즘을 나타내는 도면들이다. 도 7은 도 1과 관련하여, 스마트 벡터 밀집 패리티 스케줄러의 알고리즘을 나타내는 도면이다.

[0017]

도 1을 참조하면, 기존의 클라우드 스토리지 시스템은 노드, 디스크, 섹터 조합으로 구현되며, 패리티 스케줄러를 사용한다. 여기서, 스트립(strip)이 코딩 이론에 기초한 섹터의 $r \times K$ 어레이라고 가정하며 각 섹터를 기호로 나타낸다. 특히, 풀(pool)이 $K(1)$ 개의 데이터 노드들로 구성되고, 각 노드가 $K(2)$ 개의 데이터 디스크들로 구성되고, 각 디스크가 $K(1)$ 개의 데이터 섹터들로 구성된다고 가정한다. 또한 $m(3)/m(2)/m(1)$ 기호로 표시된 (노드/디스크/섹터) 풀의 결합된 최대 다중 레벨 장애 복구(multilevel failure recovery; MLR)의 새로운 속성을 정의한다. 즉, 풀당 최대 $m(3)$ 개의 노드들, 노드당 $m(2)$ 개의 디스크들, 디스크당 $m(1)$ 개의 섹터들을 복구할 수 있으며, 나머지 섹터들은 데이터 전용이다. 이러한 스토리지 구조를 기반으로 클라우드 환경에서, 인코딩 및 디코딩 성능을 향상시키는 두 가지 단계가 있다. 먼저, 인코딩하는 동안 사용자는 로드 밸런서 노드를 통해 데이터 청크 D를 데이터 노드에 쓰도록 요청한다. 데이터 청크 D는 패리티 생성을 위해 CPU 메모리에 할당함으로써 데이터 노드로 전송된다. 다음으로, CPU 커널은 여러 CPU 스레드를 호출하여 패리티 청크 P를 생성한다.

따라서, P는 CPU 메모리에서 전송되어 패리티 및 데이터 노드에 데이터와 패리티 청크를 저장한다.

[0018] 이전 연구들에 따르면, 인코딩 프로세스는 도 2의 (a)에 도시된 바와 같이 세 단계로 구성된다. 첫 번째 단계에서, 원본 데이터 청크(D)는 다음 시퀀스 $\{D_1, D_2, \dots, D_K\}$ 로 메모리에 할당될 것이다. 다음으로, 두 번째 단계에서, 패리티 스케줄러는 여러 CPU 스레드를 호출하여 코딩 행렬(G)과 할당된 데이터 청크를 사용하여 패리티 청크(P)를 생성한다. 마지막으로, 세 번째 단계에서, 인코더 파이프라인들은 데이터와 패리티를 데이터 및 패리티 노드들에 할당한다. 한편, 디코딩 프로세스는 도 2의 (b)에 도시된 바와 같이 세 단계로 구성된다. 첫 번째 단계에서, 디코더는 생존 데이터와 패리티를 읽고 메모리에 할당한다. 두 번째 단계에서, 패리티 스케줄러는 여러 CPU 스레드를 호출하여, 재생된 코딩 행렬(G')을 사용하여 장애 데이터를 복구하고, 데이터와 패리티 청크를 할당한다. 세 번째 단계에서, 디코더는 복구된 데이터를 예비(spare) 노드에 기록한다.

[0019] 클라우드 스토리지 애플리케이션은 안정성을 보장하고 노드 내결함성을 제공하기 위해 다양한 이레이저 코딩 기법을 사용하여 패리티 노드라고 하는 코딩 데이터 노드를 생성하도록 설계되었다. 리드-솔로몬(Reed-Solomon; RS) 코드와 단일 패리티 검사(single parity check; SPC) 코드 등 다양한 유형의 이레이저 코드가 있으며, 코딩 기법에 의해 하기 [수학식 1]과 같은 정의가 있다.

수학식 1

$$P_i = \sum_{j=0}^{K-1} P_i \oplus (d_j \times g_{i,j}) = (d_0 \times g_{i,0}) \oplus, \dots, \oplus (d_{K-1} \times g_{i,K-1}).$$

[0020]

[0021] 위크로드 D는 클라우드 스토리지 애플리케이션에서 z 개의 데이터 청크들 $D = \{D_0, \dots, D_{z-1}\}$ 로 구성된다 고 가정한다. i번째 데이터 청크는 K 개의 데이터 블록들 $\bar{D}_i = \{d_0, \dots, d_{K-1}\}$ 로 구성되며, i번째 패리티 청크는 패리티 블록 $P_i = \{p_0, \dots, p_{M-1}\}$ 로 구성된다. 여기서, M은 스트라이프들의 수를 나타낸다. 이레이저 코드에서, $(Data|Coding) = (D|P) = D \times G = D \times (I|G)$ 라 가정하며, 여기서, 제너레이터 행렬(generator matrix) $G = (I|G)$ 는 $K \times K$ 코드 단어가 사용된 항등 행렬(identity matrix) I로 구성되며, 코딩 행렬 $G = \{g_0, \dots, g_{K-1}, \dots, g_{M-1,K-1}\}$ 은 $M \times K$ 코딩 블록들을 갖는다. 패리티 스케줄러는 스트라이프의 데이터 코드 워드들 간에 XOR 연산을 실행한다. 또한, i번째에서 패리티 코드 워드 P_i를 생성하기 위한 XOR 연산 시퀀스는 상기 [수학식 1]과 같이 스트라이프 처리되며, 도 3의 알고리즘 1과 같이 코딩 행렬(G)을 사용하는 패리티 스케줄러의 의사코드가 제공된다.

[0022] 도 4의 알고리즘 2는 세 가지 개별 수준들의 패리티 스케줄러들을 사용하는 클라우드 인코딩 스케줄러(CES1)를 이용하는 마스터 노드의 의사코드를 보여준다. 2행에서, 입력 데이터는 마스터 노드에 의해 K 개의 데이터 청크들 $(d_1, \dots, d_K)$ 에 할당된다. 3행에서, 외부 마스터 노드 인코더는 패리티 노드 청크들 $(p_1, \dots, p_{m_1})$ 을 생성하기 위해 패리티 스케줄러를 호출한다. 패리티 스케줄러가 노드 기반 패리티 청크들을 생성하려면 모든 데이터 청크들과 노드 계층 코딩 행렬 G₁에 액세스해야 한다. 4행에서, 마스터 노드는 데이터 및 패리티 청크들을 패리티 노드들에 병렬로 기록한다. 첫 번째 루프는 내부 디스크 계층 전용으로 6행 내지 16행에 위치한다. 8행에서, 노드 계층에서 전송된 데이터는 각 노드의 스토리지 디스크 컨트롤러에 의해 k₂ 개의 데이터 청크들 $(d_1, \dots, d_{k_2})$ 에 할당된다. 9행에서, 디스크 컨트롤러는 패리티 스케줄러를 호출하여 패리티 디스크 청크들 $(p_1, \dots, p_{m_2})$ 을 생성한다. 패리티 스케줄러는 디스크 기반 패리티 청크들을 생성하려면 디스크 계층 코딩 행렬 G₂에 액세스해야 한다. 10행에서, 마스터 노드는 패리티 디스크 청크들을 패리티 디스크들에 병렬로 기록한다.

[0023] 결과적으로, 두 번째 루프는 섹터 계층 전용이며 11행 내지 15행에 위치한다. 12행에서, 노드 계층에서 전송된 데이터는 각 노드의 스토리지 디스크 컨트롤러에 의해 k_3 개의 데이터 청크들 $(d_1, \dots, d_{k_3})$ 에 할당된다. 13행에서, 디스크 컨트롤러는 패리티 스케줄러를 호출하여 패리티 섹터 청크들 $(p_1, \dots, p_{m_3})$ 을 생성한다. 패리티 스케줄러는 섹터 기반 패리티 청크들을 생성하려면 디스크 계층 코딩 행렬 G_3 에 액세스해야 한다. 마지막으로, 14행에서, 내부 노드 컨트롤러는 패리티 섹터 청크들을 패리티 디스크들에 기록한다. CES1의 논리적 기록 작업 지연 LW 은 하기 [수학식 2]와 같다.

수학식 2

$$LW = (k_1 + m_1)CH_1 + (k_2 + m_2)CH_2 + (k_3 + m_3)CH_3 + \\ (k_1)XOR_1 + k_2(k_1 + m_1)XOR_2 + (k_1 + m_1)(k_2 + m_2)(k_3)XOR_3 + \\ (k_1 + m_1)RW_1 + (k_1 + m_1)(m_2)RW_2 \\ + (k_1 + m_1)(k_2 + m_2)(m_3)RW_3$$

여기서,

$$CH_1 > CH_2 > CH_3 \\ XOR_1 > XOR_2 > XOR_3 \\ RW_1 > RW_2 > RW_3$$

[0024]

[0025] 6행 내지 16행의 내부 노드 루프와 11행 내지 15행에 디스크 루프 내에, 패리티 계산을 위한 패리티 디스크와 노드를 추가했다. 특히, 클라우드 레벨 스토리지 시스템은 일반적으로 내부 노드 어레이 및 디스크 어레이 사양을 인식하지 못한다. 결과적으로, CES1 유형의 스토리지 시스템에서 상당한 양의 낭비 패리티가 생성된다.

[0026] 도 5의 알고리즘 3은 CES2를 이용하는 마스터 노드의 유사 코드를 보여준다. 2행에서, 입력 데이터는 마스터 노드에 의해 K 개의 데이터 청크들 $(d_1, \dots, d_K)$ 에 할당된다. 3행에서, 외부 마스터 노드 인코더는 패리티 노드 청크들 $(p_1, \dots, p_{m_1})$ 을 생성하기 위해 패리티 스케줄러를 호출한다. 패리티 스케줄러는 노드 기반 패리티 청크들을 생성하기 위해 데이터 청크 및 노드 계층 코딩 행렬 G_1 에 액세스해야 한다. 4행에서, 마스터 노드는 데이터 및 패리티 청크들을 패리티 노드들에 병렬로 기록한다.

[0027] 첫 번째 루프는 내부 디스크 및 섹터 계층 전용이며, 6행 내지 10행에 설명되어 있다. 7행에서, 노드 계층으로부터 전송된 데이터는 각 노드의 스토리지 디스크 컨트롤러에 의해 k_2 개의 데이터 청크들 $(d_1, \dots, d_{k_2})$ 에 할당된다. 8행에서, 디스크 및 섹터 기반 패리티 스케줄러는 $p_1, \dots, p_{m_2+m_3}$ 을 생성한다. 패리티 스케줄러는 SD 코드 또는 STAIR 코드를 사용하여 해당 노드의 패리티 디스크 및 섹터를 생성하기 위해 디스크 계층 코딩 행렬 G_{23} 에 액세스해야 한다. 9행에서, 마스터 노드는 패리티 디스크와 섹터를 디스크들에 병렬로 기록한다. CES2의 논리적 기록 작업 지연은 LW 는 하기 [수학식 3]과 같다.

수학식 3

$$\begin{aligned}
LW = & (k_1 + m_1)CH_1 + (k_2 + m_2)CH_2 + \\
& (k_1)XOR_1 + k_2(k_1 + m_1)XOR_2 + (k_1 + m_1)(k_3)XOR_3 + \\
& (k_1 + m_1)RW_1 + (k_1 + m_1)(m_2)RW_2 \\
& + (k_1 + m_1)(k_2 + m_2)(m_3)RW_3
\end{aligned}$$

여기서,

$$\begin{aligned}
CH_1 & > CH_2 > CH_3 \\
XOR_1 & > XOR_2 > XOR_3 \\
RW_1 & > RW_2 > RW_3
\end{aligned}$$

[0028]

[0029]

도 6의 알고리즘 4는 CES3를 이용하는 마스터 노드의 유사 코드를 보여준다. 2행에서, 입력 데이터는 마스터 노드에 의해 K 개의 데이터 청크들 $(d_1, \dots, d_K)$ 에 할당된다. 3행에서, 외부 마스터 노드 인코더는 패리티 노드 청크들 $(p_1, \dots, p_M)$ 을 생성하기 위해 패리티 스케줄러를 호출한다. 패리티 스케줄러는 노드, 디스크 및 섹터 기반 패리티 청크들을 생성하기 위해 데이터 청크들과 노드 계층 코딩 행렬 G에 액세스해야 한다. 4행에서, 마스터 노드는 데이터와 패리티 청크들을 패리티 노드들과 디스크들에 병렬로 기록한다. CES3의 논리적 쓰기 작업 지연 LW는 하기 [수학식 4]와 같다.

수학식 4

$$\begin{aligned}
LW = & ((m_1 k_2) + m_2)CH_2 + ((m_1 k_2) + m_2)XOR_2 + m_3 XOR_3 + \\
& ((m_1 k_2) + m_2)RW_2 + (m_3)RW_3
\end{aligned}$$

여기서,

$$CH_2 > CH_3, XOR_2 > XOR_3, RW_2 > RW_3$$

[0030]

[0031]

클라우드 스토리지 시스템에서, 코딩 스트라이프의 여러 기호들이 있는 각 청크는 서로 다른 노드 및 디스크에 상주한다. m_1 개의 노드 장애들, m_2 개의 디스크 장애들, m_3 개의 섹터 장애들이 복합적으로 발생하는 경우 $m_1 +$

m_2 개의 전체 청크들과 추가적인 m_3 개의 섹터들에 영향을 미친다. 따라서,

$$m' = m_1 \times k_2 + m_2$$

개의 디스크 장애들과 추가적인 $s = m_3$ 개의 섹터 장애들을 허용하는 STAIR 코드 및 SD 코드는 이 장애 사례를 처리할 수 있다. 그러므로, 각 노드의 패리티 노드들 및 데이터 디스크들의 수가 증가함에 따라 패리티 디스크들의 수 m' 이 증가한다. 특히, 데이터 디스크들의 수가 증가함에 따라 스트립 크기가 커지기 때문에, 패리티 디스크들의 수가 증가함에 따라 STAIR 코드 및 SD 코드 성능이 급격히 저하된다.

[0032]

한편, 스마트 벡터 밀집 패리티 스케줄러에 있어서, 인텔(Intel) 명령어 집합 확장은 과학 시뮬레이션, 재무 분석, 3D 모델링 및 분석과 같은 애플리케이션의 성능을 향상시키도록 설계되었다. 계산 스케줄의 각 단계는 할당 또는 XOR 연산이다. 실제로는 비트 단위로 수행하는 대신 SIMD 명령어 집합이 있는 CPU에 대해 한 번에 비트 집

합으로 처리된다. 확장된 위드는 128비트 또는 256비트일 수 있으며, 여기서 각 명령어는 8, 64, 128, 또는 256 비트에서 작업을 실행한다. 예를 들어, 완전한 갈루와(Galois) 필드 산술 GF(4) XOR 곱하기 C 코드는 "PS512"로 명명된다. SMID AVX-2 기술을 사용한 XOR 연산은 도 7의 알고리즘 5과 같다.

[0034] 본 개시는 제안된 기술을 SD 코드 및 STAIR 코드와 같은 최첨단 다중 레벨 인코딩의 기존의 기술과 비교한다. 기존 STAIR 및 SD 스케줄러는 고밀도 행렬 기반 패리티 계산 소프트웨어에서 SSE 하드웨어 확장을 사용하기 때문에 특정 수의 패리티 디스크들 및 섹터들에 바인딩되어 있다. 반대로 제안된 MS 스케줄러는 회귀 패리티 계산 소프트웨어를 사용하여 AVX 및 OpenMP를 사용하여 많은 패리티 크기들에 대해 패리티 계산을 수행할 수 있다. 따라서 MS 스케줄러의 대각 구조 때문에 패리티 청크들을 계산하는 반복 횟수가 줄어든다. 제안된 회귀 AVX 대각 이레이저 코딩은 다중 레벨 장애들로부터의 복구와 호환되므로 MS 스케줄러는 클라우드 스토리지의 확장 가능한 환경에 적합하다.

[0035] 이하, 본 개시의 다중 섹터(MS) 스케줄러가 첨부된 도면을 참조하여 설명된다.

[0036] 도 8은 본 개시에 따른 컴퓨터 장치(100)를 도시하는 도면이다. 도 9는 도 8의 MS 스케줄러(140)를 도시하는 도면이다. 도 10은 도 9의 MS 인코딩 스케줄러(210)의 알고리즘을 나타내는 도면이다. 도 11은 도 9의 MS 디코딩 스케줄러(220)의 알고리즘을 나타내는 도면이다. 도 12는 도 9의 MS 회귀 패리티 스케줄러(230)에서의 MS 코딩 행렬 G의 회귀 행렬 개요를 나타내는 도면이다. 도 13 및 도 14는 도 9의 MS 회귀 패리티 스케줄러(230)의 당면 과제에 대한 솔루션과 관련된 알고리즘들을 나타내는 도면들이다.

[0037] 도 8을 참조하면, 컴퓨터 장치(100)는 입력 모듈(110), 출력 모듈(120), 메모리(130), 또는 MS 스케줄러(140) 중 적어도 하나를 포함할 수 있다. 어떤 실시예에서, 컴퓨터 장치(100)의 구성 요소들 중 적어도 하나가 생략될 수 있으며, 적어도 하나의 다른 구성 요소가 추가될 수 있다. 어떤 실시예에서, 컴퓨터 장치(100)의 구성 요소들 중 적어도 두 개가 하나의 통합된 회로로 구현될 수 있다.

[0038] 입력 모듈(110)은 컴퓨터 장치(100)의 적어도 하나의 구성 요소에 사용될 신호를 입력할 수 있다. 입력 모듈(110)은, 사용자가 컴퓨터 장치(100)에 직접적으로 신호를 입력하도록 구성되는 입력 장치, 주변의 변화를 감지하여 신호를 발생하도록 구성되는 센서 장치, 또는 외부 기기로부터 신호를 수신하도록 구성되는 수신 장치 중 적어도 하나를 포함할 수 있다. 예를 들면, 입력 장치는 마이크로폰(microphone), 마우스(mouse) 또는 키보드(keyboard) 중 적어도 하나를 포함할 수 있다. 어떤 실시예에서, 입력 장치는 터치를 감지하도록 설정된 터치 회로(touch circuitry) 또는 터치에 의해 발생하는 힘의 세기를 측정하도록 설정된 센서 회로 중 적어도 하나를 포함할 수 있다. 이 때, 입력 모듈(110)은 PPG 센서(111)를 포함할 수 있다.

[0039] 출력 모듈(120)은 컴퓨터 장치(100)의 외부로 정보를 출력할 수 있다. 출력 모듈(120)은, 정보를 시각적으로 출력하도록 구성되는 표시 장치, 정보를 오디오 신호로 출력할 수 있는 오디오 출력 장치, 또는 정보를 무선으로 송신할 수 있는 송신 장치 중 적어도 하나를 포함할 수 있다. 예를 들면, 표시 장치는 디스플레이, 홀로그램 장치 또는 프로젝터 중 적어도 하나를 포함할 수 있다. 일 예로, 표시 장치는 입력 모듈(110)의 터치 회로 또는 센서 회로 중 적어도 하나와 조립되어, 터치 스크린으로 구현될 수 있다. 예를 들면, 오디오 출력 장치는 스피커 또는 리시버 중 적어도 하나를 포함할 수 있다.

[0040] 일부 실시예들에 따르면, 수신 장치와 송신 장치는 통신 모듈로 구현될 수 있다. 통신 모듈은 컴퓨터 장치(100)에서 외부 기기와 통신을 수행할 수 있다. 통신 모듈은 컴퓨터 장치(100)와 외부 기기 간 통신 채널을 수립하고, 통신 채널을 통해, 외부 기기와 통신을 수행할 수 있다. 여기서, 외부 기기는 차량, 위성, 기지국, 서버 또는 다른 컴퓨터 시스템 중 적어도 하나를 포함할 수 있다. 통신 모듈은 유선 통신 모듈 또는 무선 통신 모듈 중 적어도 하나를 포함할 수 있다. 유선 통신 모듈은 외부 기기와 유선으로 연결되어, 유선으로 통신할 수 있다. 무선 통신 모듈은 근거리 통신 모듈 또는 원거리 통신 모듈 중 적어도 하나를 포함할 수 있다. 근거리 통신 모듈은 외부 기기와 근거리 통신 방식으로 통신할 수 있다. 예를 들면, 근거리 통신 방식은, 블루투스(Bluetooth), 와이파이 다이렉트(WiFi direct), 또는 적외선 통신(IrDA; infrared data association) 중 적어도 하나를 포함할 수 있다. 원거리 통신 모듈은 외부 기기와 원거리 통신 방식으로 통신할 수 있다. 여기서, 원거리 통신 모듈은 네트워크를 통해 외부 기기와 통신할 수 있다. 예를 들면, 네트워크는 셀룰러 네트워크, 인터넷, 또는 LAN(local area network)이나 WAN(wide area network)과 같은 컴퓨터 네트워크 중 적어도 하나를 포함할 수 있다.

[0041] 메모리(130)는 컴퓨터 장치(100)의 적어도 하나의 구성 요소에 의해 사용되는 다양한 데이터를 저장할 수 있다.

예를 들면, 메모리(130)는 휘발성 메모리 또는 비휘발성 메모리 중 적어도 하나를 포함할 수 있다. 데이터는 적어도 하나의 프로그램 및 이와 관련된 입력 데이터 또는 출력 데이터를 포함할 수 있다. 프로그램은 메모리(130)에 적어도 하나의 명령을 포함하는 소프트웨어로서 저장될 수 있으며, 운영 체제, 미들 웨어 또는 어플리케이션 중 적어도 하나를 포함할 수 있다.

[0042] MS 스케줄러(140)는 메모리(130)의 프로그램을 실행하여, 컴퓨터 장치(100)의 적어도 하나의 구성 요소를 제어할 수 있다. 이를 통해, MS 스케줄러(140)는 데이터 처리 또는 연산을 수행할 수 있다. 이 때, MS 스케줄러(140)는 메모리(130)에 저장된 명령을 실행할 수 있다.

[0043] MS 스케줄러(140)는 각 노드의 디스크들 및 섹터들의 수와 같은 각 노드의 내부 구조를 알고 있다. SD 및 STAIR 스케줄러와 비교하여, MS 스케줄러(140)는 평균 인코딩 및 디코딩 성능을 향상시킨다. MS 스케줄러(140)는 AVX-2, OpenMP 스레드 병렬 처리 및 회귀 패리티 계산 소프트웨어 구현을 사용하여 다수의 패리티 디스크들 및 섹터들에 대해 패리티 계산을 수행할 수 있다. 결과적으로, MS 스케줄러(140)의 대각 구조 때문에 패리티 청크들을 계산하기 위해 수행되는 XOR 연산이 더 적다. MS 스케줄러(140)의 회귀 AVX-2/-512 대각 이레이저 코딩 구현은, MS 스케줄러(140)가 많은 XOR 및 곱셈 연산을 건너뛰어 다중 레벨 장애로부터의 복구와 호환될 수 있다.

[0044] 이러한 MS 스케줄러(140)는 도 9에 도시된 바와 같은 MS 인코딩 스케줄러(210), MS 디코딩 스케줄러(220), 또는 MS 회귀 패리티 스케줄러(230) 중 적어도 하나로 동작할 수 있다. 어떤 실시예에서, MS 스케줄러(140)의 구성 요소들 중 적어도 하나가 생략될 수 있으며, 적어도 하나의 다른 구성 요소가 추가될 수 있다. 어떤 실시예에서, MS 스케줄러(140)의 구성 요소들 중 적어도 두 개가 하나의 통합된 회로로 구현될 수 있다.

[0045] MS 인코딩 스케줄러(210)

[0046] 도 10의 알고리즘 6은 PCES를 사용하는 마스터 노드의 유사 코드를 보여준다. 2행에서, 입력 데이터는 마스터 노드에 의해 K 개의 데이터 청크들 $(d_1, \dots, d_K)$ 에 할당된다. 3행에서, 외부 마스터 노드 인코더는 패리티 노드 청크들 $(p_1, \dots, p_M)$ 을 생성하기 위해 패리티 스케줄러를 호출한다. 패리티 스케줄러가 노드, 디스크, 및 섹터 기반 패리티 청크들 생성하려면 모든 데이터 청크들과 노드 계층 코딩 행렬 G에 액세스해야 한다. 4행에서, 마스터 노드는 데이터와 패리티 청크들을 패리티 노드들과 디스크들에 병렬로 기록한다. PCES의 논리적 기록 연산 지연은 하기 [수학식 5]와 같다.

수학식 5

$$LW = (K+M)CH_3 + (M)XOR_3 + (K+M)RW_3$$

[0047]

[0048] 데이터 청크는 주어진 수의 노드들, 디스크들, 및 섹터들과 관련된 요소이기 때문에, MS 스케줄러(140)는 각 노드의 디스크들 및 섹터들의 수를 포함하여 각 노드의 내부 구조를 알고 있다. 이는 MS 스케줄러(140)에게 상당한 이점을 제공한다. MS 코드들에서는, 각 패리티 노드를 사용하여, 동일한 중복성과 거의 동일한 수의 XOR 연산을 통해 $K(2) \times K(1)$ 개의 섹터들, $K(2)$ 개의 디스크들, 또는 단일 노드 장애를 복구할 수 있다. 특히, 각 패리티 디스크를 사용하여, 동일한 중복성과 거의 동일한 수의 XOR 연산을 통해 $K(1)$ 개의 섹터들 또는 단일 디스크 장애를 복구할 수 있다. 또한 각 디스크에 대해 추가적인 로컬 섹터 패리티도 포함시킬 수 있다.

[0049] MS 디코딩 스케줄러(220)

[0050] 도 11의 알고리즘은 본 문서에서 설명되는 표기법에 기초하여 제안된 MS 패리티 디코딩 스케줄러(220)의 유사 코드를 보여준다. 즉, 도 11은 MS 디코더가 클라우드 스토리지 시스템에서 m_1 개의 노드 장애, m_2 개의 디스크 장애들, 및 m_3 개의 섹터 장애들을 복구할 수 있음을 보여준다. 1행에서, MS 디코더는 $d[K]$ 로 표시된, K 개의 섹터들을 갖는 각 데이터 청크를 읽는다. 2행에서, 데이터 청크들을 읽을 때 특정 레벨 장애들의 수가 M을 초과하면, 복구할 수 없는 디제스터(disaster) 장애이다. 3행 내지 9행에서, $K - \Phi(G')$ 개의 장애들이 주어진 레벨에서 발생할 때, 알고리즘은 장애 데이터까지 복구된다. 4행에서, MS 디코더는 해당 생존 데이터와 패리티를 ds 에 할당한다. 5행 내지 6행에서, MS 디코더는 장애 데이터에 해당하는 열들(columns)이 코딩 행렬 G로부터 삭제되는 회귀 역코딩 행렬 B^{-1} 을 불러온다. 7행에서, PCES 알고리즘을 사용하여 MS 디코더는 역코딩 행렬 B^{-1} 과

생존 데이터 및 패리티 ds를 사용하여 장애 데이터 d' 을 복구한다. 8행에서, 생존 데이터 ds와 복구된 데이터 d '은 복구된 데이터 d[K]에 병합된다. 마지막으로, 9행에서, 복구된 데이터 청크들 d[K]가 호스트로 전송된다.

[0051] MS 회귀 패리티 스케줄러(230)

[0052] 패리티 컴퓨팅에 대한 이전 연구는 Galois 필드에서 조밀한 행렬 곱을 사용하는 데만 초점을 맞췄다. 본 개시는 MS 회귀 행렬 커널이 $P \leftarrow G \times D$ 형태라고 가정한다. 여기서, G는 $K \times M$ 회귀 행렬이고, D 및 P는 $M \times 1$ 및 $K \times 1$ 열 벡터(SpMV)이다. MS 회귀 커널에서 각 0이 아닌 비트에 액세스하여 행렬(데이터) 항목에 곱하고 결과는 G-벡터(코딩 행렬) 항목에 XOR 연산을 한 번 적용한다. 즉, 0이 아닌 각 비트에 대해 2개의 읽기와 1개의 읽기/쓰기가 할당된다. 제안된 회귀 패리티 계산을 구현하기 위해 SpMV 및 SPMM 알고리즘의 이전 구현을 활용한다.

[0053] 회귀 행렬의 가장 많이 적용되는 할당 포맷은 CSR(compressed storage row)이다. 도 12는 MS 코딩 행렬 G의 회귀 행렬 개요를 보여준다. 도 12는 회귀 8×12 행렬에 대한 CSR 포맷의 예를 보여준다. 조밀한 $K \times M$ 코딩 행렬 G와 회귀 행렬에 대한 행렬-벡터 곱의 구현은 도 12와 같이 CSR 형식으로 저장된다. CSR 형식에 따라 회귀 행렬의 0이 아닌 비트는 메모리(130)에 주요 행 순서로 값으로 저장 및 병합된다. 열 인덱스 어레이인 scol은 원래 행렬에 각 비트의 열을 저장하고 s + 1 크기의 srow 어레이는 각 행의 시작을 저장한다. 또한 s는 G 코딩 행렬에서 0이 아닌 비트의 수를 나타낸다.

[0054] 회귀 MS-행렬 패리티 계산(SPC) 알고리즘 구현의 당면 과제는 다음과 같다. I 조밀한 MS 코딩 행렬 G($M \times K$)와 달리 SPC의 행렬 요소는 한 번만 사용되며 회귀 코딩 매트릭스 G에는 시간적 위치 지정이 없다. II 추가 할당 작업에는 캐시 간섭을 위한 네트워크 트래픽이 포함된다. III 회귀 행렬에서는 데이터 벡터 D에 대한 불규칙한 메모리 액세스가 있는 반면, 데이터 벡터 D에 대한 접근은 조밀한 행렬에 대해 순차적이다. IV 실제로 짧은 행 길이가 있어 내부 루프의 이동 횟수가 작을 때 외부 루프의 오버헤드가 상당량 발생한다. 이에 대한 솔루션이 다음과 같이 제안된다. 사전 설정된 데이터에 회귀 MS 코딩을 사전 할당함으로써 알고리즘은 SIMD를 사용하여 곱셈 프로세스를 위한 통합 액세스를 생성한다. XOR 감소를 위한 원자적 액세스는 SPC 컴퓨팅 중에 개방형 MP 원자 연산을 사용한다. 자세한 구현 프로세스는 도 13 및 도 14의 알고리즘 7 및 8에 설명되어 있다.

[0056] 다양한 실시예들은 클라우드 스토리지 시스템을 위한 고성능 회귀 AVX-2 패리티 인코딩 디코딩의 컴퓨터 장치(100), 및 그의 방법을 제공한다.

[0057] 다양한 실시예들에 따르면, 컴퓨터 장치(100)는 메모리(130), 및 메모리와 연결되고, 메모리에 저장된 적어도 하나의 명령을 실행하도록 구성되는 다중 섹터 스케줄러(140)를 포함한다.

[0058] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)는, 노드들, 디스크들, 및 섹터들의 조합으로 구현되는 클라우드 스토리지 시스템에 대해, 노드들, 디스크들, 및 섹터들의 조합을 기반으로 패리티 계산을 수행하도록 구성된다.

[0059] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)는, 0이 아닌 비트들이 대각으로 배열되는 회귀 코딩 행렬을 사용하도록 구현된다.

[0060] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)가 MS 인코딩 스케줄러(210)로 동작하는 경우, 다중 섹터 스케줄러(140)는 노드들, 디스크들, 및 섹터들의 조합을 기반으로, 회귀 코딩 행렬을 사용하여 패리티 청크들을 생성하도록 구성된다.

[0061] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)는 입력 데이터를 복수의 데이터 청크들에 할당하고, 데이터 청크들과 패리티 청크들을 패리티 노드들과 패리티 디스크들에 병렬로 기록하도록 구성된다.

[0062] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)는 패리티 디스크들을 사용하여, 클라우드 스토리지 시스템에서 디스크들 중 적어도 하나 또는 섹터들 중 적어도 하나의 장애를 복구하도록 구성된다.

[0063] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)는 패리티 청크들을 생성하기 위해, 데이터 청크들과 회귀 코딩 행렬에 액세스하도록 구성된다.

[0064] 다양한 실시예들에 따르면, 다중 섹터 스케줄러(140)가 MS 디코딩 스케줄러(220)로 동작하는 경우, MS 디코더는 데이터 청크들을 읽으면서 장애 데이터와 생존 데이터를 검출하고, 장애 데이터에 해당하는 열들을 노드 계층

코딩 행렬로부터 삭제하여 회귀 역코딩 행렬을 도출하고, 회귀 역코딩 행렬, 생존 데이터, 및 패리티 체크들을 사용하여, 장애 데이터를 복구한다.

[0065] 다양한 실시예들에 따르면, 복구된 데이터와 생존 데이터는 병합되어 호스트로 전송된다.

[0066] 다양한 실시예들에 따르면, 각 노드가 복수의 디스크들로 구성되고, 각 디스크가 복수의 섹터들로 구성된다.

[0068] 상술한 방법은 컴퓨터에서 실행하기 위해 컴퓨터 판독 가능한 기록 매체에 저장된 컴퓨터 프로그램으로 제공될 수 있다. 매체는 컴퓨터로 실행 가능 한 프로그램을 계속 저장하거나, 실행 또는 다운로드를 위해 임시 저장하는 것일 수도 있다. 또한, 매체는 단일 또는 수개 하드웨어가 결합된 형태의 다양한 기록 수단 또는 저장수단일 수 있는데, 어떤 컴퓨터 시스템에 직접 접속되는 매체에 한정되지 않고, 네트워크 상에 분산 존재하는 것일 수도 있다. 매체의 예시로는, 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체, CD-ROM 및 DVD 와 같은 광기록 매체, 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical medium), 및 ROM, RAM, 플래시 메모리 등을 포함하여 프로그램 명령어가 저장되도록 구성된 것이 있을 수 있다. 또한, 다른 매체의 예시로, 애플리케이션을 유통하는 앱 스토어나 기타 다양한 소프트웨어를 공급 내지 유통하는 사이트, 서버 등에서 관리하는 기록매체 내지 저장매체도 들 수 있다.

[0069] 본 개시의 방법, 동작 또는 기법들은 다양한 수단에 의해 구현될 수도 있다. 예를 들어, 이러한 기법들은 하드웨어, 펌웨어, 소프트웨어, 또는 이들의 조합으로 구현될 수도 있다. 본원의 개시와 연계하여 설명된 다양한 예시적인 논리적 블록들, 모듈들, 회로들, 및 알고리즘 단계들은 전자 하드웨어, 컴퓨터 소프트웨어, 또는 양자의 조합들로 구현될 수도 있음을 통상의 기술자들은 이해할 것이다. 하드웨어 및 소프트웨어의 이러한 상호 대체를 명확하게 설명하기 위해, 다양한 예시적인 구성요소들, 블록들, 모듈들, 회로들, 및 단계들이 그들의 기능적 관점에서 일반적으로 위에서 설명되었다. 그러한 기능이 하드웨어로서 구현되는지 또는 소프트웨어로서 구현되는지의 여부는, 특정 애플리케이션 및 전체 시스템에 부과되는 설계 요구사항들에 따라 달라진다. 통상의 기술자들은 각각의 특정 애플리케이션을 위해 다양한 방식으로 설명된 기능을 구현할 수도 있으나, 그러한 구현들은 본 개시의 범위로부터 벗어나게 하는 것으로 해석되어서는 안된다.

[0070] 하드웨어 구현에서, 기법들을 수행하는 데 이용되는 프로세싱 유닛들은, 하나 이상의 ASIC들, DSP들, 디지털 신호 프로세싱 디바이스들(digital signal processing devices; DSPD들), 프로그램가능 논리 디바이스들(programmable logic devices; PLD들), 필드 프로그램가능 게이트 어레이들(field programmable gate arrays; FPGA들), 프로세서들, 제어기들, 마이크로제어기들, 마이크로프로세서들, 전자 디바이스들, 본 개시에 설명된 기능들을 수행하도록 설계된 다른 전자 유닛들, 컴퓨터, 또는 이들의 조합 내에서 구현될 수도 있다.

[0071] 따라서, 본 개시와 연계하여 설명된 다양한 예시적인 논리 블록들, 모듈들, 및 회로들은 범용 프로세서, DSP, ASIC, FPGA나 다른 프로그램 가능 논리 디바이스, 이산 게이트나 트랜지스터 로직, 이산 하드웨어 컴포넌트들, 또는 본원에 설명된 기능들을 수행하도록 설계된 것들의 임의의 조합으로 구현되거나 수행될 수도 있다. 범용 프로세서는 마이크로프로세서일 수도 있지만, 대안으로, 프로세서는 임의의 종래의 프로세서, 제어기, 마이크로제어기, 또는 상태 머신일 수도 있다. 프로세서는 또한, 컴퓨팅 디바이스들의 조합, 예를 들면, DSP와 마이크로프로세서, 복수의 마이크로프로세서들, DSP 코어와 연계한 하나 이상의 마이크로프로세서들, 또는 임의의 다른 구성의 조합으로서 구현될 수도 있다.

[0072] 펌웨어 및/또는 소프트웨어 구현에 있어서, 기법들은 랜덤 액세스 메모리(random access memory; RAM), 판독 전용 메모리(read-only memory; ROM), 비휘발성 RAM(non-volatile random access memory; NVRAM), PROM(programmable read-only memory), EPROM(erasable programmable read-only memory), EEPROM(electrically erasable PROM), 플래시 메모리, 콤팩트 디스크(compact disc; CD), 자기 또는 광학 데이터 스토리지 디바이스 등과 같은 컴퓨터 판독가능 매체 상에 저장된 명령들로서 구현될 수도 있다. 명령들은 하나 이상의 프로세서들에 의해 실행 가능할 수도 있고, 프로세서(들)로 하여금 본 개시에 설명된 기능의 특정 양태들을 수행하게 할 수도 있다.

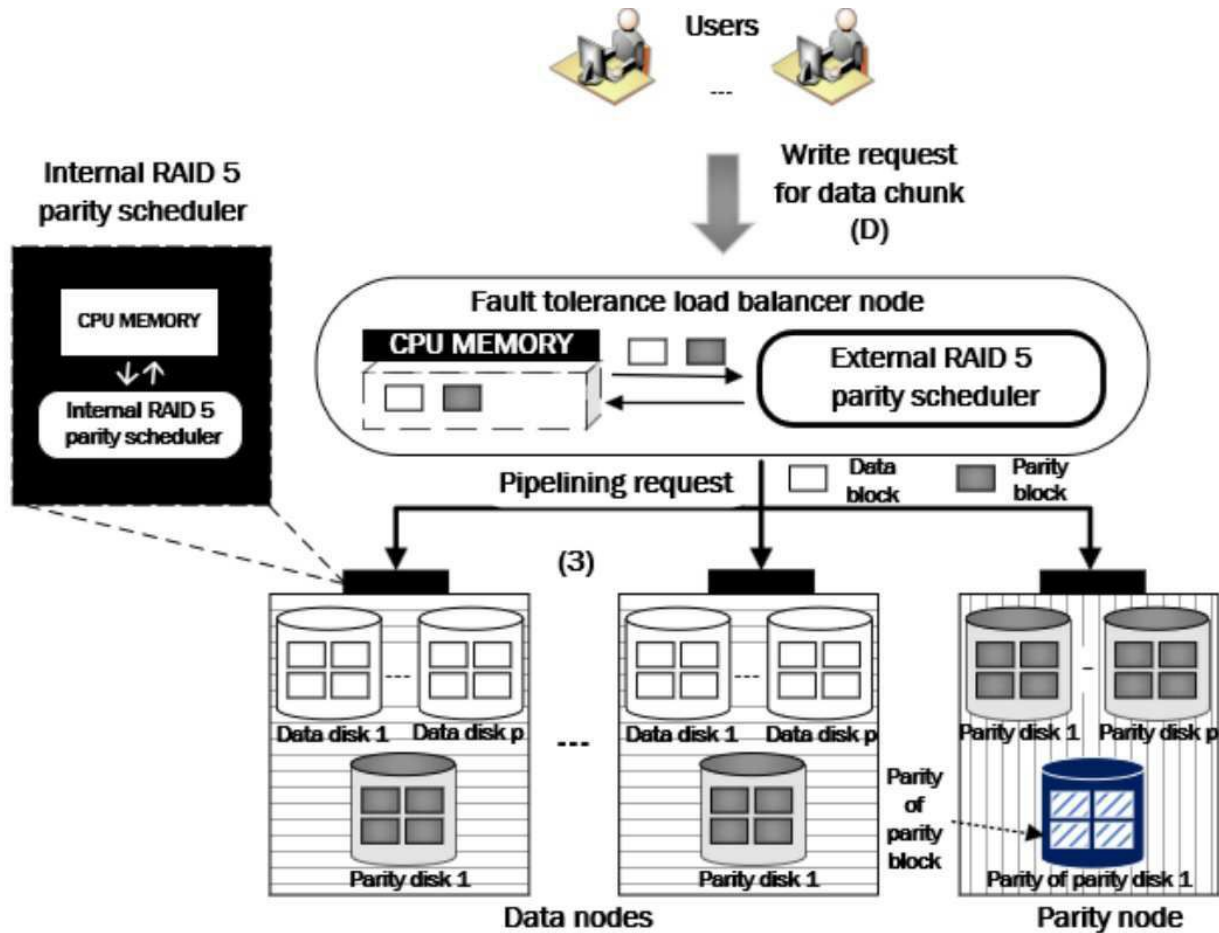
[0073] 이상 설명된 실시예들이 하나 이상의 독립형 컴퓨터 시스템에서 현재 개시된 주제의 양태들을 활용하는 것으로 기술되었으나, 본 개시는 이에 한정되지 않고, 네트워크나 분산 컴퓨팅 환경과 같은 임의의 컴퓨팅 환경과 연계하여 구현될 수도 있다. 또 나아가, 본 개시에서 주제의 양상들은 복수의 프로세싱 칩들이나 장치들에서 구현될 수도 있고, 스토리지는 복수의 장치들에 걸쳐 유사하게 영향을 받게 될 수도 있다. 이러한 장치들은 PC들, 네트워크 서버들, 및 휴대용 장치들을 포함할 수도 있다.

[0074]

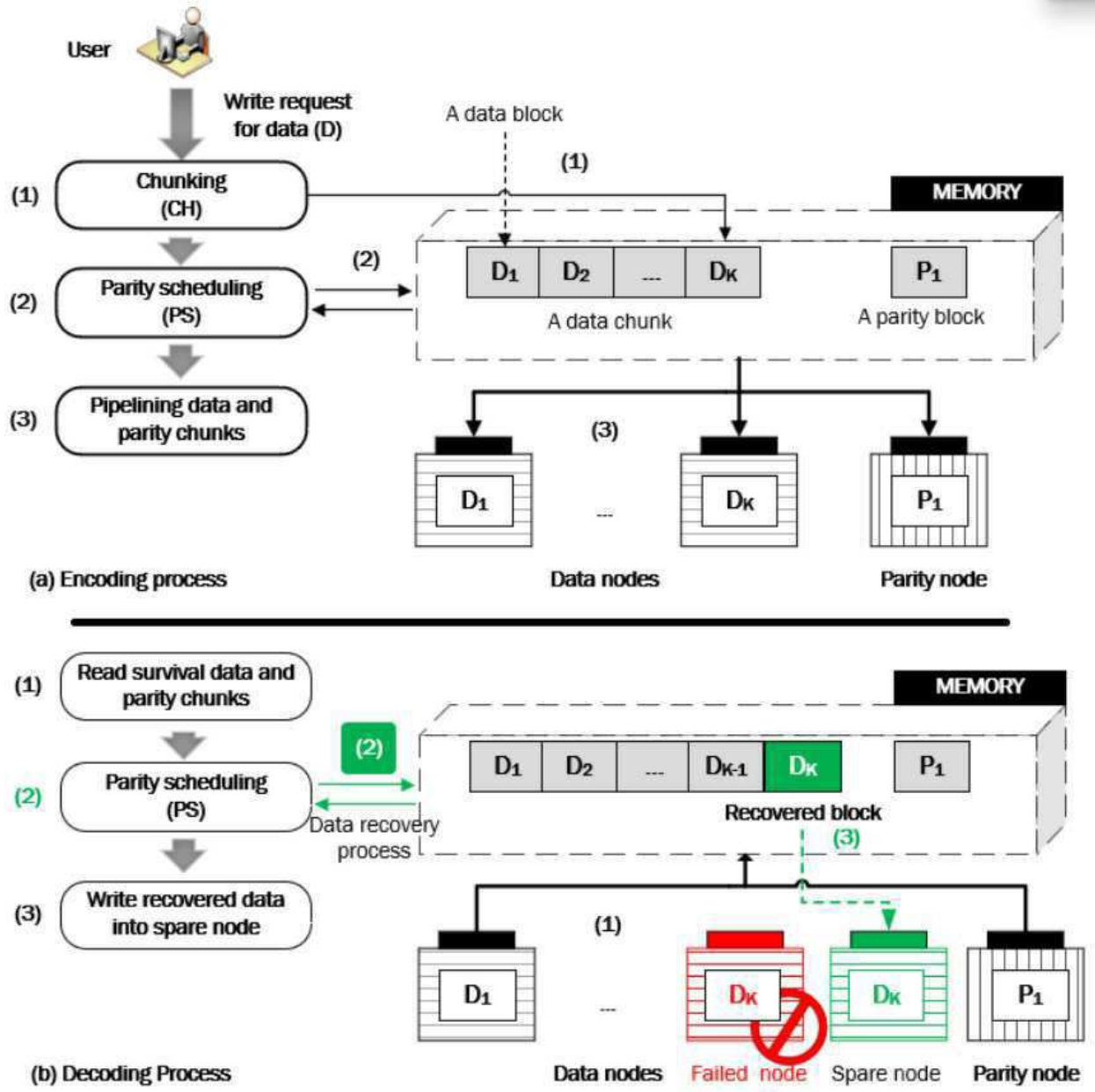
본 개시가 일부 실시예들과 관련하여 설명되었지만, 본 개시의 발명이 속하는 기술분야의 통상의 기술자가 이해할 수 있는 본 개시의 범위를 벗어나지 않는 범위에서 다양한 변형 및 변경이 이루어질 수 있다. 또한, 그러한 변형 및 변경은 본 명세서에 첨부된 특허청구의 범위 내에 속하는 것으로 생각되어야 한다.

도면

도면1



도면2



도면3

Algorithm 1: The parity scheduler

```

1 Function PS( $d[Kw]$ ,  $G$ ,  $K$ ,  $M$ ):
2   for  $j \leftarrow 0$  to  $K - 1$  do
3     for  $i \leftarrow 0$  to  $M - 1$  do
4        $p_i = p_i \oplus (d_j \times g_{i,j})$ .
5     end
6   end
7   return  $p[M]$ :  $m$  parity sectors.

```

도면4

Algorithm 2: The traditional CES1	
1	Function CES1():
2	$d[K] = \text{read data into external memory.}$
3	$p[m_1] = PS(d[K], K_1, m_1, G_1)$
4	<i>pipeline</i> data d and parity $p[m_1]$ into data and parity nodes.
5	// for each node
6	for $i \leftarrow 0$ to $(k_1 + m_1)$ do
7	// for each data disk in a given node.
8	$d[K_2] = \text{read data from disks into internal memory.}$
9	$p[m_2] = PS(d[k_2], K_2, m_2, G_2).$
10	<i>pipeline</i> parity $p[m_2]$ into parity disks.
11	for $j \leftarrow 0$ to $(K_2 + m_2)$ do
12	$d[K_3] = \text{read data sectors into internal memory.}$
13	$p[m_3] = PS(d[k_3], K_3, m_3, G_3).$
14	<i>write</i> parity sectors $p[m_3]$ into disks.
15	end
16	end

도면5

Algorithm 3: The traditional CES2	
1	Function CES2():
2	$d[K] = \text{read data into external memory.}$
3	$p[m_1] = PS(d[K], K_1, m_1, G_1)$
4	<i>pipeline</i> data d and parity $p[m_1]$ into data and parity nodes.
5	// for each node
6	for $i \leftarrow 0$ to $(k_1 + m_1)$ do
7	$d[K_2] = \text{read data from disks into internal memory.}$
8	$p[m_2 + m_3] = PS(d[k_2], K_2, (m_2 + m_3), G_{23}).$
9	<i>pipeline</i> parity $p[m_2 + m_3]$ into parity disks.
10	end

도면6

Algorithm 4: The traditional CES3

```

1 Function CES3():
2    $d[K]$  = reads the data into external memory.
3    $p[M] = PS(d[K], K, M, G)$ 
4   Pipeline data  $d$  and parity  $p[M]$  into the data,
   parity nodes, and disks.

```

도면7

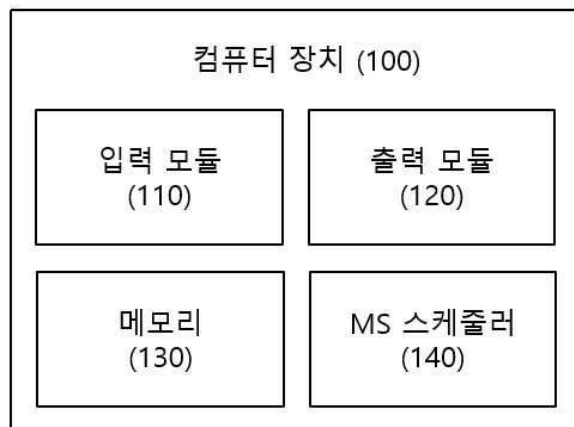
Algorithm 5: AVX-512 Parity scheduler

```

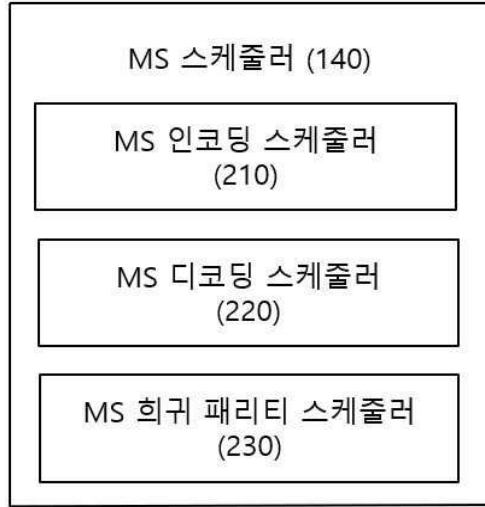
1 Function PS512( $p$  *,  $c$ ,  $d$  *,  $length$ ):
2   Set  $reg1$ ,  $reg2$ ,  $ri[2]$  as 512 bits register.
3   Set  $sp[2]$ ,  $mi[2]$  as 512 bits register.
4   Set *  $rg = pow[c]$  as 8 bits integer pointer.
5    $mi[0] \leftarrow$  Allocate 8bits of 0x55.
6    $mi[1] \leftarrow$  allocate 8bits of 0xaa.
7    $sp[0] \leftarrow$  Allocate  $g[0]$ .
8    $sp[1] \leftarrow$  Allocate  $g[1]$ .
9   for  $j \leftarrow d + length$  to  $d < j$  do
10     $reg1 \leftarrow$  Load 512 bits from  $p$  pointer.
11     $ri[0] \leftarrow$  Add 512 bits of  $reg2$  and  $mi[0]$ .
12     $ri[1] \leftarrow$  Add 512 bits of  $reg2$  and  $mi[1]$ .
13     $ri[1] \leftarrow$  Make one bit right shift in  $ri[1]$ .
14     $ri[0] \leftarrow$  Multiply  $ri[0]$  and  $sp[0]$ , and store the low half.
15     $ri[1] \leftarrow$  Multiply  $ri[1]$  and  $sp[1]$ , and store the low half.
16     $ri[0] \leftarrow$  XOR of  $ri[0]$  and  $ri[1]$ .
17     $ri[0] \leftarrow$  XOR of  $ri[0]$  and  $reg1$ .
18    Store 512-bits from  $ri[0]$  into memory  $p$ .
19     $p+ = 64$ .
20     $d+ = 64$ .
21 end
22 return  $p$ :

```

도면8



도면9



도면10

Algorithm 6: The proposed PCES

```

1 Function PCES():
2    $d[K]$  = read data into external memory.
3   Load sparsified MS coding matrix into memory.
4    $p[M] = SPS(d[K], K, M, G)$ 
5   pipeline data  $d$  and parity  $p[M]$ .

```

도면11

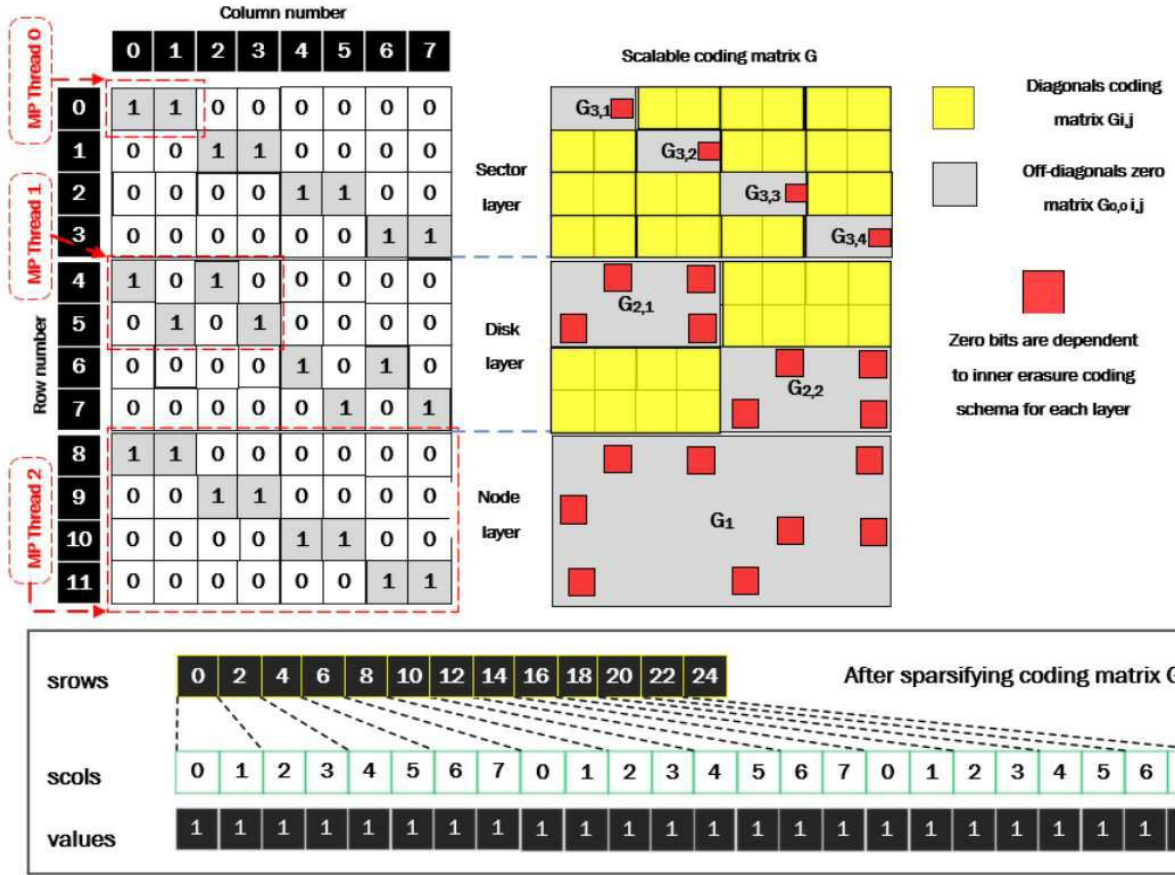
MS Decoder ($^1G', ^2\Phi(G'), K, M$).

1. Read K sectors and transfer them to the host.
 2. if $(M < (K - \Phi(G')))$ return.
 3. if $(1 \leq \Phi(G') \leq M)$ {
 4. Allocate the survival data and parity to ds .
 5. Generate B by deleting the failed columns of G_i
 6. Load sparsified B^{-1} into memory.
 7. $d' = SPS(ds[K], \Phi(G'), K - \Phi(G'))$.
 8. Merge ds and d' into $d[K]$.
 9. Transfer $d[K]$ from main memory to the host.
 10. }
-

¹ G' contains the group of survival data and parity.

² $\Phi(G)$ denotes the number of survival data and parity.

도면12



도면13

Algorithm 7: Atomic, Open MP, and AVX-based SPMV matrix XOR multiplication

```

1  Function SPMV():
2      Load rows, cols, matrix_value_offset.
3      Load temp_parity[], prefetch_data[].
4      Load parity[], values[].
5      Load scols[], srows[].
6      # pragma omp parallel for
7      for (c ← 0; c < rows; c++) do
8          p ← 0.
9          # pragma omp simd
10         for (j ← srows[c]; j < srows[c + 1]; j += 64) do
11             temp_parity[c * rows + j] ← values[j] *
                prefetch_data[j].
12         end
13         # pragma omp atomic
14         for (j ← srows[c]; j < srows[c + 1]; j++) do
15             parity[c] ← Perform the XOR operation between
                parity[c] and temp_parity[c * rows + j].
16         end
17     end
18     return parity;

```

도면14

Algorithm 8: The proposed OpenMP AVX-based sparsified coding matrix

```

1 Function sparsify_coding_matrix():
2   Load  $M[], rows, cols, prefetch\_data[]$ .
3   Load  $data[], matrix\_value\_offset$ .
4   Load  $values[], scols[], srows[]$ .
5    $SV \leftarrow 0$ .
6    $Sr \leftarrow 0$ .
7    $srows[0] \leftarrow 0$ .
8   # pragma omp parallel for
9   for ( $r \leftarrow 0, r < rows, r++$ ) do
10    for ( $r \leftarrow 0, r < rows, r++$ ) do
11      # pragma omp simd
12      if ( $M[r * cols + c] \neq 0$ ) then
13         $values[sv] \leftarrow M[r * cols + c]$ .
14         $prefetch\_data \leftarrow data[c]$ .
15         $scols[sv] \leftarrow c$ .
16         $sv++$ .
17      end
18       $srows[++sr] \leftarrow sv$ .
19    end
20  end

```
