Journal of KIIT. Vol. 17, No. 11, pp. 17-26, Nov. 30, 2019. pISSN 1598-8619, eISSN 2093-7571 17
http://dx.doi.org/10.14801/jkiit.2019.17.11.17

플래시 저장 시스템의 Full Stripe Parity를 위한 메타데이터 로그 관리 방법

임 승 호*

Metadata Log Management for Full Stripe Parity in Flash Storage Systems

Seung-Ho Lim*

이 연구는 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2019R1F1A1057503).

요 약

플래시 스토리지 장치의 신뢰성을 향상시키기 위해서 사용되는 기술 중의 하나가 RAID-5 기술이 있다. RAID-5에는 고유한 패리티 업데이트 오버헤드가 있는데, 특히 부분 스트라이프 쓰기에 대한 패리티 오버헤드는 플래시 기반 RAID-5 기술의 중요한 문제 중 하나이다. 본 논문에서는 RAID-5에서 발생하는 런타임 부분 패리티 오버헤드를 제거하기 위해 효율적인 패리티 로그 아키텍처를 설계하였다. 런타임 동안, 전체 스트라이프 쓰기가 완료될 때까지 부분 패리티가 버퍼 메모리에 유지되며, 스트라이프 쓰기가 완료될 때 패리티는 전체 스트라이프 쓰기로 기록된다. 페리티 로그는 전체 스트라이프 그룹이 데이터 쓰기에 사용될 때까지 메모리에서 유지된다. 이 패리티 로그를 사용하면 갑작스러운 전력 손실로부터 부분 패리티를 복구할 수 있으므로 데이터 손실에도 문제가 발생하지 않는다. 패리티 로그 방법은 작은 패리티 로그 양으로 부분 패리티 쓰기 오버헤드를 제거할 수 있으므로, 같은 신뢰성 수준에서 쓰기 오버헤드를 줄일 수 있다.

Abstract

RAID-5 technology is one of the choice for flash storage device to enhance its reliability. However, RAID-5 has inherent parity update overhead, especially, parity overhead for partial stripe write is one of the crucial issues for flash-based RAID-5 technologies. In this paper, we design efficient parity log architecture for RAID-5 to eliminate runtime partial parity overhead. During runtime, partial parity is retained in buffer memory until full stripe write completed, and the parity is written with full strip write. In addition, parity log is maintained in memory until whole the stripe group is used for data write. With this parity log, partial parity can be recovered from the power loss. In the experiments, the parity log method can eliminate partial parity writes overhead with a little parity log writes. Hence it can reduce write amplification at the same reliability.

Keywords
RAID-5, partial parity, full stripe write, StripeMapLog, recovery

* 한국외국어대학교 컴퓨터.전자시스템공학부 교수
- ORCID: https://orcid.org/0000-0003-3096-0785
· Received: Aug. 30, 2019, Revised: Oct. 23, 2019, Accepted: Oct. 26, 2019
· Corresponding Author: Seung-Ho Lim
Division of Computer and Electronic Systems Engineering, Hankuk Universithy of Foreign Studdies, Korea,
Tel.: +82-31-330-4704, Email: lim.seungho@gmail.com

Ⅰ. 서 론

낸드 플래시 메모리는 최근에 셀 당 더 많은 비트를 저장함으로써 저장 용량 및 밀도가 증가했다. 밀도가 증가하면 플래시 메모리의 내구성이 저하되어 가용한 P/E cycle이 많이 줄어든다[1][2]. 이러한 짧은 내구성과 오류 증가에 대한 일반적인 접근 방식은 각 페이지의 OOB(Out-of-Band) 영역에 저장된 ECC(Error Correction Code)를 사용하는 것이다[3][4]. 그러나 밀도 증가로 인해서 증가한 오류는 더 강력한 ECC가 필요하며, 게다가 ECC는 버스트 오류, 페이지, 블록 및 칩 수준 오류를 수정하기가 어렵다. ECC의 제한사항을 보완하기 위해 RAID-5와 같은 안정성 보장 기술이 적용될 수 있다[5]. 이전의 많은 연구 결과에 따르면 RAID-5 기술을 사용한 패리티는 ECC보다 페이지 오류 비율을 줄일 수 있다고 주장한다.

그러나 RAID-5 아키텍처는 일반적으로 패리티 생성 및 저장방법에 오버헤드가 있으며, 데이터에 대한 많은 읽기 및 쓰기를 동반한 패리티 연산이 필요하다. 패리티 업데이트 오버헤드를 줄이는 문제는 RAID-5 아키텍처의 주요 문제이다. 패리티 오버헤드를 줄이기 위한 몇 가지 기존 연구 결과가 있다. FRA[6]는 일정 유휴 시간 동안 버퍼 메모리에 연기 패리티 쓰기에서 패리티 블록을 유지하므로 읽기 작업 및 쓰기 응답 시간을 줄일 수 있지만, 전체 스트라이프가 기록될 때까지 패리티 업데이트가 연기되는 도중, 부분 스트라이프 쓰기 포인트에서 시스템 충돌을 복구하는 방안이 포함되어 있지 않다. PPC[7]는 부분 스트라이프에 대한 부분 패리티를 생성하므로 부분 쓰기가 가능하지만 NVRAM에서 부분 패리티가 유지되므로 추가적인 비휘발성 RAM (NVRAM) 하드웨어가 필요하다. eSAP[8]는 가변 크기 스트라이핑을 사용하여 전체 스트라이프의 일부에 기록된 데이터로 새 스트라이프를 구성하고, 추가 하드웨어 지원 없이 해당 부분 스트라이프에 대한 패리티를 생성하여 기록한다. 그러나 이 방법은 플래시에 기록되는 부분 패리티 데이터를 너무 많이 생성하므로 쓰기 성능이 나빠질 수 있으며, 쓰기 증폭을 발생시키는 요인이 된다.

이 논문에서는 부분 패리티 오버헤드를 줄이기 위해 RAID-5에 대한 스트라이프 정보에 대해서 메타데이터 로그를 생성하고 저장 관리하는 구조를 설계하였다. 제안된 설계에서는, 새로운 쓰기 데이타에 대해서 전체 스트라이프 쓰기가 기록될 때까지 부분 패리티가 버퍼 메모리에 존재한다. 이 패리티 버퍼의 내용은 스트라이프에 대한 데이터 페이지 쓰기 연산이 발생할 때마다 패리티 데이터가 업데이트된다. 이와 동시에 쓰기가 발생한 데이터 페이지의 해당 매핑 정보가 로그 구조에 기록된다. 메모리 버퍼로 관리되는 패리티와는 별도로, 사용자 데이터는 부분 스트라이프로 구성되어 있어도 정상적인 절차로 플래시에 기록된다. 스트라이프의 모든 데이터가 플래시에 기록되면, 즉, 해당 스트라이프의 데이터에 해당하는 패리티 계산이 완료되면, 해당 패리티가 최종적으로 고정되고 스트라이프의 적절한 위치로 플러시 된다.

제안된 시스템에서는 일반 데이터가 기존의 시스템과 같은 경로이므로 일반 RAID-5 시스템과 비교하여 일반 데이터 쓰기 작업에 대해서 추가로 발생하는 오버헤드는 크지 않다. 추가로 발생하는 오버헤드는 부분 패리티를 메모리에서 관리하기 위한 페이지 버퍼와 스트라이프 그룹을 위한 메타 데이터 로그를 기록하기 위한 로그 쓰기 연산 및 공간에 대한 오버헤드이다. 패리티 업데이트의 오버헤드에 대해서는, 부분 패리티 쓰기 연산이 없으므로, 패리티 쓰기에 대한 오버헤드를 줄일 수 있다. 또한, 메모리에서 업데이트되던 패리티가 전원 유실로 인해 잃어버렸을 경우, 해당 데이터들의 기록된 로그를 이용하여 부분 스트라이프의 데이터를 추적하여, 부분 패리티를 복원할 수 있다.

Ⅱ. 배경 지식 및 전반적인 설계

2.1 낸드 플래시 메모리

낸드 플래시 메모리는 플로팅 게이트 기반의 반도체 소자를 집적하여 구성한 비휘발성 반도체 저장장치이며, 주요 구성요소로는 읽기 및 쓰기 단위인 페이지, 지우기 단위인 블록 단위가 있다. 페이지 크기는 주로 4KB, 8KB, 16KB 정도이며, 수십개의 페이지가 모여서 하나의 블록을 구성한다. 낸

드 플래시 메모리는 집적도의 향상으로 저장 용량은 큰 증가를 이루었지만, 인접 셀 간의 간격이 좁아지고, 하나의 셀 당 저장할 수 있는 저장 단위의 증가로 인해서 셀 간의 간섭도가 증가하였으며, 이로 인해서 읽기 및 쓰기 도중에 발생할 수 있는 오류도 급격히 증가하게 되었다. 일반적으로 낸드 플래시 메모리의 수명을 나타내는 지표인 P/E cycle 값이 SLC(Single Level Cell)의 경우 100,000 정도이나, MLC(Multi Level Cell)의 경우 10,000 이하, TLC(Triple Level Cell)의 경우 수천 이하로 급격히 에러 발생률이 늘어가고 있으며, 낸드 플래시 메모리 기반 저장장치의 수명이 주요 이슈로 주목받았다. 이러한 에러에 대처하는 방안으로 BCH, LDPC와 같은 ECC 방법이 낸드 플래시 메모리 컨트롤러에 내장되고 있으며, RAID-5와 같은 블록 수준 또는 칩 수준의 에러 복원 기술이 적용되고 있다.

2.2 RAID-5 설계 방향

본 논문의 스트라이프 로그 아키텍처의 설계 목표는 RAID-5를 지원하는 플래시 스토리지 시스템의 패리티 업데이트 오버헤드를 최소화하는 것이다. 읽기 및 쓰기 요청에 대한 다른 모든 일반 작업은 기존 RAID-5 플래시 스토리지 시스템과 동일하다. 이는 FTL(Flash Translation Layer)[9] 매핑 테이블과 같은 플래시 스토리지 관리를 위한 필수 메타 데이터 아키텍처가 다른 플래시 시스템과 유사하거나 다른 플래시 시스템으로 대체 될 수 있음을 의미한다. 매핑 테이블 관리는 이 논문의 주요 문제가 아니며, 이 논문의 주요 디자인 문제는 패리티 업데이트 오버헤드를 최소화하면서 RAID-5를 지원하도록 하는 메타데이터(Metadata) 로깅 아키텍처 및 해당 메타데이터 작업을 설계하는 방법이다.

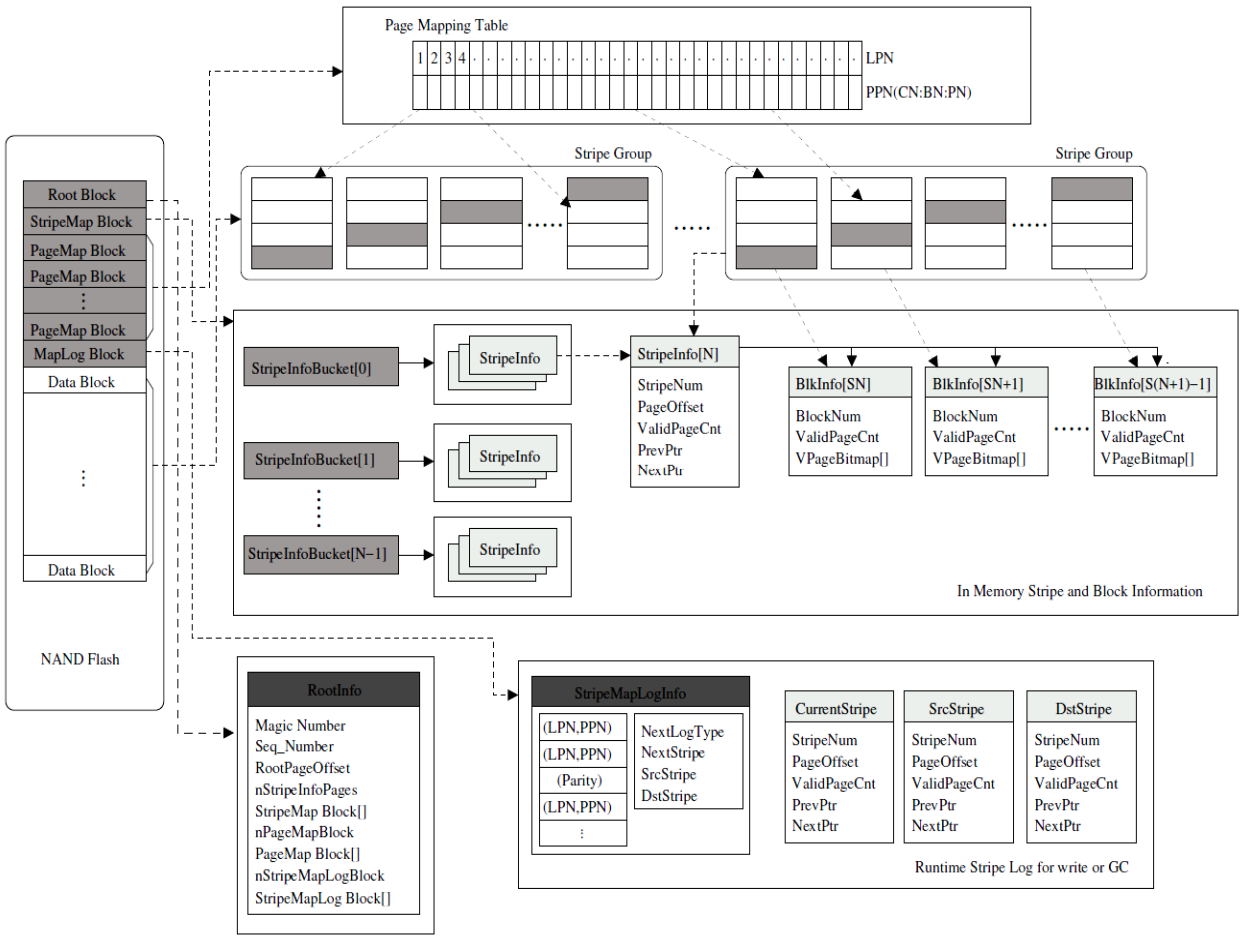

그림 1. RAID-5 지원을 위한 스트라이프 메타데이터 및 로그 아키텍처
Fig. 1. Stripe metadata and Log architecture for RAID-5 support

전체적인 메타 데이터 아키텍처는 그림 1에 도식화 하였다. 그림과 같이 본 논문에서 설계한 전체 메타 데이터 구조는 Root 블록, Stripe Map 블록, PageMap 블록 및 StripeMapLog 블록 등으로 구성된다. NAND 플래시 메모리 내에 저장되는 이러한 메타데이터 블록에는 메타데이터의 스냅샷(Snapshot) 상태가 포함되어 있으며, RAM과 같은 메인 메모리에는 낸드 플래시 메모리 스냅샷 이후에 스토리지 연산에 의해서 업데이트 된 최신 메타데이터 정보가 기록되어 있다. 각각의 메타데이터 블록의 의미는 다음과 같다.

Root 블록에는 메타데이터 블록 (예: StripeMap 블록, PageMap 블록 및 StripeMapLog 블록)이 저장된 블록 번호가 기록된다. StripeMap 블록에는 각 스트라이프 데이터 그룹 블록 및 스트라이프 그룹 정보가 기재되어 있다. 본 논문에서 스트라이프 그룹은 패리티 클러스터를 구성하는 블록 그룹을 의미한다. 스트라이프 사용정보를 나타내기 위해서, BlkInfo[], StripeInfo[] 및 StripeInfoList[]등 세 가지 데이터 구조가 존재한다. 자세한 내용은 다음 장에서 설명한다.

본 논문에서 적용한 FTL의 매핑 테이블 관리 방법은 페이지 각각에 대한 페이지 수준의 매핑 관리를 사용하며, 이는 각 논리적 페이지가 각 물리적 페이지에 할당되어 있음을 나타낸다. 즉, NAND 플래시 메모리 내의 PageMap 블록에는 논리 페이지와 실제 페이지 간의 페이지 수준 매핑 정보 스냅샷이 포함되어 있으며, RAM 영역의 페이지 매핑 테이블은 최신 매핑 정보를 유지한다.

본 논문에서 설계하는 로깅 전략은 데이터 로깅이 아닌 메타 데이터 로깅을 위한 것이다. 따라서 데이터 쓰기가 발생할 때마다 해당 쓰기 데이터는 낸드 플래시 미디어 쓰기 연산이 수반되며, 이에 더해서 해당 쓰기 연산과 관련된 메타 데이터의 로깅 작업 및 패리티 업데이트 작업을 추가로 설계한 것이다. 본 시스템에서, RAID-5 아키텍처의 패리티 데이터는 스트라이프의 페이지 레벨에서 계산된다. RAID-5에 대한 부분 페리티 연산과 스트라이프 그룹의 기록 동작에 대한 흐름도를 그림 2에 나타내었다.

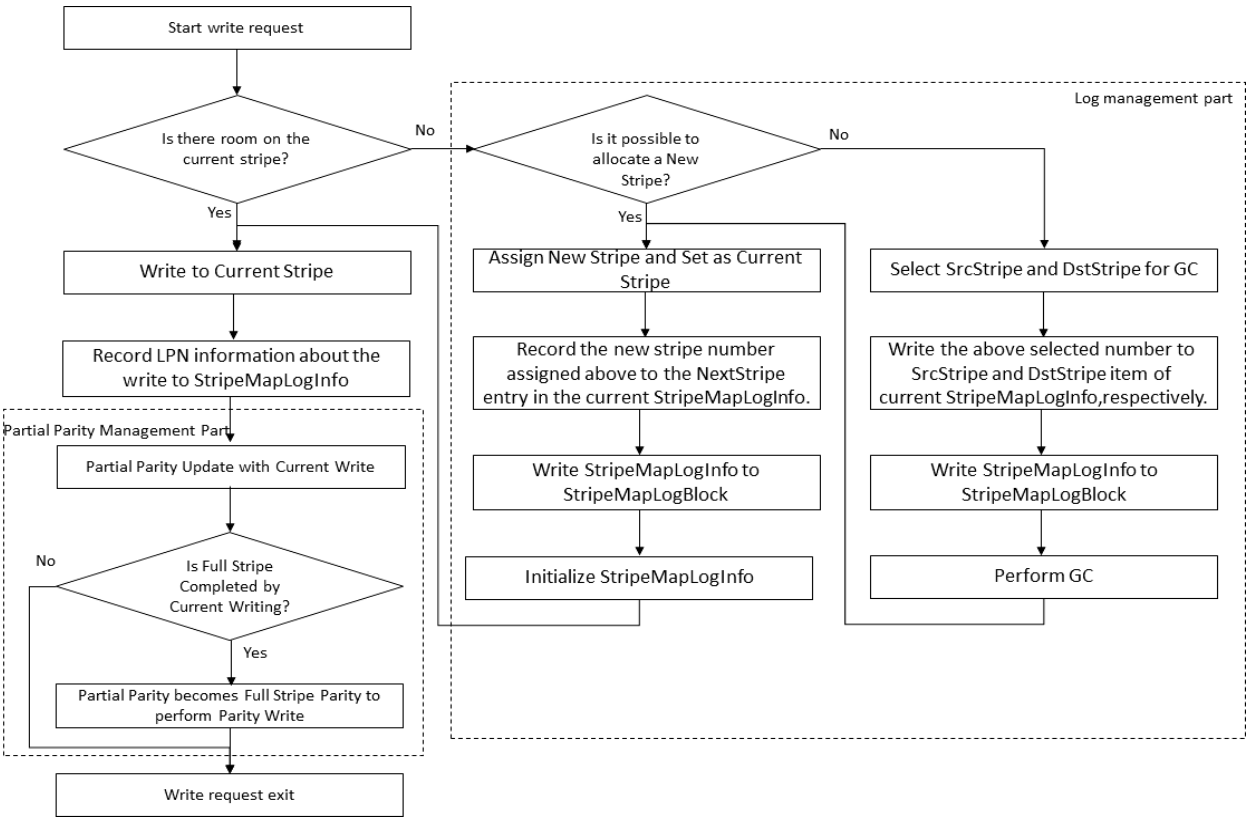

그림 2. RAID-5 패리티 스트라이프 쓰기 및 부분 패리티 관리를 대한 StripeMapLog 동작 흐름도
Fig. 2. Flowchart of logging operations StripeMapLogInfo and partial parity for RAID-5

그림에 나타난 바와 같이, 스트라이프에 대한 패리티는 전체 스트라이프 쓰기가 완료될 때까지 버퍼 메모리에 유지되는 반면, 스트라이프의 데이터 영역은 전체 스트라이프 쓰기가 완료되는 것과 관계없이 기록된다. 플래시 저장장치의 쓰기는 항상 스트라이프 영역의 새로운 영역에 발생하며, 이에 대해서 패리티가 업데이트되므로 스트라이프의 부분 패리티를 업데이트하기 위해 이전 데이터를 읽을 필요가 없다. 마지막에, 전체 스트라이프가 완료되면 해당 스트라이프에 대해 완성된 패리티가 플래시 미디어에 쓰이게 된다.

부분 패리티의 경우 메모리에 유지되므로 정전으로 인해 손실될 수가 있다. 정전에서 부분 패리티를 복구하기 위해 스트라이프 로깅이 적용된다. 설계된 로깅 시스템에서 현재 스트라이프에 대한 매핑 변경 사항은 StripeLogInfo의 구조로써 메모리 영역에 존재한다. 현재 스트라이프 그룹 StripeMapLogInfo의 매핑 로그 정보는 해당 스트라이프의 데이터 쓰기 영역이 완료되어 해당 스트라이프의 패리티가 플러시 될 때, 전용 플래시 블록에 StripeMapLog의 형태로 페이지 단위로 플러시 된다. 만약, 패리티가 버퍼에 보유된 부분 상태에 있는 동안 전원 손실이 발생하면, 메모리에 로깅된 정보 역시 분실된다. 그러나 이 메타정보는 정전이 발생하기 직전 쓰기가 진행중이었던 스트라이프의 플래시 메모리 OOB (Spare) 영역에 기재가 되어 있기 때문에, 해당 스트라이프 영역을 스캔하여 해당 부분 패리티를 복구할 수 있다. 세부적인 동작 방식은 다음 장에서 설명한다.

III. RAID Metadata Log

3.1 Metadata 구조

앞장에서 설명한 대로 전체 메타 데이터 구조가 그림 1에 표시되어 있다. 그림에 나타난 바와 같이, 플래시 메모리 블록 0 또는 1 둘 중에서 하나가 피벗 루트 블록으로 사용된다. 처음 시작 시, 블록 0이 루트 블록에 할당된다. 루트 블록의 마지막 업데이트된 페이지에는 메타데이타의 정보가 기록된 블록들의 위치가 있는 RootInfo 값을 가지고 있다. 메타 블록이 변경될 때마다 RootInfo는 Root 블록의 한 페이지에 로그 방식으로 업데이트된다. 업데이트가 현재 루트 블록의 마지막 페이지 (예 : 블록 0)에 도달하면 루트 블록이 블록 1로 변경되고 업데이트가 계속된다. StripeMap 블록에는 각 스트라이프, 데이터 블록 및 스트라이프 버킷 정보가 기록된다. 스트라이프 사용현황 등의 정보를 위해서 BlkInfo[], StripeInfo[] 및 StripeInfoList[]의 세 가지 데이터 구조가 있다. BlkInfo[]는 각 데이터 블록에 대한 상태 정보를 가지며, StripeInfo[]는 각 스트라이프 그룹의 상태 정보를 가지고 있다. 스트라이프 그룹의 크기는 해당 스트라이프에 포함된 데이터 블록의 개수이다. 예를 들어, 스트라이프 크기가 S이면 하나의 스트라이프 그룹은 S 블록으로 구성된다. StripeInfoList[]는 0에서 (S*N)까지 버킷을 갖는 해시 데이터 구조이다. 여기서 N은 스트라이프 그룹 내 유효한 페이지 수이다. 각 버킷에는 같은 유효한 페이지 수를 가진 이중 연결 스트라이프 그룹 목록이 있다. 이 목록을 사용하여 다음 쓰기를 위한 프리 스트라이프 그룹과 GC가 필요할 때 수행할 희생 스트라이프 그룹을 선택할 수 있다. PageMap 블록은 페이지 매핑 테이블 정보를 가진다. 페이지 매핑 테이블의 각 항목은 실제 페이지 번호 (PPN)의 배열이다. 메타 데이터의 마지막 부분은 StripeMapLog 블록으로, 쓰기에 대한 메타 데이터 정보를 로깅 하는 데 사용된다.

NAND 플래시 메타 블록은 최신 버전의 스냅샷을 나타내며 메인 메모리에는 스냅샷 이후에 변경된 최신 메타 데이터 정보가 있다. 쓰기가 발생할 때마다 모든 데이터가 플래시 스토리지 시스템의 새로운 클린 영역에 기록되므로 메타데이터가 업데이트가 발생한다. 변경된 메타데이터 정보는 특정 시점마다 낸드 플래시 메모리로 보내져서 최신 메타데이터 정보를 유지할 수 있도록 한다. 이렇게 보내지는 정보가 로그 정보이다.

3.2 StripeMapLogInfo 데이터

사용자의 쓰기 연산에 대한 전체적인 로그 동작의 흐름도가 그림 2에 도식화되어 있다. 최신 스냅샷 이후, 업데이트되는 메타 데이터 정보는 메모리

내 StripeMapLogInfo 구조 정보를 생성하여 차례로 기록되며 일정량의 메타 데이터 업데이트로 채워지면 StripeMapLog 블록에 하나 이상의 페이지로 스트라이프를 구성하여 낸드 플래시 메모리에 로그 형태로 플러시(Flush) 된다. 만약, StripeMapLog 블록 로딩된 페이지로 가득 차면 로깅 할 공간이 없게 되어, 다음 사용 가능한 블록 영역이 필요하다. 새로운 StripeMapLog 블록을 할당하기 전에 메인 메모리 내 메타 데이터 정보가 NAND 플래시 메모리에 동기화되어 최신 스냅샷이 된다. 그 후 새로운 StripeMapLog 블록 할당한다. 이 과정을 거친 후, 앞서 설명한 통상적인 로깅 메커니즘이 수행된다.

StripeMapLogInfo는 스트라이프 그룹에 있는가각 스트라이프의 논리적 주소에 대한 배열로 구성되며 논리적-물리적 매핑 정보를 가지게 된다. 스트라이프 그룹은 RAID 스트라이프를 구성하기 위해 그룹화된 블록 그룹이다. 스트라이프 그룹은 블록 레벨이지만 스트라이프에 대한 패리티는 그룹 내의 페이지 레벨이다. 예를 들어서 4개의 블록으로 스트라이프를 구성할 경우, 패리티는 그룹 내에서 같은 페이지 오프셋을 가진 3개의 페이지를 기반으로 생성된다. 메타 데이터 로깅 절차는 다음과 같다. 새로운 쓰기가 발생할 때마다 새 데이터가 스트라이프 그룹 내의 새 페이지에 할당된다. 그런 다음 이 스트라이프의 물리 페이지에 대한 논리-물리 주소 매핑 쌍 값들이 메모리의 StripeMap LogInfo에 기록된다. 한편, 데이터가 플래시 페이지에 기록될 때 LPN은 페이지의 OOB(Spare) 영역에 저장되는데, 이는 전원 손실 복구 및 갑작스런 전원 손실로 인한 패리티 복구에 중요하다.

로그 변경 외에도 StripeMapLogInfo에는 다음 스트라이프 그룹에 대한 정보를 가지고 있는데, NextLogType 및 NextStripe 또는 SrcStripe 및 DstStripe이라는 필드에 해당 정보를 가지고 있다. NextLogType은 현재가 아닌 다음 로그의 유형을 나타낸다. 스트라이프 그룹 변경에는 두 가지 유형이 있는데, 하나는 사용자 데이터 쓰기를 위한 스트라이프 그룹이고 다른 하나는 GC를 위한 스트라이프 그룹이다. 이에 따라 두 가지 유형의 로그가 있다. NextLogType이 0이면 다음 로그 유형은 사용자 쓰기이며 정상적인 쓰기 요청에 대해 다음 스트라이프가 선택되며, NextLogType이 1이면 다음 로그 유형은 GC이며 GC 작업을 위한 소스 스트라이프 및 대상 스트라이프가 선택된다. NextLog 유형 및 다음 스트라이프 그룹의 정보는 시스템 복구 시 중요한 역할을 한다. NextLogType을 설정하는 방식은 다음과 같다. 현재 쓰기 요청에 대해서 새 스트라이프 그룹이 필요할 경우, 사용 가능한 스트라이프 그룹 풀에서 새 스트라이프 그룹을 선택한다. 스트라이프 그룹 풀에 프리 스트라이프 그룹이 있어서 이를 선택하면 선택된 스트라이프 그룹 넘버를 StripeMapLogInfo 구조의 NextStripe 필드에 기록하며 NextLogType을 0으로 설정한다. 만약, 사용 가능한 스트라이프 그룹이 충분하지 않은 경우, 특정 스트라이프 그룹을 다시 확보하기 위해 회수해야 한다. 이 경우가 GC에 해당하며, 이때, GC에 필요한 GC 희생 스트라이프 그룹과 희생 그룹으로부터 옮겨갈 목적지 스트라이프 그룹을 선택한다. 선택한 희생 스트라이프는 SrcStripe에, 목적지 스트라이프는 DstStripe에 기록된다.

3.3 부분 패리티의 복구 방법

메모리에 유지되는 부분 패리티는 정전으로 인해 손실될 수 있는데, 이는 기록 된 메타 데이터로 복구할 수 있다. 풀 스트라이프 쓰기에 대한 패리티 상태로 거의 패리티 데이터가 플래시 메모리에 저장되므로, 풀 스트라이프에 대한 패리티는 복구할 필요가 없고, 부분 스트라이프 쓰기 도중 전원이 복구할 실제 부분은 부분 패리티와 관련된 스트라이프 영역만 복구해주면 된다.

부분 패리티 복구 단계는 다음과 같다. 먼저, 부팅의 첫 번째 단계에서 SSD는 Root 블록을 스캔하여 최신 RootInfo를 찾는다. 두 번째 단계에서는 StripeMap 및 PageMap 블록을 검색하여 StripeInfo List[], StripeBlkInfo[], BlkInfo[] 및 페이지 매핑 테이블로 구성된 메타 데이터 스냅샷을 읽어서 스냅에 대한 메타데이터를 복원한다. 세 번째 단계에서는 StripeMapLog 블록에 저장된 메타 데이터 변경 사항이 복구된다. 이후의 단계가 부분 패리티를 복구하는 단계이다. 마지막 단계에서 StripeMapLog 블록에 기록되지 않은 매핑 변경 사항을 복구하고, 이와

관련된 부분 패리티를 복구함으로써 복구가 마무리 되는데, 이는 마지막 StripMapLog 블록의 마지막 유효 페이지의 StripeMapLogInfo에 기재되어 있는 NextStripe에 해당하는 스트라이프 그룹을 스캔함으로써 이 작업이 가능하다. 이 NextStripe에 해당하는 스트라이프 그룹이 바로 쓰기 연산을 수행하다가 전원 손실이 발생한 스트라이프 그룹이기 때문이다. 먼저, NextStripe의 메타데이터 변경 사항은 해당 스트라이프 그룹의 각 페이지의 스페어 영역(OOB)을 스캔하여 복구할 수 있다. 이는, 각 페이지의 스페어 영역에 해당 페이지의 논리 주소(LPN)이 적혀 있기 때문이다. 모든 스트라이프 페이지 스캔에 대해 모든 페이지에 유효한 논리 주소가 있으면 전체 스트라이프가 안전하게 저장되고 해당 패리티도 안전하다는 의미이다. 유효한 LPN 정보를 가지는 페이지에 대해서는 그와 연관된 메타 데이터를 업데이트하여 준다. 일부 페이지에 유효한 논리 주소가 없으면 이 스트라이프의 모든 데이터가 유효하지 않기 때문에 이 스트라이프에 대한 완전한 전체 스트라이프 패리티가 없다. 즉, 부분 패리티를 메모리에 유지하던 도중 전원 손실이 발생하였다는 의미이다. 이 경우, 해당 스트라이프의 각 유효한 페이지를 검색하여, 유효한 페이지의 데이터를 읽어 패리티 연산을 해줌으로써 부분 패리티를 복구할 수 있다, 이렇게 함으로써 최종적으로 전원 손실에 대한 복구를 수행할 수 있게 된다.

Ⅳ. 성능평가

본 논문에서 설계한 스트라이프 관리 및 로깅 방식은 플래시 메모리 기반의 SSD 시뮬레이터인 Flashsim 공개 소스 시뮬레이터를 기반으로 하여 구현하였다[10][11].

그림 3은 공개 소스 시뮬레이터에 기반을 두어 본 논문에서 설계한 스트라이프 로그관리 방식을 구현한 소스코드 파일 일체와 실제 성능평가를 한 결과의 예를 캡처한 자료이다. 성능평가 방식은 두 개의 낸드 플래시 메모리 규격에 대해서 페이지 매핑 방식의 FTL과 로그 연산에 대한 모델링을 통한 시뮬레이션으로 성능평가를 실시하였다. 성능평가 항목은 로깅 연산으로 인한 메타데이타 로그 오버헤드와 풀 스트라이프 패리티 연산 및 부분 패리티 관리로 인한 패리티 연산 오버헤드에 대해서 성능평가를 실시하였다. 시뮬레이션 평가를 위해서 사용된 낸드 플래시 메모리 모델은 NAND conf. 1과 NAND Conf. 2이며, NAND conf. 1의 경우, 4KB의 페이지 크기, 256KB의 블록 크기, 저장 용량은 2GB를 사용하였으며, NAND conf. 2의 경우 8KB의 페이지 크기, 1MB의 블록 크기, 8GB의 저장 용량을 모델링 하여 사용하였다.

먼저, RAID-5 구성에 따른, 스트라이프 로깅 방식에 대한 오버헤드를 평가하였다. 이 오버헤드에는 StripeMapLog 블록에서 스트라이프 그룹 변경 사항에 대한 페이지 업데이트로 인한 StripeMapLogInfo 페이지 쓰기 오버헤드 연산을 포함한 전체 메타 데이터의 쓰기 오버헤드이다. 스트라이프 로깅 오버헤드는 몇 개의 StripeMapLog 블록을 할당하느냐에 따라서 전체 스냅샷 연산 오버헤드 주기가 달라지기 때문에 StripeMapLog 블록의 개수 설정이 중요한 부분 중의 하나이다. 본 평가에서는 StripeMagLog 블록의 수를 1에서 4로 변경하면서 request 크기가 4~32KB인 임의 쓰기(Ramdom Write) 요청에 대해서 발생하는 플래시 저장 시스템의 쓰기 연산의 횟수에 대해서 각 블록 개수당 스트라이프 로깅의 오버헤드를 평가하여 그림 4에 도식화하였다.

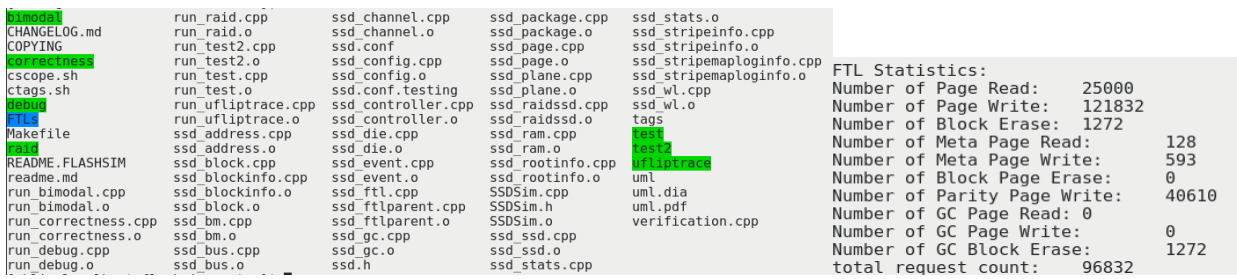

그림 3. 시뮬레이션 소스코드와 시뮬레이션 데이터 예시
Fig. 3. Simulation Source code and example of simulation result

그래프에서, x-축은 두 개의 낸드 플래시 메모리 저장장치 설정에 대한 스트라이프 그룹의 블록 수를 나타내고, y-축은 정규화된 쓰기 양을 나타낸다. 그림에서 나타난 바와 같이, 전반적인 로깅 방식의 오버헤드는 스트라이프 그룹의 크기 및 스트라이프 맵 로그 블록의 개수에 따라 변화하며, 대략 1%~10% 정도임을 알 수 있다. 로그 블록 수 관점에서 보면, 로그 블록 수가 많을수록 스냅샷의 플러시 주기가 길어지기 때문에 전체적인 스트라이프 로깅 오버헤드가 줄어드는 것을 알 수 있으며, 스트라이프 그룹 수가 많아지면, 그만큼 스트라이프 메타데이터 양이 상대적으로 줄어들기 때문에 스트라이프 로깅 오버헤드가 줄어들게 된다. 물론 이 로그 오버헤드는 낸드 플래시 저장시스템의 저장 용량에 따라서 편차가 보일 수 있다.

그렇지만 대체로 스트라이프 그룹의 개수와 로그 블록의 개수를 어느 정도 적정한 선에서 결정할 경우 적정한 오버헤드를 가질 수 있도록 조절할 수 있다. 만약, 스트라이프 그룹을 16으로 정하고, 4개의 로그 블록을 사용하면 저장 용량이 수백 GB가 되더라도 RAID-5 스트라이프 구성에서 로깅 오버헤드가 5% 미만이 될 것으로 보인다.

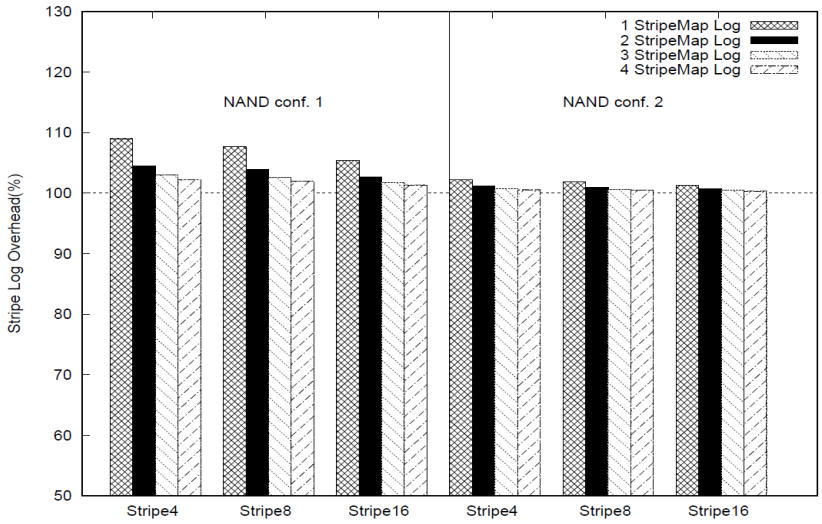

그림 4. 스트라이프 메타 데이터 로그 오버헤드 분석 결과
Fig. 4. Experimental results for stripe metadata log overhead

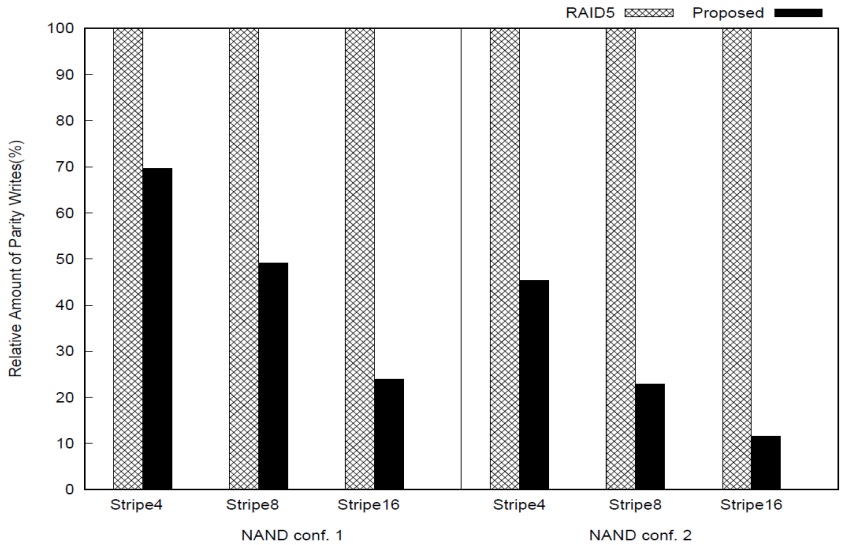

그림 5. RAID-5와 비교한 패리티 및 로그 오버헤드 분석
Fig. 5. Parity write overhead in comparison with RAID-5

다음으로, 기본 RAID-5의 패리티 연산과 비교해서 풀 스트라이프 패리티 쓰기만 발생할 경우의 패리티 I/O의 오버헤드에 대해서 비교 평가하여 보았다. 이 평가 분석에서는, FTL의 메타 데이터 관리는 같은 영향을 미치기 때문에 앞서 언급한 메타데이터 스냅샷 쓰기 연산 오버헤드는 제거하고 패리티 연산에 대한 오버헤드를 평가 분석하였다.

그림 5는 풀 RAID-5와 본 논문에서 설계한 패리티 연산 방식에 대한 패리티 연산 오버헤드를 상대적으로 비교하여 도식화한 것이다. 원래 RAID-5의 경우에는 모든 사용자의 쓰기 연산에 대해서 업데이트되는 부분 패리티를 읽고 쓰는 연산이 발생하기 때문에 패리티 연산 오버헤드가 많이 존재한다. 본 논문 방식의 경우에는 부분 패리티에 대해서는 메인 메모리에서 업데이트 연산이 발생하고, 사용자 데이터에 대한 풀 스트라이프 쓰기가 완료된 이후에, 해당 스트라이프에 대한 패리티가 함께 저장소에 쓰이기 때문에 부분 패리티의 쓰기 연산은 발생하지 않는다. 다만, 추가로 발생하는 연산은 메타데이터에 대한 로깅 연산이 추가로 발생하는 부분이 있으나, 이 양은 부분 패리티 쓰기 연산에 비하면 적은 양이라 할 수 있다. 그래프의 분석 결과에서 알 수 있듯이 본 논문의 방식이 일반적인 RAID-5 패리티 쓰기 연산 양보다 상대적으로 많이 줄어든다.

Ⅴ. 결론 및 향후 과제

RAID는 플래시 스토리지 장치에 안정성을 높이기 위해 사용되었지만, 고유 패리티 업데이트 오버헤드가 있다. 특히 부분 스트라이프 쓰기 패리티는 플래시 기반 RAID 기술의 중요한 문제 중 하나이다. 본 논문에서는 RAID-5 기법을 적용한 플래시 스토리지 장치에서 런타임 부분 패리티 연산의 오버헤드를 줄이기 위해 패리티 스트라이프 로그 아키텍처 및 연산방법을 제안했다. 런타임 동안, 풀 스트라이프 쓰기가 완료될 때까지 부분 패리티가 버퍼 메모리에서 업데이트되며, 전체 스트라이프 그룹이 데이터 쓰기가 완료될 때까지 패리티 로그 구조가 메모리에 유지된다. 패리티 쓰기 연산은 풀 스트라이프가 완성되면 해당 스트라이프의 데이터와 함께 플래시 메모리에 저장된다. 이 패리티 로그를 사용하면 정전으로 인한 전력 손실 발생 시 잃어버린 부분 패리티를 복구할 수 있다.

References

[1] N. Mielke et al., "Bit error rate in NAND Flash memories", 2008 IEEE International Reliability Physics Symposium, Phoenix, AZ, USA, pp. 9-19, Apr. 2008.

[2] Y. Cai, O. Mutlu, E. F. Haratsch and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation", 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, pp. 123-130, Oct. 2013.

[3] Micron, "Enabling Software BCH ECC on a Linux platform", Technical Note, TN-29-71, 2012.

[4] K. Zhao et al., "LDPC-in-SSD: Making advanced Error Correction CodesWork Effectively in Solid State Drives", 11th USENIX FAST, pp. 243-256, 2013.

[5] P. M. Chen and Edward K. Lee, "Striping in a RAID Level 5 Disk Array", ACM SIGMETRICS Performance Evaluation Review, Vol. 23, No. 1, pp. 136-145, 1995.

[6] Y. Lee, S. Jung, and Y. H. Song, "FRA: A flash-aware redundancy array of flash storage devices", 7th IEEE/ACM Int. Conf. Hardware/Softw. Codes. Syst. Synthesis, pp. 163-172, 2009.

[7] S. Im and D. Shin, "Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD", IEEE Transactions on Computers, Vol. 60, No. 1, pp. 80-92, Jan. 2011.

[8] J. Kim, E. Lee, J. Choi, D. Lee, and S. H. Noh, "Chip-Level RAID with Flexible Stripe Size and Parity Placement for Enhanced SSD Reliability", IEEE Transactions on Computers, Vol. 65, No. 4, pp. 1116-1130, Apr. 2016.

[9] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a

Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings", Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, Washington, DC, USA, pp. 229-240, Jan. 2009.

[10] Youngjae Kim, Brendan Tauras, Aayush Gupta, Bhuvan Urgaonkar, "FlashSim: A Simulator for NAND Flash-Based Solid-State Drives", International Conference on Advances in System Simulation, Porto, Portugal, pp. 125-131, Sep. 2009.

[11] Matias Bjorling, "Extended FlashSim", GitHub, https://github.com/MatiasBjorling/flashsim.

저자소개

임 승 호 (Seung-Ho Lim)

2001년 2월 : 한국과학기술원 전기 및 전자 전공(공학사)
2003년 2월 : 한국과학기술원 전기 및 전자 전공(공학석사)
2008년 2월 : 한국과학기술원 전기 및 전자 전공(공학박사)
2008년 3월 ~ 2010년 2월 : 삼성전자 메모리 사업부 책임 연구원
2010년 3월 ~ 현재 : 한국외국어대학교 컴퓨터.전자시스템 공학부 교수
관심분야 : 운영체제, 파일 시스템, 임베디드 시스템, 비휘발성 메모리 시스템, 빅데이터 처리