DDR-SSD를 위한 소프트웨어 RAID의 효과적인 작은 쓰기 처리 기법

Efficient Small Write Method for DDR-SSD based Software RAID

길기정*, 곽동호*, 곽윤식*, 정승국**, 황정연**, 최길성***, 송석일*

Ki-Jeong Khil*, Dong-Ho Kwak*, Yun-Sik Kwak*, Seung-Kook Cheong**, Jung-Yeon Hwang**, Kil-Seong Choi*** and Seo-Kil Song*

요 약

이 논문에서는 DDR-SSD 기반의 소프트웨어 RAID에서 작은 쓰기 (Small Write) 요청에 대한 RAID5의 RMW (Read Modify Write) 성능 향상을 위한 차-로깅 (Differential Logging) 기법을 제안한다. 엔터프라이즈 응용에서 빈번하게 발생하는 작은 쓰기 요청은 RAID 5에서 주요한 성능 저하의 요인이다. RAID 5 에서는 패리티 블록의 일관성을 유지하기 위해 변경이 발생하면 패리티 블록에 대한 변경을 같이 수행해야 한다. 작은 쓰기가 발생하면 기존 데이터에 대한 변경 뿐 아니라, 패리티 블록을 다시 계산하기 위한 추가 입출력연산 및 패리티 계산이 병행되어야 하며 이를 RMW 연산이라 한다. 기존의 하드 디스크 기반의 소프트웨어 RAID 에서는 이러한 작은 쓰기로 인한 성능저하 문제를 해결하기 위해 다양한 방법을 제안하였다. 이 논문에서는 하드 디스크와 전혀 다른 특성을 보이는 DDR-SSD를 고려하여 RAID5의 작은 쓰기 성능을 향상 시키는 차-로깅 기법 기반의 RAID5를 제안한다. 제안하는 기법은 시뮬레이션을 통해서 리눅스 기반의 MD와 비교하여 성능의 우수함을 보였다.

Abstract

In this paper, we propose differential-logging method to improve the performance of RMW(Read Modify Write) operations of DDR-SSD based software RAID. Small writes that are frequently occurred in enterprise applications are main factor to degrade the performance of RAID5. Once a block is updated in RAID 5, the parity block of the block must be updated to maintain consistency of parity. Therefore, to process a small write request, we need to read its parity block stored in disk, read old data, perform XOR operation, and write updated data and parity block. Several methods for hard disk based software RAID are proposed to solve the small write problems in RAID 5. In this paper, we propose a differential-logging method which carefully considers the DDR-SSD to solve the small write problem in RAID 5. We show that our proposed method out performs the existing software RAID in LINUX through simulations.

Key words : DDR-SSD, Software RAID, RMW

I. 서 론

기존의 전통적인 디스크 시스템은 대량의 데이터를 신뢰성 있게 저장하고 빠른 입출력 속도를 제공하

* 충주대학교 컴퓨터공학과(Computer Engineering Department, Chungju National University)
** 한국전자통신연구원
*** 동아방송예술대학 미디어기술학부 방송통신과
· 제1저자 (First Author) : 길기정
· 투고일자 : 2010년 10월 14일
· 심사(수정)일자 : 2010년 10월 14일 (수정일자 : 2010년 10월 23일)
· 게재일자 : 2010년 10월 30일

는데 있어 한계가 있다. 이러한 요구사항을 만족하기 위해서 다수의 디스크를 이용하는 RAID (Redundant Array of Independent Disk)가 현실적인 대안이 되고 있다. RAID 는 성능, 신뢰성, 가격 측면에서 다양한 구현 방법과 구성방법이 있다. 하드웨어 RAID 는 RAID 기능을 하드웨어로 구현한 것으로, 시스템 보드에 내장되거나, PCI 버스 슬롯에 삽입되는 애드온 보드 형태를 가진다. 또한, RAID 기능 고속화를 위한 가속 회로를 포함하여 대체적으로 고가이다. 반면에 소프트웨어 RAID는 RAID 기능을 운영체제의 커널 또는 장치 드라이버에서 구현한 것으로 대부분의 상용 운영체제에서 제공하고 있다. 소프트웨어 RAID는 별도의 전용하드웨어를 사용하지 않아서 비교적 저가로 신뢰성과 성능을 얻을 수 있다.

RAID 구성 방법은 논리 블록들을 물리디스크에 저장하는 방식에 따라서, RAID 0, 1, 2, 3, 4, 5, 6, 0+1, 1+0, 5+0 등 다양한 수준의 RAID를 구성할 수 있다[1]. 이중 2, 3, 4 는 잘 사용되지 않고 있으며 주로 수준 0, 1, 5 의 RAID나 이들간의 혼합 기법이 상용제품에 사용되고 있다. RAID 0 은 논리 블록들을 다수의 물리디스크들에 분산하여 저장(striping)하는 것으로 병렬성을 높여 다른 수준의 RAID 에 비해 높은 입출력 속도를 얻을 수 있다. 하지만, 하나의 디스크라도 고장이 발생하면 데이터를 잃게 되어 신뢰성은 다른 수준의 RAID에 비해 낮다. RAID 1은 논리 블록들을 두 개의 물리디스크에 중복해서 저장하여 스토리지 시스템의 신뢰성을 높인다. 하지만, 같은 데이터를 두 디스크에 중복해서 써야 하므로 입출력 성능은 그만큼 감소된다.

RAID5는 미리 정해진 블록 단위로 데이터를 분할하고 분할된 블록들 간의 패리티를 계산한다. 패리티는 별도의 패리티블록에 저장되고 패리티 블록은 RAID를 구성하는 각 저장장치에 분산하여 작업부하를 분산한다. RAID 5는 디스크 고장이 발생하더라도 패리티 블록을 이용하여 원래 데이터를 복구해 낼수 있다. 읽기 요청에 대해서도 높은 병렬성을 이용하여 비교적 높은 성능을 제공한다. 하지만 쓰기나 변경의 경우 데이터 블록외에도 패리티 블록을 써야 하거나, 기존 패리티 블록을 변경해야 하기 때문에 많은 부담이 있다.

앞에서 언급한 소프트웨어 RAID는 스토리지 시스템을 구성하는 여러 저장 장치를 소프트웨어적으로 하나로 묶어 하나의 저장장치처럼 보이게 한다. 소프트웨어 RAID는 별도의 추가적인 하드웨어를 가지고 있지 않기 때문에 RAID 5의 경우 패리티로 인한 부담이 하드웨어 RAID에 비해 전체적인 성능에 더 영향을 미치게 된다. 특히, 엔터프라이즈 응용에서 발생하는 빈번한 작은 쓰기 (Small Write)는 RMW (Read Modify Write) 를 발생시켜 저장시스템의 성능을 더욱더 저하시키게 된다. SPC (Storage Performance Council)에서 제공하는 I/O 트레이스[2]들 중 Financial1 과 Financial2 분석해서 표 1과 같은 결과를 얻을 수 있었다. 표에서 보는 것처럼 전체 I/O 요청의 평균 크기는 약 3.38K, 2.39K 정도 되었고, 그 중 2K 보다 작은 쓰기 요청의 비율이 전체의 44%, 68%에 달했다.

표 1. SPC I/O 트레이스 분석 결과
Table 1. Analysis results of SPC I/O traces

	Financial1	Financial2
건수	5,334,987	3,669,195
쓰기 비율	77%	18%
평균 요청 크기	3.38K	2.39K
2K 보다 작은 요청비율	44%	68%

RAID 5 는 디스크 고장이 발생했을 때 데이터를 복구할 수 있기 위해서는 패리티 블록을 항상 일관되게 유지해야한다. 이를 위해 스트라이프를 구성하는 일부 블록이 변경이 되면 다시 패리티를 계산해서 패리티 블록에 기록해야 한다. 이때, 스트라이프 전체가 변경되거나 새롭게 쓰이는 경우에는 기존에 계산되었던 패리티 블록을 다시 읽어서 패리티를 다시 계산할 필요가 없다. 하지만, 작은 규모로 변경이 되는 경우에는 변경되기 전 블록과 패리티 블록을 저장장치로부터 읽어낸 후에 새로운 블록, 변경되기 전 블록, 패리티 블록간의 XOR (eXclusive OR) 연산을 수행한 후에 새로운 블록과 패리티 블록을 디스크에 기록하는 과정을 수행한다.

정리하면, 전체 스트라이프를 구성하는 블록 중 하나가 변경이 되는 경우에도 총 4회의 IO, 2회의 블

록간 XOR 연산을 수행해야 한다. 이와 같은 변경연산을 RMW (Read Modify Write)라고 한다. 표 1과 같이 엔터프라이즈 응용에서는 작은 쓰기의 비율이 매우 높으므로, 빈번한 RMW를 발생시키게 되고 이로 인해 전체적인 스토리지 시스템의 성능이 크게 저하된다.

기존에 이 문제를 해결하기 위해 다수의 기술이 개발되었다. 기존의 관련 기술을 분류해보면 작은 쓰기를 고려하는 캐쉬 관리 기술들[3,4]과 패리티 로깅 기술[5,6]로 나누어 볼 수 있다. 이들은 하드디스크를 기반으로 하는 RAID에서 작은 쓰기의 성능을 개선하기 위한 방법을 제안하고 있거나, 하드웨어 RAID에서 별도의 하드웨어 장치를 필요로 한다. 이 논문에서 제안하는 새로운 RAID 5 의 RMW 성능 개선 방법은 DDR-SSD 소프트웨어 RAID에 기반한 것이다.

DDR-SSD는 DRAM을 기반으로한 SSD로 기존의 하드디스크에 비해서 입출력 성능이 최대 100 배 이상 높다. 그리고, 순차I/O와 랜덤I/O 간의 성능차이가 거의 없다[7]. 이 논문에서는 DDR-SSD의 이와 같은 특성을 충분히 고려하여 RAID 5를 위한 차-로깅(Differential Logging)을 이용한 RMW 기법을 제안한다. 제안 하는 RMW 기법은 기존의 패리티 로깅 기법의 변형으로 I/O 비용 뿐 아니라 패리티 로깅 비용을 줄이고 시스템 고장으로 인한 데이터의 손실도 막을 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해서 기술하고, 3장에서는 제안하는 차-로깅 기반의 RMW 기법을 설명한다. 4장에서는 성능평가 결과를 기술하며 5장에서 결론을 맺는다.

Ⅱ. 관련연구

이미 언급한 것처럼, 기존의 관련 기술을 분류해보면 작은 쓰기를 고려하는 캐쉬 관리 기술들과 패리티 로깅 기술로 나누어 볼 수 있다. 이 논문에서 제안하는 RMW는 패리티 로깅 기법을 기반으로 한다. 이에 따라, 기존에 제안된 패리티 로깅 기술을 요약하여 기술한다.

[5]에서 제안하는 패리티 로깅 기술은 작은 쓰기로 인한 IO 횟수를 줄이기 위해 로깅 기법을 이용한다. 작은 쓰기가 발생하면 패리티 로그를 생성하고 다수의 패리티 로그들을 모아서 몇 개의 큰 쓰기로 변환하여 IO 횟수를 줄인다. 패리티 로깅 기술은 로그를 기록하기 위한 추가 디스크를 필요로 한다. 즉, RAID5 가 N+1 (N개의 데이터 디스크, 1개의 패리티 디스크) 개의 디스크를 필요로 한다면 패리티 로깅 기법은 N+2 (N개의 데이터 디스크, 1개의 패리티 디스크, 1개의 로그 디스크) 개의 디스크를 필요로 한다.

패리티 로깅 기법에서는 작은 쓰기가 발생하면 변경되기 전 블록을 읽어서 새로운 블록과 XOR 연산을 수행하여 패리티 블록을 생성한다. 이 블록을 패리티 로그라고 한다. 이 패리티 로그는 일단 주기억장치 상의 로그 버퍼에 기록되고, 버퍼에 로그가 일정량 이상 차게 되면 로그 디스크에 쓰인다. 로그 디스크에 저장되는 패리티 로그는 RAID5의 패리티 블록과 마찬가지로 여러 디스크에 분산될 수 있다.

[6] 에서는 패리티 로그 압축 기법을 제안하였다. 패리티 로그 압축 기법은 패리티 블록(로그)를 특정 압축기술을 이용해 압축하여 로그의 크기를 줄이는 기술이다. 패리티 로깅기법의 로그가 저장되는 버퍼는 안정성 문제로 고가의 비휘발성 메모리에 위치하는데, 고가이므로 충분한 크기의 메모리를 사용하기 어렵다. 패리티 로그 압축 기술은 로그를 압축하여 보다 효과적으로 비휘발성 메모리를 사용하고 동시에 IO 횟수를 더 줄인다.

Ⅲ. 제안하는 차-로깅 기반의 RMW

이 논문에서는 DDR-SSD의 특성을 고려하여 소프트웨어 RAID에서 RAID 5에 대한 작은 쓰기 연산 처리 성능을 높이는 차-로깅기법(Differential Logging)을 제안한다. 제안하는 차-로깅기법은 저장장치 고장으로 인한 블록 손실뿐 아니라 시스템 고장으로 인한 최신 데이터의 손실 문제도 해결한다.

이 논문에서 제안하는 차-로깅 기법은 변경전 블록과 변경된 블록을 XOR 연산하여 패리티 로그를

만들 때, 블록 과 블록에 대한 XOR을 수행하지 않고, 블록의 변경된 섹터들에 대해서만 XOR 연산을 수행하여 로그를 만들어 내는 것을 말한다. 이때 생성되는 로그를 이 논문에서는 차-로그라 한다. [6]에서 제안하는 패리티 로그와 차-로그가 다른 점은 먼저 로그의 크기이다. 패리티 로그의 크기는 블록크기와 같지만, 차-로그의 크기는 변경된 섹터의 개수와 같다. 그리고, 로그를 생산해 내기 위한 XOR 연산의 비용이 줄어든다.

디스크 기반의 소프트웨어 RAID에서는 XOR 연산에 대한 비용이 디스크 I/O 비용에 비해 무시할 정도로 작았다. 하지만, DDR-SSD 기반의 소프트웨어 RAID 에서는 DDR-SSD의 I/O 비용이 XOR 연산에 근접하므로 무시할 수 없다. 따라서, DDR-SSD 기반의 소프트웨어 RAID 에서는 I/O 횟수를 줄이는 것과 더불어 XOR 연산의 비용을 줄이는 것이 성능 향상의 중요한 요인이 된다.

차-로그는 블록 중 변경된 섹터들에 대해서만 만들어지므로, 차-로그를 설명하는 헤더 정보가 필요하다. 헤더 정보를 포함한 완전한 차-로그의 형식은 그림 1과 같다. 이때 종류는 일반 차-로그 와 더미(DUMMY) 로그를 구분하기 위해서 이다. 더미 로그에 대해서는 뒤에서 다시 언급한다.

차-로그 (디스크번호, 블록번호, 시작섹터, 끝섹터, 종류, 로그)

그림 1. 차-로그 형식
Fig. 1. Differential log

그림 2는 차-로깅 기법을 포함한 소프트웨어 RAID의 구성을 보여준다. 스트라이프 관리자는 파일 시스템으로부터 전달되는 I/O 요청을 받아서 RAID를 구성하는 저장장치 숫자와 RAID 수준에 따라서 스트라이프를 생성한다. 이 논문에서 제안하는 차-로깅 기법은 작은 쓰기의 성능을 개선하기 위한 것에 초점이 맞추어져 있으므로, 이후의 설명에서 I/O는 작은 쓰기 요청이라고 가정한다. 작은 쓰기 요청에 의해 생성된 스트라이프는 차-로그 관리자에게 전달되며, 차-로그 관리자는 I/O 관리자로부터 변경된 블록의 구 블록을 읽어 온다. 변경된 블록의 실제 변경된 섹터들에 대해서 XOR 연산을 수행하여 차-로그를 생성하면, 이 로그를 먼저 해당 스트라이프의 패리티 블록이 저장된 디스크의 로그 영역에 기록한다. 또한, 이 차-로그를 차-로그 버퍼 관리자에 전달하여 차-로그 버퍼에 기록하여 관리하게 한다.

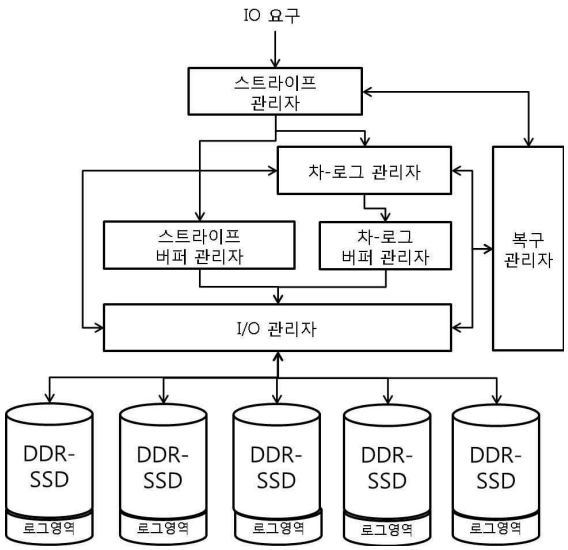

그림 2. 차-로깅 기법을 이용한 소프트웨어 RAID의 구조
Fig. 2. Proposed Software RAID Architecture

스트라이프 관리자는 생성한 스트라이프를 스트라이프 버퍼 관리자에게 전달하고, 스트라이프 버퍼 관리자는 이를 주기억장치의 스트라이프 버퍼에 저장하여 관리한다. 스트라이프 버퍼 관리자는 LRU (Least Recently Used) 정책을 사용하여 스트라이프 버퍼를 관리하며, 특정 디스크 블록을 플러시 할 때에는 해당 블록과 관련된 차-로그를 차-로그 버퍼 관리자를 통해 읽어오고, 저장장치로부터 패리티블록을 읽어서 새로운 패리티 블록을 생성한다. 이어서, 생성한 패리티 블록을 저장장치에 기록하고, 스트라이프를 플러시 한다. 동시에 패리티 블록이 저장되는 디스크의 차-로그 영역에 해당 패리티블록이 플러시 되는 디스크 블록에 대해서 변경되었음을 알려주는 더미 로그를 기록한다. 더미 로그는 해당 블록에 대한 로그가 더 이상 필요가 없음을 알리는 역할을 한다. 디스크 고장 등으로 복구를 해야 할 때 더미 로그가 쓰이기 이전의 로그들은 고려대상이 되지 않게 된다.

차-로그 버퍼 관리자는 차-로그 관리자가 전달해 준 차-로그가 어떤 블록의 변경으로 생성되었는지 확인한다. 그리고, 차-로그 버퍼에서 해당 블록과 관련하여 기존에 생성된 다른 차-로그가 있는지 확인한다. 기존에 생성된 차-로그가 존재하는 경우에는 새로운 차-로그와 겹치는 섹터를 찾아서 XOR 연산을 수행하고, 겹치지 않는 섹터들과 합쳐서 하나의 차-로그를 생성한다. 합쳐진 차-로그는 다시 차-로그 버퍼에 기록하며, 이때 기존 차-로그는 삭제가 가능하다.

복구 관리자는 시스템의 고장이 발생해서 스트라이프 버퍼를 잃게 되었을 때와 RAID를 구성하는 저장장치의 고장이 발생할 때 동작한다. 시스템 고장으로 재시동 될 때는 스트라이프 버퍼를 잃게 되므로, 일부 스트라이프는 최신 데이터가 아닌 구 데이터를 가지게 된다. 이때 차-로그 버퍼도 같이 잃게 되므로, 유실된 최신의 데이터를 복원하기 위해서는 각 저장장치의 로그 영역을 주기억장치로 읽어온다.

각 저장장치 별로 로그 영역의 차-로그를 뒤에서 앞으로 (최신의 차-로그부터) 순차적으로 읽어 가면서, 각 디스크 블록에 해당하는 차-로그를 병합한다. 이때 만일 특정 디스크 블록에 대한 DUMMY 로그가 발견이 되면 그 디스크 블록에 대해서는 더 이상 차-로그를 읽어서 병합하는 과정을 수행하지 않는다. 모든 차-로그에 대해서 위의 작업을 수행한 후 병합된 차-로그와 저장장치에 저장된 디스크 블록 (구 디스크 블록)을 읽어서 XOR 연산을 수행하면 최신의 디스크 블록을 구할 수 있다.

저장장치 고장으로 인해 손실된 디스크 블록들을 복원할 때는 RAID5 가 수행하는 일반적인 복구 방법을 통해서 복구가 수행된다. 즉, 손실되지 않은 디스크들로부터 데이터 블록들과 패리티 블록을 읽고 이들을 모두 XOR 연산을 수행하여 손실된 블록을 복원한다. 본 발명의 차-로깅 기법에서 저장장치에 있는 하나의 스트라이프의 데이터 블록들에 대한 패리티 블록은 항상 일관된 패리티를 가지고 있다.

변경된 데이터 블록에 대한 패리티의 업데이트 시점이 해당 블록의 플러시 시점 (변경된 블록이 저장장치에 반영되는 시점) 이므로 저장장치 측면에서 데이터 블록의 업데이트는 패리티 블록의 업데이트와 같은 시점에 발생한다. 또한, 각 저장장치의 로그영역이 가득 차서 로그를 플러시 해야 하는 상황에서도 패리티 블록의 업데이트와 변경된 블록의 플러시를 모두 수행하므로 저장장치의 데이터 블록들과 패리티 블록은 항상 일관되다. 따라서, 손실된 디스크의 블록을 복구하기 위해서는 디스크에 저장된 스트라이프 (패리티 블록 포함)를 읽어서 XOR 연산을 수행하면 된다.

그림 3은 본 발명의 차-로깅 기법이 작은 쓰기 요청을 어떤 과정으로 처리하는지를 보여준다. 그림에서 볼 수 있는 것처럼, 작은 쓰기 요청은 스트라이프로 만들어진 후 스트라이프 버퍼에 저장된다. 동시에 차-로그 관리자에 의해 차-로그가 만들어지고 이것이 디스크의 로그 영역과 차-로그 버퍼에 저장되어서 관리된다.

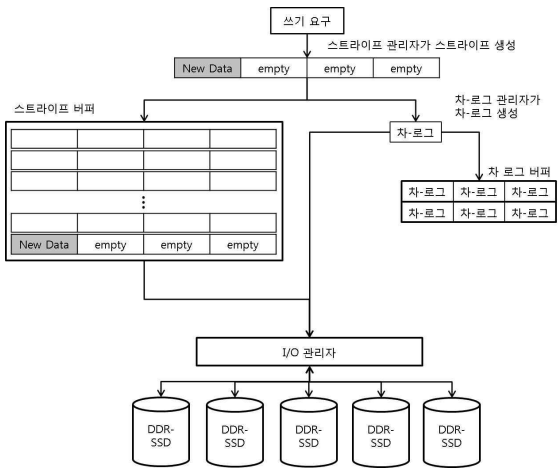

그림 3. 작은 쓰기 요청 처리 과정
Fig. 3. Process of small write request

그림 4는 이 논문의 차-로깅 기법에 대한 이해를 돕기 위한 예를 보여주고 있다. 그림에서 DDR-SSD의 개수는 3개이고 각 디스크의 가장 아래쪽 블록이 차-로그를 저장하는 로그 영역이다. 설명의 편의상 차-로그 버퍼는 생략하였다. 그림에서 보는 것처럼, 스트라이프 버퍼에는 블록 D00를 변경한 D00' 이 저장되어 있다. D00를 D00'으로 변경하기 위해서는 그림의 오른쪽 첫 번째 박스에 있는 것처럼 먼저 D00를 디스크에서 읽고, 차-로그 (Diff-log00)를 구한다음 차-로그를 바로 Disk2에 기록한다.

그림 4. 차-로깅 기법의 예
Fig. 4. Example of differential logging method

Disk2는 D00가 포함된 스트라이프의 패리티 블록이 저장되는 곳이다. D00' 가 어떠한 이유로 플러시 될 때는 제일 먼저 로그 버퍼의 차-로그를 읽고 해당 블록으로부터 생성된 로그 레코드들을 병합하여 하나의 차-로그 (Diff-log00sum)를 생성한다. 그리고, 패리티 블록 P0를 읽고 P0와 Diff-log00sum 간의 XOR 연산을 수행한 후에 결과를 디스크에 기록하고 D00'를 디스크에 기록한다.

시스템 고장이 발생하여 재시동을 하는 경우에는 시스템 고장으로 유실된 데이터를 복구하기 위해서는 먼저 각 디스크의 로그 영역의 차-로그를 읽고 블록 별로 발생한 로그를 정렬하여 병합한다. 그리고, 병합한 데이터를 디스크에 기록하여 복구를 한다. 마지막으로 디스크가 고장이 날 경우에 대한 예이다. 그림 오른쪽의 마지막 박스에 보면 Disk1이 고장난 상태에서 D01을 읽으려 할 때는 스트라이프를 구성하는 D00를 읽고 패리티블록 P0를 읽어서 XOR 연산을 수행한다.

Ⅳ. 성능평가

이 논문에서는 제안하는 차-로깅 기반의 RMW의 성능을 측정하기 위해서 시뮬레이터를 리눅스 플랫폼 상에서 C 언어로 구현하였다. 비교 대상은 현재 리눅스에서 기본으로 제공되고 있는 소프트웨어 RAID로 하였으며 비교를 위해 소프트웨어 RAID에 대한 시뮬레이터를 역시 리눅스 플랫폼 상에서 C 언어로 구현하였다. 실험에 사용한 데이터는 SPC에서 제공하는 Financial1이다. 기타 시뮬레이션에 사용된 파라미터는 표 2와 같다.

표 2. 시뮬레이션 파라미터
Table 2. Simulation parameters

Data Set	SPC Financial 1 Trace
Number of Disks	8
Page Size	4K ~ 32K
Read Time (us)	4K(3.7), 8K(4.8), 16K(7.1)
Write Time (us)	4K(13.8), 8K(15.8), 16K(19.6)
Log Region	1M ~ 128M/Disk
Stripe Buffer	(128 ~ 1024) X Stripe Size
XOR Time (us)	4K XOR 4K -> 11us

표에서 DDR-SSD의 읽기 시간 및 쓰기 시간은 [8]을 참조하였다. 그림 6은 페이지의 크기를 4k ~ 32k로 변경하고, 로그의 크기가 1M ~ 128M 일때의 IOPS를 측정한 결과이다. 그림에서 보는 것처럼 기존의 리눅스에서 제공하는 소프트웨어 RAID에 비해 제안하는 기법의 성능이 월등히 앞서고 있음을 볼 수 있다. 로그 영역의 경우 크기가 클수록 높은 성능을 보였지만, 큰 차이는 없었다.

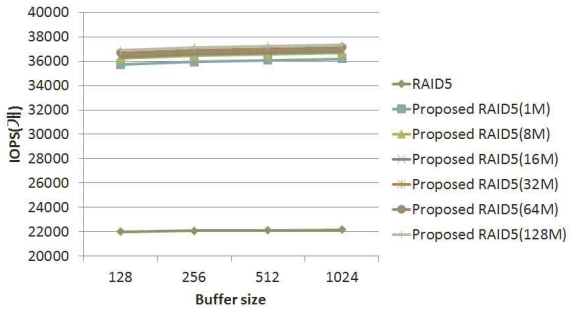

그림 5. 버퍼크기변경시 IOPS
Fig. 5. IOPS with varying buffer size

그림 7은 페이지 크기를 변경할 때에 각 방법의 IOPS를 측정한 결과를 보여준다. 그림에서 보는 것처럼, 기존 리눅스가 제공하는 소프트웨어 RAID에 비해서 높은 성능을 보임을 알 수 있다. 특히, 페이지 크기가 8k일 때 부터는 성능향상이 뚜렷하게 없음을 알수 있으며, 페이지 크기가 커 질수록 로그 영역이 작으면 성능이 저하되는 것을 볼수 있다.

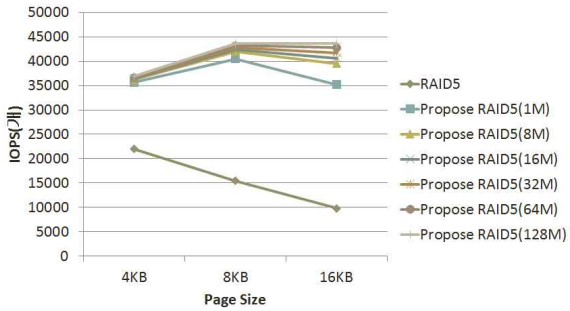

그림 6. 페이지 크기 변경시 IOPS
Fig. 6. IOPS with varying pase size

V. 결 론

이 논문에서는 DDR-SSD 기반의 소프트웨어 RAID의 작은 쓰기 성능 향상을 위해 차-로깅 기법의 RMW를 제안하였다. 제안하는 차-로깅 기법은 디스크 고장으로 인한 데이터의 복구 뿐 아니라, 시스템 고장으로 인한 데이터의 손실도 복구가 가능하다. 또한, DDR-SSD가 I/O 성능이 하드 디스크에 비해 월등히 높아서 패리티 계산 비용이 성능에 높은 영향을 미치는 것을 막기 위해서 패리티 연산을 최소화하는 차-로깅 기법을 제안하게 되었다. 성능평가 결과에서 볼 수 있듯이 디스크 별로 약 1 ~ 8M 의 추가 공간만 있으면, 기존 리눅스에서 제공하는 소프트웨어 RAID에 비해 높은 성능의 RMW를 제공함을 볼수 있었다. 향후 연구에서는 리눅스를 기반으로 제안하는 차-로깅 기법을 설계하고 구현하여 실제 응용에서 어떠한 성능을 내는지 확인한다.

감사의 글

이 논문은 한국전자통신연구원 정보통신개발사업(NGS 시스템 기술개발)의 위탁연구과제로 수행한 연구결과임

참 고 문 헌

[1] P. Chen, E. Lee, G. Gibson, R. Katz and D. Patterson, “RAID : High -Performance, Reliable Secondary Storage,“ *ACM Computing Surveys, vol. 26, no. 2, pp. 145-185,* June 1994.

[2] http://www.storageperformance.org/

[3] 김종훈, 노삼혁, 원유헌, "소프트웨어 RAID에서 효율적 작은 쓰기를 위한 캐쉬 관리 기법", *한국정보과학회 1996년도 가을 학술발표논문집, 제23권, 제2호(B), pp. 857-860*, 1996. 10.

[4] 김종훈, 노삼혁, 원유헌, "소프트웨어 RAID 파일 시스템에서 작은 쓰기와 참조 횟수를 고려한 캐쉬 교체 정책," *한국정보과학회 1997년도 봄 학술발표논문집, 제24권, 제1호(A), pp. 123-126*, 1997. 4.

[5] D. Stodolsky, G. Gibson and M. Holland, "Parity logging overcoming the small write problem in redundant disk arrays" *ACM SIGARCH, vol. 21, no. 2, pp. 64-75,* May 1993.

[6] 김근혜, 장은정, 최황규, "디스크 배열에서 작은쓰기 문제 해결을 위한 압축 패리티 로깅 기법," *한국정보과학회 1998년도 가을 학술발표논문집, 제25권, 제2호(III), pp. 12-14,* 1998. 10.

[7] 황정연, 정승국, "DDR 기반의 SSD 스토리지 시스템 기술 동향," *정보통신산업진흥원 pp. 28-41,* 2009. 10.

길 기 정 (丁道令)

2007년 3월 ~ 현재 : 충주대학교 컴퓨터공학과 학사과정
관심분야 : 데이터베이스, 스토리지 시스템, 모바일 프로그램 등

곽 동 호(丁道令)

2010년 2월 : 충주대학교 컴퓨터공학과 (공학사)
2010년 3월 ~ 현재 : 충주대학교 컴퓨터공학과 석사과정
관심분야: 스토리지 시스템, 모바일 프로그래밍 등

곽 윤 식 (丁道令)

1984년 2월: 청주대학교 전자공학과 (공학사)
1986년 2월: 경희대학교 전자공학과 (공학석사)
1994년 2월: 경희대학교 전자공학과 (박사)
1995년 1월 ~ 1996년 1월 : Texas Tech University 파견 교수
1991년 5월 ~ 현재 : 충주대학교 전기전자공학부 교수
관심분야 : 영상처리, 마이크로프로세서, 유비쿼터스 컴퓨팅

정 승 국 (丁道令)

2004년 2월 : 한남대학교 전자정보통신공학과(박사)
1985년 ~ 현재 : 한국전자통신연구원 (책임연구원)
관심분야 : Grid Computing, Utility Computing, Solid State Disk, Storage& Server Virtualization

황 정 연 (丁道令)

1993년 2월 : 중앙대학교 통계학과 (이학석사)
2006년 2월 : 충남대학교 통계학과 (이학박사)
1993년 6월 ~ 현재 : 한국 전자통신 연구원 책임연구원
관심분야 : 데이터마이닝, EA/ITA, DDR 기반의 SSD 시스템, SRM S/W 개발

최 길 성 (丁道令)

1988년 2월 : 대전산업대학교 전자계산학과(이학사)
1992년 2월 : 수원대학교 전자계산학과(이학석사)
1999년 2월 : 충북대학교 정보통신공학과(공학박사)
1999년 ~ 현재 : 동아방송예술대학 방송통신과 교수
관심분야:데이터베이스, 색인구조, 스토리지 시스템 등

송 석 일 (丁道令)

1998년 2월 : 충북대학교 정보통신공학과(공학사)
2000년 2월 : 충북대학교 정보통신공학과(공학석사)
2003년 2월 : 충북대학교 정보통신공학과(공학박사)
2003년 7월 ~ 현재 : 충주대학교 컴퓨터공학과 부교수
관심분야: 데이터베이스, 센서 네트워크, 색인구조 등