공학석사학위논문

플래시메모리 저장장치를 위한 리셋 기반의 읽기 성능 최적화 기법

2018 년 2 월

서울대학교 대학원

컴퓨터공학부

이재훈

플래시메모리 저장장치를 위한 리셋 기반의 읽기 성능 최적화 기법

지도교수 김지홍

이 논문을 공학석사 학위논문으로 제출함

2017 년 10 월

서울대학교 대학원

컴퓨터공학부

이 재 훈

이재훈의 공학석사 학위논문을 인준함

2017 년 12 월

위 원 장 : 하 순 회 (인)

부위원장 : 김 지 홍 (인)

위 원 : 유 승 주 (인)

요약

본 연구에서는 기존의 읽기 응답시간의 최적화 기법들을 소개하고 기존 기법의 한계를 해결하기 위한 리셋 명령을 제안한다. 또한 기존 기법들과 새롭게 제안하는 리셋 명령을 통합 적용하여 실험을 진행하고 각 기법의 장점과 단점을 비교 분석하였다.

리셋 명령은 읽기 요청이 쓰기/소거가 진행중인 칩을 선점하는데 까지 걸리는 시간이 일시정지에 비하여 빠르지만, 수명을 단축시키는 문제가 존재한다. 따라서 중요한 읽기에 대해서만 선택적으로 적용할 것을 제안하였다.

본 논문에서 제안하는 우선순위 기반 선택적 리셋 기법은 저장장치 내의 가비지 컬렉션 동안에 유효 페이지 읽기와 사용자 읽기가 서로 다른 우선순위를 갖고 실행되는 것에 착안하여, 저장장치 밖의 호스트 시스템에서 유발되는 가비지 컬렉션 읽기와 사용자 읽기에도 서로 다른 우선순위를 부여하여 선택적으로 리셋을 적용하였다.

평가 결과, 쓰기/소거 충돌이 발생하는 모든 사용자 읽기 요청에 리셋을 수행하는 경우와 우선순위 기반으로 선택적 리셋을 적용하는 경우, 평균 읽기 응답시간, 99.99^{th} 읽기 꼬리응답시간, 수명이 각각 9%, 6.9%, 20% 향상되는 것을 확인하였다. 모든 충돌에 대하여 리셋을 수행하는

경우에 비하여 선택적 리셋 기법이 더 좋아지는 이유는 기존에 해결할 수 없었던 읽기 요청 간의 큐 지연시간이 해결되기 때문이다. 또한 저장장치의 수명이 리셋을 전혀 사용하지 않는 경우와 비교하여 단 2.8% 정도의 수명 단축이 발생함을 관찰하였다.

주요어: 사용자 읽기 요청, 읽기 응답시간 감소, 선택적 리셋, 우선순위.

학번: 2016-21226

목차

그림목차

표목차

제 1 장 서론

1.1 연구 배경

낸드 플래시 메모리는 저전력, 내구성, 빠른 접근 속도, 휴대성, 높은 신뢰도, 가벼운 무게 등의 특징으로 스마트폰, 디지털카메라 등 모바일 기기의 저장장치 시장에서 널리 사용되고 있다. 이러한 장점에 더하여 설계 공정의 미세화 및 최근 개발에 성공한 삼차원 낸드플래시 기술을 통하여 단위 면적 당 가격을 대폭으로 줄이면서 서버 시장까지 점유해 나가고 있다. 또한 최근에는 초대용량 데이터의 빠른 처리를 위해서 모든 저장장치를 SSD(Solid-State Drives)로 사용하는 올플래시 스토리지(All-Flash Storage) 형태의 고성능 서버 시스템도 등장하게 되었다.

현재 서버 시장에서 적극적으로 낸드 플래시 메모리를 선택하는 이유는 최근의 기하급수적으로 늘고 있는 방대한 양의 데이터를 빠르게 처리함과 동시에 사용자의 요청에 대한 응답 시간을 특정 수준으로 보장하기 위해서이다.

사용자의 쓰기 요청은 서버에게 정보 기록을 요청하는 것이며, 이는 서버에 존재하는 다양한 계층의 캐시에 먼저 기록되고 추후에 후경 프로세스가 처리하는 등 수많은 최적화가 되어있다. 하지만 읽기 요청은 서버의 각 계층 별 캐시에 존재하지 않는다면 캐시 접근 실패가 발생하여 저장장치에 직접 접근하여 정보를 읽어내야 한다. 만약 이때의 저장장치가 다양한 사용자 읽기와 후경 프로세스가 처리하는 쓰기 요청으로 바쁜 상황이라면, 가장 최근에 저장장치에 들어온 사용자 읽기의 응답시간은 더욱 지연될 것이다. 저장장치의 여러가지 사정에 의해서 사용자가 요청한 데이터를 서버가 적시에 제공하지 못할 경우 사용자 경험(User Experience)에 나쁜 영향을 주게 될 것이다.

읽기 요청이 저장장치까지 보내졌을 때 발생할 수 있는 지연의 원인은 다음과 같다. 먼저 읽기 요청은 저장장치의 버퍼에서 요청하는 정보가 캐싱되어 있는지 확인한다. 캐시 접근 실패가 발생하는 경우 메모리 칩 별로 유지하는 큐에 삽입되며 이전에 삽입된 모든 요청의 처리가 완료될 때까지 기다리게 된다. 이 경우 읽기 요청에 대한 큐잉 지연이 발생한다. 또한 큐의 가장 앞부분까지 도착했더라도 메모리 칩에서 읽기, 쓰기, 소거 연산이 이미

진행중이라면 연산이 완료될 때까지 기다려야 하기 때문에 추가적인 지연이 발생한다. 이외에도 낸드 플래시의 저장 공간 확보를 위해 쓰레기 수거 작업이 전경에서 발생하게 되면, 저장장치는 가비지 컬렉션(Garbage Collection; GC) 작업이 완료될 때까지 사용자 요청을 받지 못하기 때문에 가비지 컬렉션이 완료될 때까지 요청이 지연된다.

게다가 소셜 네트워크와 같은 대규모의 서로 연결된 그래프를 처리해야하는 서버에서는 사용자가 보낸 하나의 요청을 제공하기 위해 수십 개의 질의 요청이 추가로 발생한다[1-3]. 하나의 읽기 요청을 위해서 추가적인 읽기 요청이 발생한다면. 사용자 읽기 요청에 대한 응답시간은 더욱 지연될 것이다.

위와 같은 이유로 서비스를 제공하는 서버 업계에서는, 대규모 데이터를 처리해야 하는 서버들은 고속 플래시 스토리지 성능을 최대로 활용하고 신속한 데이터 처리가 가능한 NoSQL 기반 데이터베이스 등의 솔루션을 선택하게 되었다. 또한 이러한 고성능 서버의 빠른 응답시간에 대한 요구를 만족시켜주기 위해서 인텔과 마이크론은 응답속도가 기존의 SSD 보다 1000 배가 빠른 PRAM 기반의 3D XPoint 를 출시하였으며, 삼성에서도 이를

대응하기 위해서 순차 읽기 성능이 PRAM 보다 빠르고 응답시간은 동일한 Z-SSD 등을 출시하였다.

학계에서도 읽기 응답시간을 줄이기 위한 다양한 노력을 하였다. 첫 번째 예로, 가비지 컬렉션 과정에서도 사용자의 요청을 처리할 수 있는 선점 가능한 가비지 컬렉션이 있다[4]. 이 기법은 기존에 하나의 트랜잭션에 형태로 동작하던 가비지 컬렉션이라는 동작을 분할하여 사용자의 요청이 선점 가능하게 하여 사용자 응답시간을 줄였다. 또다른 기법으로는 소프트웨어 기반 무순서 중첩 실행 스케줄링 기법이 있다[5]. 이 기법은 기존의 하드웨어 큐를 소프트웨어로 구현하였고, 플래시 변환 계층의 다양한 정보를 이용하여 중첩 실행 가능성을 최대로 하였다. 또한 QoS(Quality of Service)를 고려하여 사용자 읽기 요청을 가장 먼저 처리하도록 하였다. 쓰기와 소거에 대한 일시정지 /재시작 기법 (Program and Erase Suspension)은 쓰기 혹은 소거 요청이 이미 수행 중에 있는 상황에서 읽기 요청이 수행중인 칩을 선점함으로써 읽기 응답시간 향상시켰다[6]. 이렇게 읽기 요청이 가비지 컬렉션을 선점하고, 큐에서 가장 먼저 처리되며, 칩에서 수행중인 연산도 선점하지만, 칩을 선점하고 읽기 요청을 처리하는데 까지 걸리는 시간은 읽기

수행 시간에 비하여 여전히 길다. 따라서 이러한 추가적인 지연을 단축시키는 노력이 필요하다.

1.2 연구 동기

기존의 쓰기와 소거 일시정지 / 재시작 기법의 경우에 읽기가 수행중인 칩을 선점하기까지의 시간이 읽기 연산의 속도에 비하여 상대적으로 크다. 예를 들어, 쓰기가 진행중에 일시정지를 하기 위해서 걸리는 시간은 150us 가 걸리며, 소거 연산의 경우에는 읽기의 준비를 위한 소거 일시정지까지 2300us 의 지연이 발생한다. 따라서 만일 큐의 깊이가 1 이고 읽기 요청이 현재 진행중인 소거 연산과 충돌이 발생했다고 가정한다면, 일시정지 후 읽기를 처리할 때까지 100us + 2300us = 2400us 가 걸리게 된다. 즉, 쓰기와의 충돌로 인해 발생하는 추가적인 150us 는 읽기 요청이 100us 안에 처리되는 것을 감안하면 2 배 이상이 증가하게 되며, 소거와의 충돌로 인해 발생하는 추가적인 2400us 는 읽기 요청이 100us 안에 처리되는 것에 비하여 24 배 이상이 증가하게 된다.

따라서 본 연구에서는 수행 중에 있는 쓰기와 소거 연산이 읽기 요청과 충돌하였을 때, 큰 오버헤드를 가지는 일시정지 명령 대신에 적은 오버헤드를 가지는 리셋 명령(Reset Command)를 제안한다. 리셋 명령은 일시정지와는 다르게, 읽기 요청이 쓰기 연산이 진행중인 칩을 선점할

때까지 30us 가 필요하며, 소거 연산이 진행중인 칩을 선점할 때까지 200us 가 소요된다.

따라서 읽기 요청이 쓰기 연산과 충돌이 발생하는 경우 리셋 명령을 수행하면, T_{prog_reset}(30us) + T_{Read}(100us) = 130us 으로 일시정지를 수행한, $T_{prog\_suspend}$(150us) + T_{Read}(100us) = 250u 의 경우보다 약 2 배의 빠른 응답을 사용자에게 줄 수 있게 된다. 만약 사용자의 읽기 요청이 진행중인 소거 연산과의 충돌을 하게 되는 경우 리셋 명령을 하게 되면, T_{erase_reset}(200us) + T_{Read}(100us) = 300us 으로 기존에 제안된 일시정지 T_{erase_reset}(2300us) + T_{Read}(100us) = 2400us 보다 8 배나 더 빠른 응답시간을 제공할 수 있다.

하지만 쓰기 연산에 대한 리셋 명령은 해당 페이지를 무효화 시킨다. 만일 SSD 가 MLC 라면 두 페이지에 대한 무효화가 TLC 라면 세 페이지에 대한 무효화가 발생한다. 소거 연산에 대한 리셋 명령 또한 해당 블록에 추가적인 소거 연산을 수행한 정도의 셀 스트레스를 주게 된다. 따라서 빈번한 읽기와 쓰기 / 소거 충돌이 발생하는 상황에서 항상 리셋 명령을 수행하게 된다면 SSD 수명에 심각한 영향을 줄 수 있다. 낸드 플래시 메모리는 쓰기 / 소거의 횟수가 한정되어 있고, 공정의 미세화로 인하여 지속해서 수명이 감소하고 있다. 따라서 사용자의 읽기 요청과 쓰기 혹은 소거 연산의 충돌이 발생하였을 때, 읽기 요청의 중요도에 따라 리셋

명령을 선택적으로 하여 SSD 수명 단축을 최대한 막을 수 있는 기법이 필요하다.

1.3 연구 기여

본 연구의 기여는 크게 두 가지로 정리할 수 있다. 첫 번째로, 기존의 읽기 응답시간의 최적화 기법들을 소개하고 기존 기법들에 대한 한계점을 밝혔다. 그리고 이러한 문제를 해결하기 위한 리셋 명령을 제안하였다. 또한 제안된 리셋 명령과 기존의 기법들을 통합하여 쓰기 연산에는 일시정지를 하고 소거 연산에는 리셋을 하는 등의 다양한 조합으로 실험을 진행하여 각 기법들의 조합에 대한 장점과 부작용에 대하여 분석하였다.

두 번째로, 각 기법들을 통합한 평가 결과를 바탕으로 얻어진 읽기 응답시간과 낸드 플래시 메모리의 수명 간의 트레이드오프 관계를 보이고, 이를 해결할 수 있는 기법을 제안한다. 실험 분석의 결론은 다음과 같다. 빠른 읽기 응답시간을 보장하기 위해서는 일시정지 보다는 리셋을 수행하는 것이 효과적이지만 수명이 감소하는 부작용이 존재한다. 그렇기 때문에 모든 읽기에 대해서 리셋을 수행하기 보다는 읽기 요청의 중요도에 따라서 선택적으로 해야 한다. 이에 따라 본 연구에서는 사용자로부터 전달 된 읽기 요청 중 중요도가 명확하게 구분이 가능한 응용을 소개하고, 해당 응용과 저장장치가 읽기 우선순위를 고려하여

동작하도록 수정하였다. 수정된 상황에서 응용은 읽기 요청에 우선순위를 구분하여 저장장치에 전달하고, 요청을 전달받은 SSD 는 우선순위 정보를 통해서 더 중요한 읽기 요청을 큐에서 먼저 처리한다. 이로써 기존에 해결할 수 없었던 읽기와 읽기 요청 간의 큐에서의 지연을 해결하였다. 동시에 우선순위가 높은 일기 요청만 선택적으로 리셋 명령을 수행하도록 하여 낸드 플래시의 수명 단축을 최소화 하였다.

제 2 장 읽기 응답시간 감소를 위한 기존의 연구

2.1 선점 가능한 가비지 컬렉션

선점 방식 가비지 컬렉션 기법은 SSD 에 유휴 시간이 존재하지 않을 만큼 호스트 시스템에서 요청이 많이 들어와 후경 가비지 컬렉션이 수행될 수 없는 상황의 문제를 해결한다. SSD 에 유휴 시간이 없이 쓰기와 읽기 요청이 혼합하여 들어오게 된다면, 더 이상 쓰기 요청을 받을 공간이 없어지게 되고 이 경우 전경에서 가비지 컬렉션 작업이 수행된다. 전경 가비지 컬렉션 동안에는 사용자의 요청을 받지 않고 모든 작업을 끝마치고 나서 다시 사용자의 요청을 받기 때문에, 사용자 요청에 대한 응답시간이 유효 페이지의 수에 비례하여 길어지게 된다.

선점 가비지 컬렉션 기법은 희생 블록의 유효 페이지가 한 페이지씩 옮겨지는 동안에 사용자의 요청이 발생했다면 다음 유효 페이지가 복사되기 이전에 사용자 요청을 처리한다. 만약 유효 페이지 복사를 위한 요청과 사용자 요청이 동일하다면 두 요청일 병합하여 한 번에 처리하며, 병렬적으로 수행 가능하다면 동시에 처리한다. 다음 그림 1 은 선점 가능한 가비지 컬렉션의 한 예이다.

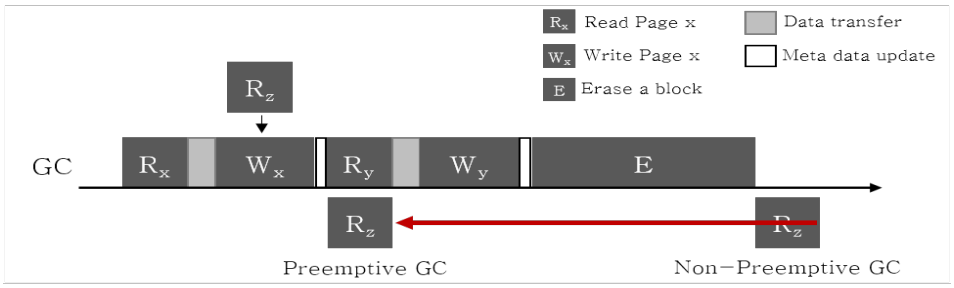

그림 1 선점 가비지 컬렉션 동작

2.2 소프트웨어 기반 무순서 중첩 실행 스케줄링 기법

소프트웨어 기반 무순서 중첩 실행 스케줄링(Software-based Out-of-Order Scheduling; SOS) 방식은 최적의 병렬 실행을 위해서 기존 하드웨어로 무순서 중첩 실행을 하던 방식을 소프트웨어적으로 동작시킨다. SOS 는 플래시 변환 계층과 함께 같은 소프트웨어 계층에서 동작하며 플래시 변환 계층에서 유지하고 있는 정보의 쉬운 접근 및 수정이 가능하다. SOS 는 소프트웨어의 장점을 이용하여 불필요한 하드웨어 계층의 동작들을 줄일 수 있으며 다양한 알고리즘을 사용하여 성능을 향상시킬 수 있다.

SOS 는 데이터 버퍼링을 위한 큐, Dynamic Scheduler, Queue size leveler, Hit manager 로 구성되어 있다. FTL 은 사용자로부터 받은 요청을 낸드 플래시 메모리의 물리적 페이지 사이즈로 분리하여 페이지 단위의 요청을 만든다. 그 후, 각 요청의 논리적 주소에 물리적 주소를 할당을 진행한다. 물리적 페이지 번호가 할당된 각 요청들은 페이지 기반의 데이터 버퍼 역할을 하는 칩 별로 존재하는 큐에 요청에 대한 정보와 데이터를 포함하여 추가된다. SOS 의 Dynamic Scheduler 는 요청들이 독립적으로 각 칩에서 병렬성을 최대한 확보하며 실행되도록 큐에 버퍼링된 요청들을 각 칩에 보낸다. 특정 칩이 유휴 상태가 되었을 때 Dynamic Scheduler 는 해당 칩에 속한 큐에서 존재하는 다음 플래시 연산을 수행한다.

SOS 는 성능 향상을 위해서 기존의 하드웨어적으로 구현된 무순서 중첩 실행 스케줄링 기법 이외에 추가적인 기능을 포함한다. Queue size leveler 는 칩이 바쁜 상황으로 계속 지연되는 요청들을 유휴 칩에

재할당하여 응답시간을 낮춘다. Hit Manager 는 큐의 버퍼에 삽입된 읽기 요청이 해당 버퍼 내부에 동일한 데이터가 존재하는지를 판단하고 해당 읽기가 성공한다면 곧바로 데이터를 읽고 사용자 요청을 완료한다. 동일 쓰기 요청이 다른 칩에 중복된다면 이전 쓰기 요청을 찾아 큐에서 제거한다. 따라서 해당 큐의 전체 대기시간이 짧아지게 된다. Dynamic Scheduler 는 추가적으로 QoS(Quality of Service)를 고려하여 사용자 읽기 요청을 큐의 버퍼에서 가장 먼저 처리하도록 하였다. 다음 그림 2 는 Dynamic Scheduler 가 큐 버퍼에서 각 요청들을 스케줄링 하는 방식을 보여준다.

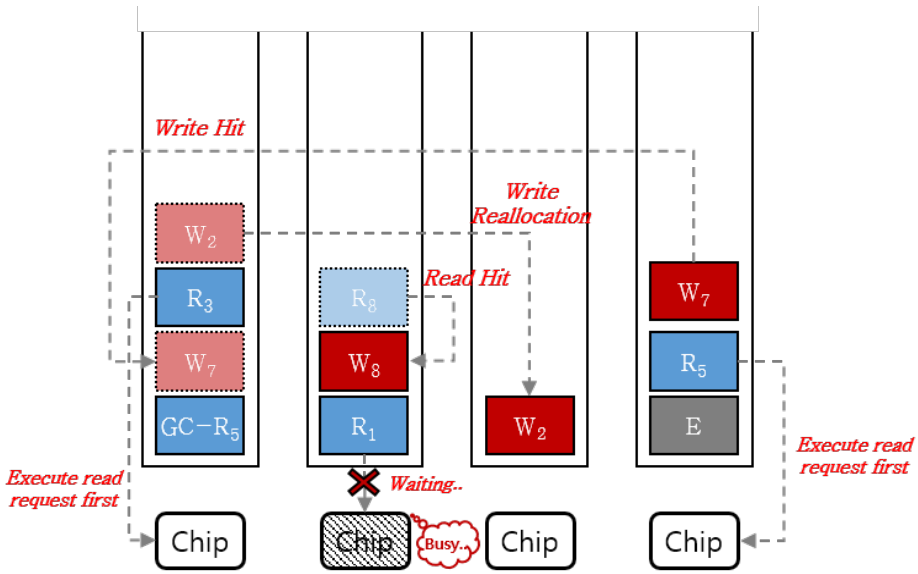

그림 2 소프트웨어 기반 무순서 중첩 실행 스케줄링 방식

SOS 는 선점 가비지 컬렉션을 포함하고 큐의 버퍼에서 사용자 읽기를 우선적으로 처리하여 읽기 응답시간을 추가적으로 높이는 시도를 하였다.

하지만 칩이 유휴 상태인 경우에만 읽기를 처리할 수 있고 쓰기/소거 연산이 칩에서 진행중인 경우에는 상대적으로 수행 시간이 짧은 읽기 요청이 긴 쓰기/소거 작업을 기다려야만 된다. 따라서 SOS 기법에서 읽기 요청을 지연시키는 원인은 수행중인 쓰기와 소거 연산이 읽기와 충돌하는 경우이다.

2.3 쓰기와 소거 일시정지 /재시작 기법

읽기 요청이 가비지 컬렉션을 선점하는 방법, 병렬성 확보, 중복 쓰기 제거, 쓰기 재할당 그리고 QoS 를 고려한 사용자 높은 우선순위를 갖는 읽기 요청 등의 기법 등을 적용하더라도, 쓰기/소거 연산이 읽기와 충돌하는 경우에는 상대적으로 큰 지연시간이 발생했다. 읽기 vs 쓰기 와 읽기 vs 소거 연산이 출동하는 문제를 해결하기 위한 일시 정지 /재시작 기법은 읽기 요청이 쓰기/소거가 진행중인 칩을 선점할 수 있도록 한다. 즉, 실행중인 쓰기 / 소거 연산이 대기하는 읽기 요청을 처리하기 위해 일시정지 하고 읽기 요청이 완료된 후 재시작한다.

다음 그림 3 은 읽기 요청이 수행중인 쓰기 요청을 선점하여 먼저 실행되는 상황을 보여준다.

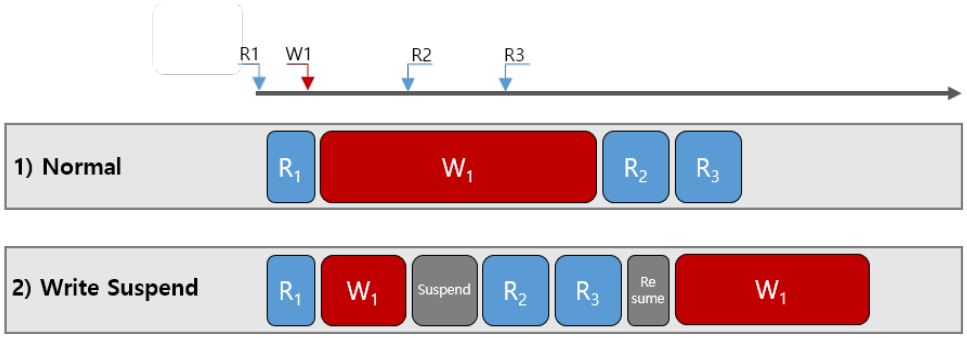

그림 3 쓰기 연산 일시정지

일반적인 경우, R1 이 먼저 도착하여 읽기가 수행되며, 읽기 요청이 진행되는 사이 쓰기 요청 W1 이 들어오게 된다면, W1 을 수행시킨다. W1 의 수행 동안에 들어온 R2, R3 읽기 요청은 W1 요청이 완료된 후 수행된다. 쓰기 일시정지를 적용한 2)번을 보면, W1 이 진행되는 동안에

R2 요청이 들어오면 진행중인 W1 을 일시정지하고 R2 를 처리한다. R2 요청의 처리가 완료된 후, 큐 버퍼에서 또다른 읽기 요청이 있는지 확인하고, 읽기 요청이 없다면 쓰기 재시작, 일기 요청이 있다면 쓰기 재시작을 하지 않고 큐의 읽기 요청을 처리하게 된다.

다음은 그림 4 는 소거 연산의 일시정지를 수행하는 한 예이다.

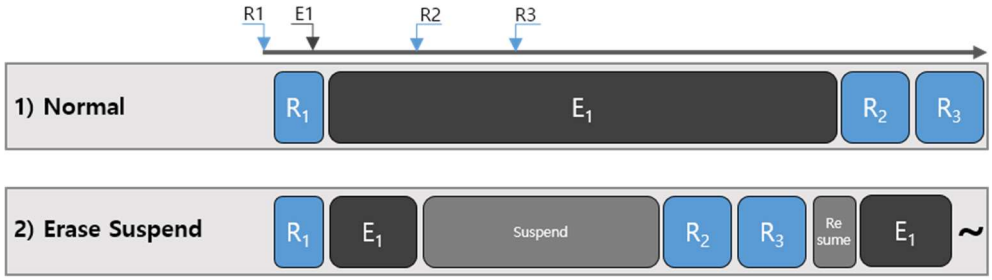

그림 4 소거 연산 일시정지

그림 3 과 동일한 순서로 요청이 큐에 들어오지만, 읽기 요청이 소거가 진행중인 칩을 선점하는데 까지 걸리는 시간이 쓰기 일시정지에 비해서 약 15 배가 길다. 쓰기 일시정지의 시간은 Tprog_suspend = 150us 이며, 소거 일시정지는 Terase_suspend = 2300us 이다. 따라서 읽기 vs 소거 충돌이 발생하면, 최소 TRead + Terase_suspend = 2400us 의 추가 지연시간이 발생한다. 따라서 최악의 경우에 읽기-소거 충돌이 발생하여, 소거 연산이 진행중인 칩을 선점한다 하더라도 읽기 요청의 응답시간은 충돌이 발생하지 않았을 때보다 24 배가 지연됨을 알 수 있다. 즉, 읽기 응답시간 단축을 위해서는 충돌이 발생하였을 때, 읽기 요청이 칩을 선점하는데 까지 걸리는 시간의 단축이 필요하다.

제 3 장 리셋 명령어(Reset Command) 제안

앞선 절에서, 가비지 컬렉션이라는 하나의 큰 트랜잭션 형태로 수행되는 작업을 읽기 요청이 선점하고, SSD 내의 큐에서 읽기 요청을 높은 우선순위로 처리하며, 쓰기/소거가 수행중인 칩을 선점함으로써 읽기 응답시간을 줄일 수 있는 기존의 기법들을 소개하였다. 하지만 쓰기/소거 연산과 읽기 연산의 충돌 시, 일시정지에 오버헤드로 인한 응답시간 지연은 해결해야할 문제로 남아있다. 따라서 본 연구에서는 읽기 요청과 쓰기 혹은 소거 연산이 충돌하였을 때, 더 적은 지연시간을 갖는 리셋 명령을 제안한다. 리셋 명령은 수행중인 연산을 취소시키기 때문에 셀의 상태 확인하고 저장하는 등의 작업이 수행되지 않는다.

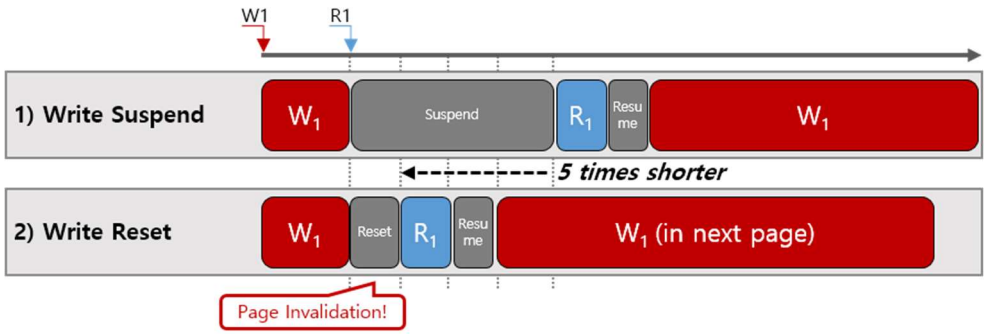

그림 5 쓰기 연산 리셋

다음 그림 5 는 쓰기 일시정지와 쓰기 리셋의 수행과정을 보여준다. 리셋을 하게 되면, 쓰기 연산을 정지시키고 읽기 연산을 수행하는데 까지 걸리는 시간이 5 배 이상 줄어들게 된다. 하지만 쓰기 연산에 대한 리셋 명령은 해당 페이지를 무효화 시킨다. 만일 SSD가 MLC라면 두 페이지에 대한 무효화가 TLC 라면 세 페이지에 대한 무효화가 발생한다.

, 다음 그림 6 은 소거 일시정지와 소거 리셋의 수행과정을 보여준다. 소거 연산에 대한 리셋을 하게 되면, 소거를 정지시키고 읽기 연산을 수행하는데 까지 걸리는 시간이 10 배 이상 줄어들게 된다.

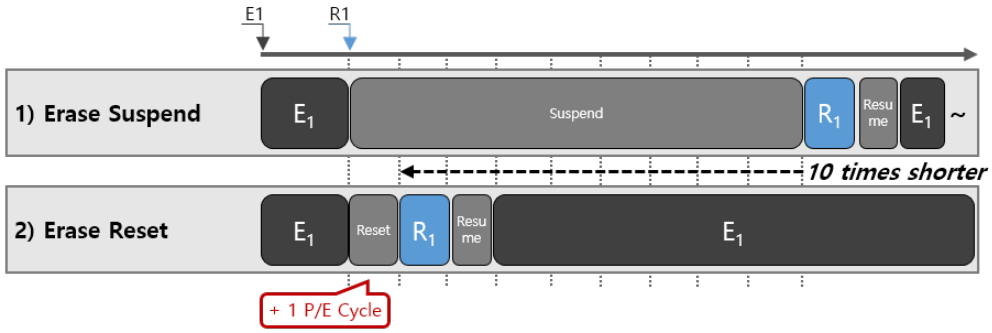

그림 6 소거 연산 리셋

소거 연산에 대한 리셋 명령 또한 해당 블록에 추가적인 소거 연산을 수행하기 때문에 수명을 단축시킨다.

3.1 읽기 응답시간 최적화 기법 통합 평가

본 절에서는 기존의 응답시간을 줄이기 위한 기법들과 제안된 리셋 명령을 통합하여 실험을 진행한다. 또한 진행된 실험을 통해서 각 기법 조합의 장단점을 살펴보고 분석한다.

각 기법들의 조합을 비교하기 위한 기준은 선점 가비지 컬렉션 기능을 포함하고 사용자 읽기 요청에 높은 우선순위를 적용한 SOS 로 선택하였다. 각 실험의 기법 조합은 다음 표 1 과 같다.

번 호	읽기 성능 최적화 기법 조합	최적화 기법 종류
1	SOS	선점 가비지 컬렉션, 중복 쓰기 제거, 쓰기 재할당, 읽기 우선순위 스케줄링
2	SOS + W-S & E-S	SOS, 쓰기 일시정지, 소거 일시정지
3	SOS + W-S & E-R	SOS, 쓰기 일시정지, 소거 리셋
4	SOS + W-R & E-S	SOS, 쓰기 리셋, 소거 일시정지
5	SOS + W-R & E-R	SOS, 쓰기 리셋, 소거 리셋

표 1 읽기 응답시간 최적화 기법 조합

실험의 목적은 기존의 SOS 기법에서 읽기-쓰기 충돌과 읽기-소거 충돌을 해결하기 위한 일시정지 와 리셋 명령을 각각 적용했을 때의 성능 비교 및 각각의 장단점을 분석해 보기 위함이다. 실험은 fio 를 사용하였으며, 일시정지와 리셋 명령을 지원하는 16GB 크기의 SSD

에뮬레이터에서 진행하였다. SSD 는 버스는 2 개, 칩은 각 버스당 4 개로 구성하였다. Fio 는 8GB 크기의 읽기/쓰기 요청을 5:5 의 비율로 수행한다. 또한 Fio 의 큐 깊이를 8 로 설정하였기 때문에, SSD 내부에서 큐에 의한 지연을 최소한으로 하고 읽기-쓰기, 읽기-소거에 대한 충돌 발생에 의한 지연시간을 보다 효과적으로 관찰할 수 있도록 설정하였다.

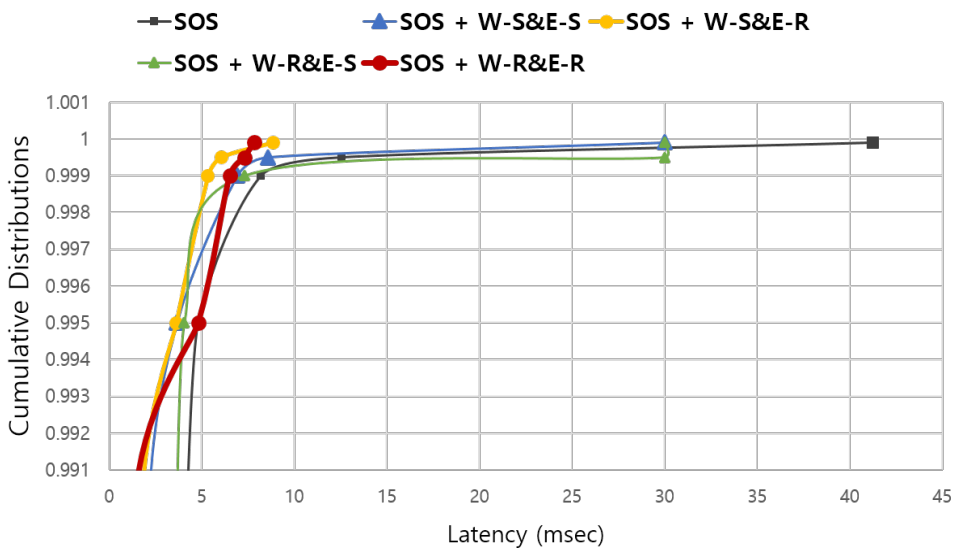

그림 7 최적화 기법 조합의 읽기 응답시간 누적 분포 그래프

먼저 그림 7 의 각 기법의 읽기 응답시간 누적 분포 그래프를 보면, 쓰기 요청에 대한 일시정지와 리셋은 꼬리응답시간에 영향을 거의 주지 않는다. 대신, 소거 연산의 일시정지와 리셋에 영향을 받는 것으로 나타났다. 가장 최악의 경우에는 읽기-소거 충돌이며 이 상황이며, 읽기가 소거를 일시정지 시킬 것인지 리셋을 시킬 것인지에 따라서

꼬리응답시간에 차이를 보였다. 결국 최악의 경우에 읽기 요청의 응답시간을 보장하기 위해서는 소거 연산 일시정지 대신에 리셋이 필요함을 알 수 있다.

다음 그림 8 은 각 기법 조합의 평균 읽기 응답시간을 보여준다. 결과를 보면, SOS + W-S&E-S 에 비해서 SOS + W-R&E-S, SOS + W-R&E-R 모두가 약 7% 이상 평균 응답시간이 줄어든 것을 확인할 수 있다. 이는 읽기의 평균 응답시간은 소거 연산의 일시정지와 리셋의 영향보다는 쓰기 연산의 일시정지와 리셋의 영향을 받는 것을 알 수 있다. 이는 SSD 내부에서 주로 쓰기 연산과 충돌이 발생하는 것을 의미한다. 따라서 평균 읽기 응답시간을 줄이기 위해서는 쓰기에 대해 일시정지를 하기 보다는 리셋을 해야 하는 것을 알 수 있다.

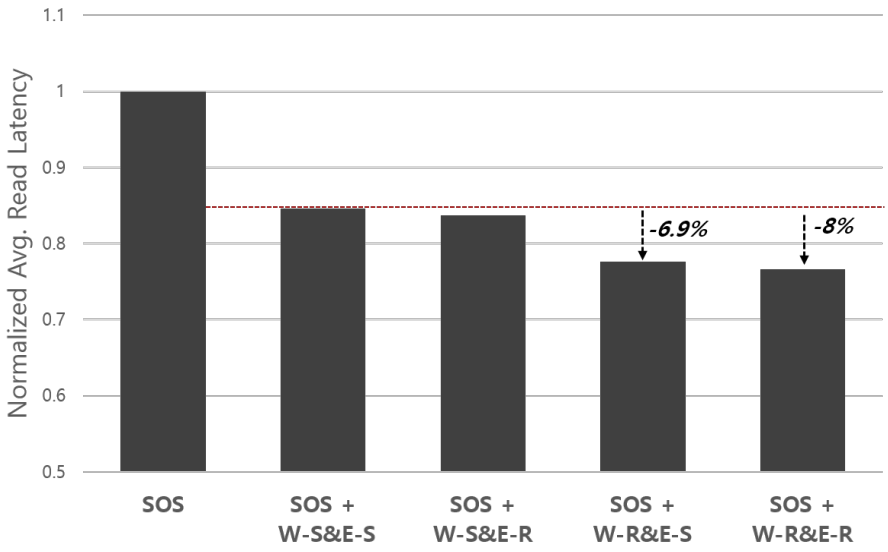

그림 8 최적화 기법 조합의 정규화된 평균 읽기 응답시간

위의 두 결과로 보아, 전체적인 읽기 응답시간을 높이기 위해서는 읽기 요청이 어떤 충돌을 하더라도 일시정지 보다는 리셋을 해야한다.
하지만 리셋을 사용하게 되면 읽기 성능이 좋아지는 것과 반비례하게 낸드 플래시 메모리의 수명이 감소할 것이다.

다음 그림 9 는 쓰기/소거와 읽기 충돌로 인한 리셋의 부작용으로, 페이지 무효화와 희생 블록의 수명 단축 효과를 보기 위해서 fio 실험 동안에 발생한 총 소거 횟수를 측정한 결과이다. 수명의 경우에는 이전 실험 결과와는 조금 다르다. 이전 결과에 의해서 읽기-쓰기 충돌이 대부분이라는 것을 알 수 있었고, 따라서 쓰기 리셋에 대한 수명 단축이 더 심할 것으로 예상됐지만, 읽기-소거에 대한 리셋은 전체 블록에 추가적인 셀 스트레스를 주기 때문에 빈번하 게 발생하는 쓰기 리셋과 거의 동등한 수명 단축을 유발시켰다. 따라서 수명 단축을 최소화 하기 위해서는 쓰기와 소거 모두 리셋을 선택적으로 해야함을 알 수 있다.

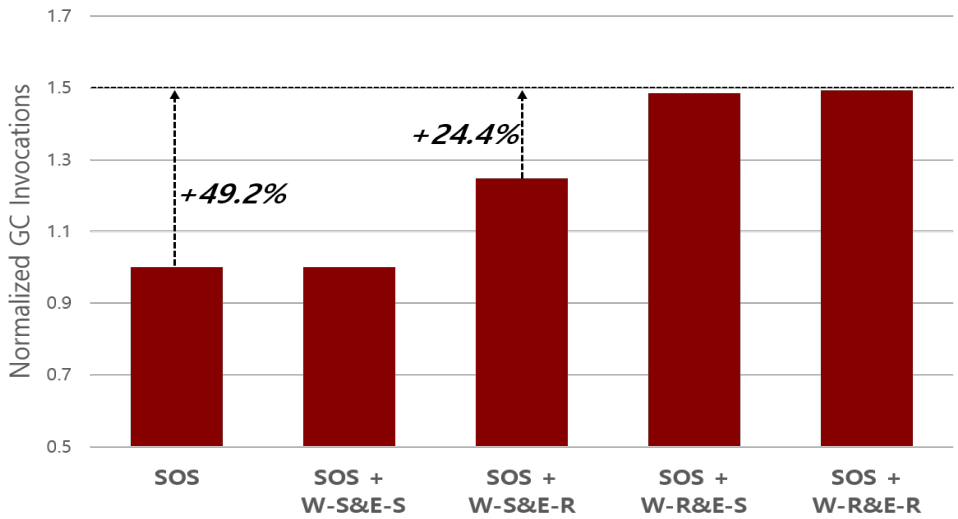

그림 9 최적화 기법 조합의 정규화된 평균 GC 호출 횟수

3.2 통합 평가 결론

평균 읽기 응답시간을 보장하기 위해서는 빈번하게 충돌을 일으키는 읽기-쓰기 충돌에 대해서, 칩을 선점하는 방식으로 일시정지 보다는 리셋을 선택해야 한다. 읽기 꼬리응답시간 지연의 발생원인인 읽기-소거 충돌에 대해서도 빠른 읽기 응답시간을 보장하기 위해서는 리셋을 수행하는 것이 효과적이다. 하지만, 읽기와 쓰기/소거 충돌이 발생할 때마다 리셋을 사용하게 되면, 본 실험의 결과 낸드 플래시 메모리의 수명을 약 50% 단축시켰다. 결국, 모든 읽기에 대해서 리셋을 수행하는 것은 낸드 수명에 치명적인 영향을 주기 때문에 모든 충돌에 사용하기 보다는 읽기 요청의 중요도에 따라서 선택적으로 해야 한다.

제 4 장 우선순위를 고려한 리셋 기법

SOS에서는 사용자의 읽기 요청에 우선순위를 주어 처리한다. 가비지 컬렉션 동안에 유효 페이지 읽기가 발생하여도 사용자의 읽기가 우선순위를 갖게 된다. SSD 저장장치 밖의 호스트 시스템에서도 이러한 가비지 컬렉션 읽기가 존재하지만, 저장장치 수준에서는 알 수 없기 때문에 모든 읽기가 동일하게 높은 우선순위로 처리된다. 따라서 호스트 시스템에서 가비지 컬렉션을 수행하기 위해 발생하는 읽기와 사용자 읽기 요청을 구분하여 SSD에게 알려줄 수 있다면, 사용자가 반응을 기다리는 읽기 요청에 대한 응답시간이 더욱 짧아질 것으로 기대된다.

4.1 Log-Structured Merge-Tree 기반 시스템

LSM-Tree(Log-Structured Merge-tree)는 빠른 쓰기 속도를 위해서 개발된 자료구조로써, LevelDB, RocksDB, Cassandra 등의 Key-Value Store에서 사용된다. LSM-Tree구조는 다음의 그림 10과 같다. LSM-Tree 자료구조는 하나의 메모리 테이블과 다수의 저장장치에 유지되는 테이블로 이루어져 있다. LSM-Tree는 먼저 삽입 요청이 들어오게 되면 메모리 테이블에 일정 크기가 될 때까지 쓰기를 진행한다. 일정 크기가 넘어가게 되면 메모리 테이블에 유지하던 데이터를 저장장치로 내려쓰게(Flush)된다. 저장장치의 테이블은 트리 형식의 특정 레벨에 저장된다. 상위 레벨에 수용할 수 있는 테이

블 개수를 넘어가게 되면 해당 테이블을 모두 읽고, 병합정렬하여 하위 레벨로 테이블을 내려쓰기 하는 압축(Compaction)작업을 수행한다. LSM-Tree 자료구조는 쓰기가 임의의 순서로 요청되어도 저장장치로 내려가는 쓰기는 순차적으로 이루어진다는 특징이 있다. 하지만 요청한 데이터가 메모리 테이블에 없을 경우, 저장장치에 가서 데이터를 찾아야 한다. 또한 원하는 데이터가 어느 레벨에 존재하는지 알 수 없기 때문에 상위 레벨부터 하위 레벨까지 순차적으로 검색을 해야하는 단점을 갖는다. 만일 내려쓰기를 요청하려 하는데 상위 레벨에 테이블이 수용 가능한 개수로 가득 차 있다면, 테이블의 모든 데이터를 읽고 쓰기 때문에 추가적인 읽기와 쓰기가 발생한다.

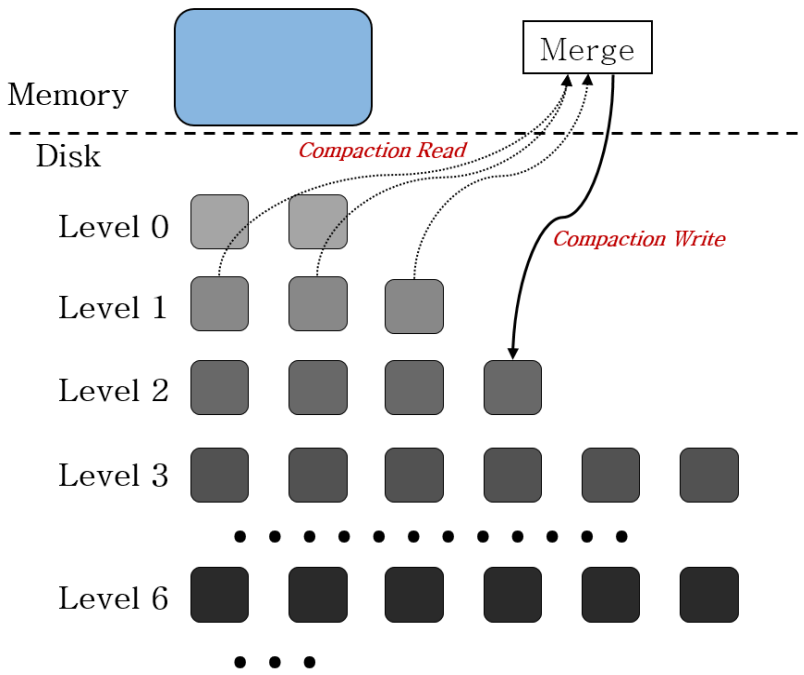

그림 10 Log-Structured Merge-Tree 구조

4.2 PAReset(Priority-Aware Reset) 구조

우선순위를 고려한 리셋 기법은 SOS 구조에서 일시정지, 리셋, 재시작 명령을 낸드 플래시 메모리 컨트롤러가 지원하도록 수정하였다. 또한 기존 SOS의 Dynamic Scheduler가 각 요청에 추가된 우선순위 정보를 기반으로, 선택적 리셋 기능을 포함하여 스케줄링을 수행한다. 호스트 시스템에서 발생하는 읽기 및 쓰기 요청은 추가적인 우선순위 태그를 포함한다. 호스트에서 전달된 요청 태그는 물리적 페이지 단위의 요청으로 변환될 때까지 계속 유지한다. PA Dynamic Scheduler는 각 칩의 큐에서 우선순위가 가장 높은 요청을 처리하며, 같은 우선순위는 FIFO형식으로 처리한다. 각 요청이 가질 수 있는 우선순위는 다음 표 2 와 같다.

할당 가능한 우선순위	요청 타입
1 (리셋)	사용자 읽기
2 (일시정지)	
3 (기본)	
4	사용자 쓰기, GC 읽기, GC 쓰기, GC 소거
5	

표 2 PAReset에서 각 요청에 할당 가능한 우선순위

호스트에서는 사용자 읽기 응답시간 중요도에 따라서 선택적으로 1에서 3까지의 우선순위를 부여할 수 있으며, 아무런 태그에 아무런 표기를 하지 않는다면 기본값인 3이 할당된다. 사용자 읽기는 항상 사용자 쓰기에 우선한다. 사용자 쓰기의 경우 기본적으로 가비지 컬렉션 읽기, 쓰기, 소거에 우선하지만, 여유 공간이 없을 경우 가비지

컬렉션 요청의 우선순위가 사용자 쓰기에 우선한다. 다음 그림 11 은 PAReset의 구조를 보여준다.

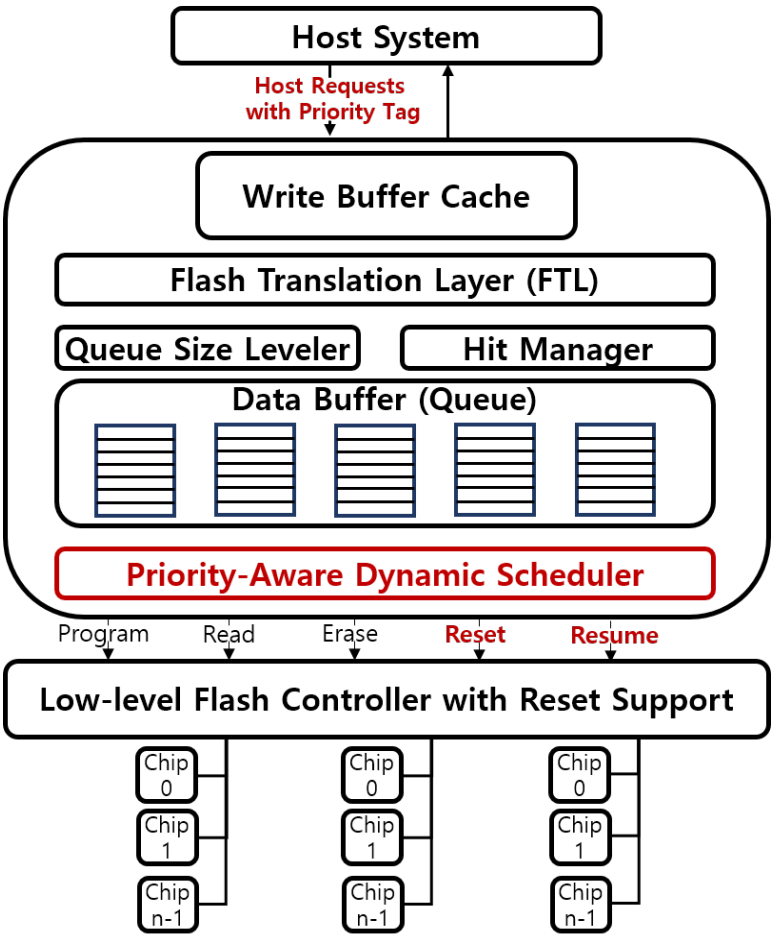

그림 11 Priority-Aware Reset 구조

PAReset은 호스트로부터 사용자의 읽기 요청이 들어오면 먼저 쓰기 버퍼 캐시에서 요청하는 데이터가 있는지 확인한다. 캐시에서 읽기 실패가 발생하면 물리적 페이지 단위로 요청이 나누어져 큐에 삽입된다. PA Dynamic Scheduler는 각 칩에서 우선순위가 가장 높은 읽기 요청을 먼저 처리한다. 이때의 읽기 요청 우선순위가 3이라면

칩에서 수행중인 연산이 완료될 때까지 기다린다. 우선순위가 2라면 진행중인 쓰기, 소거 연산을 일시정지한다. 우선순위가 가장 높은 1이라면 현재의 쓰기 혹은 소거 연산을 리셋 하여 읽기를 먼저 수행시킨다. 동시에 Queue Size Leveler는 지연되는 쓰기 요청을 유휴 칩으로 재할당하며, Hit Manager는 중복된 쓰기 요청을 제거하는 기능을 수행하여 전체적인 응답시간을 높인다.

제 5 장 실험 결과 및 분석

5.1 실험 환경

실험은 LSM-Tree 기반의 NoSQL 데이터베이스인 RocksDB 를 기반으로 진행하였다. RocksDB 는 Facebook 에서 고속 플래시 스토리지 성능을 최대로 활용하고 신속한 데이터 처리가 가능하도록 개발한 Key-Value 데이터베이스 시스템이다[8]. 현재 Facebook 의 그래프 엔진인 Dragon 에 사용되고 있다[9]. 그 이외에도 LinkedIn, Yahoo, Airbnb, Netflix 등 수많은 서버에서 사용되고 있기 때문에, 본 실험의 결과가 대용량 데이터를 처리하는 서버 상황을 보다 잘 반영할 수 있도록 하였다. 벤치마크는 RocksDB 를 내장하고 있는 db_bench 를 사용하였으며, 실험은 서버용 워크로드를 사용하였다. 실험에서 사용한 워크로드는 여러 개의 임의 읽기를 수행하는 스트림과 하나의 임의 쓰기를 진행하는 스트림이 10 억개의 키 범위 내에서 각각 2 백만 번의 읽기, 쓰기를 유발시킨다. 요청의 우선순위는 RocksDB 의 Compaction 스레드에서 발생하는 읽기를 낮은 우선순위, 그 이외의 읽기는 높은 우선순위로 구분하였다.

실험은 1) PAReset, 2) SOS, 3) SOS + Suspend, 4) SOS + Reset, 네 가지 기법을 비교하였으며, PAReset 기능을 포함한 램디스크 기반의 SSD 에뮬레이터에서 수행되도록 진행하였다.

5.2 실험 결과

다음 그림 12 는 PAReset 을 포함한 각 기법의 평균 읽기 응답시간, 읽기 IOPS(Input / Output Operations Per Second), 낸드 플래시의 수명인 P/E Cycle(Program / Erase Cycle)을 측정한 결과이다. SOS 의 경우, 가비지 컬렉션 동안에도 읽기 요청을 받고 처리하며, 큐에서 의존성이 없는 읽기 요청을 가장 우선적으로 처리하는 경우이다. SOS+W-S&E-R 은 진행중인 쓰기 / 소거와 사용자 읽기 요청 간에 충돌이 발생하였을 때, 진행중인 연산을 일시정지하고 사용자 읽기를 우선적으로 처리하는 방식이다. 다음으로 SOS+W-R&E-R 은 모든 읽기 요청이 충돌한 쓰기 요청과 소거 요청에 대하여 현재 진행중인 칩을 선점하여 연산을 리셋 시키고 우선 처리되는 경우이다. 마지막 PAReset 의 경우, RocksDB 가 발생시키는 Compaction 읽기와 일반 사용자 읽기를 구분하여 큐에서 더 높은 우선순위로 처리된다. 쓰기/소거 연산과 읽기가 충돌이 발생하였을 때는 사용자 읽기인 경우에만 선택적으로 리셋을 적용한 경우이다.

그림 12 의 실험 결과를 보면, 수행중인 쓰기, 소거 연산을 선점하지 못하는 SOS 에 비하여 나머지 세 기법의 읽기 성능이 높은 것을 알 수 있다. 또한 쓰기/소거 연산에 대한 일시정지를 적용하면 기존 SOS 보다 읽기 성능이 높아지는 것을 알 수 있다. 만약 모든 읽기 요청과 쓰기/소거 연산이 충돌할 때, 리셋을 적용하면 일시정지 보다 약 18.7%의 평균 응답시간의 성능향상을 보인다. PAReset 기법의 경우, 저장장치의 큐에 존재하는 읽기 요청 간의 우선순위 차이로 인해 사용자

읽기 요청이 Compaction 읽기보다 먼저 선택된다. 따라서 큐에서의 지연시간이 줄어들게 되어 약 9%의 추가적인 읽기 응답시간의 향상을 보였다.

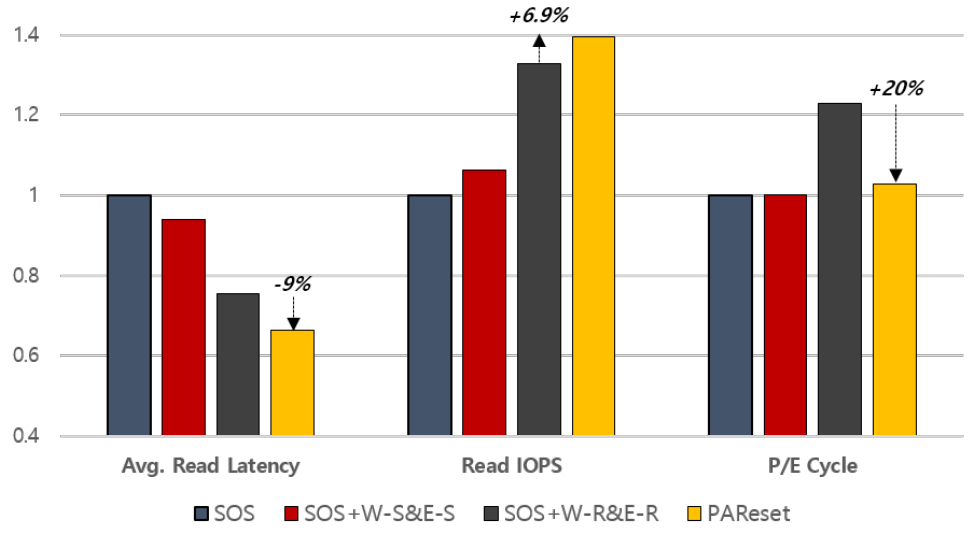

그림 12 PAReset 의 읽기 성능 및 수명 단축 실험결과

결과를 보면, 수행중인 쓰기, 소거 연산을 선점하지 못하는 SOS 에 비하여 나머지 세 기법의 읽기 성능이 높은 것을 알 수 있다. 또한 쓰기/소거 연산에 대한 일시정지를 적용하면 기존 SOS 보다 읽기 성능이 높아지는 것을 알 수 있다. 만약 모든 읽기 요청과 쓰기/소거 연산이 충돌할 때, 리셋을 적용하면 일시정지 보다 약 18.7%의 평균 응답시간의 성능향상을 보인다. PAReset 기법의 경우, 저장장치의 큐에 존재하는 읽기 요청 간의 우선순위 차이로 인해 사용자 읽기 요청이 Compaction

읽기보다 먼저 선택된다. 따라서 큐에서의 지연시간이 줄어들게 되어 약 9%의 추가적인 읽기 응답시간의 향상을 보였다.

낸드 플래시 메모리의 수명을 보면, 사용자의 읽기 요청만 선택적으로 리셋을 적용했기 때문에 SOS+W-R&E-R 보다 리셋 요청을 더 제한적으로 수행되어, PAReset 이 SOS+W-R&E-R 에 비하여 약 20%의 수명 향상을 보였다. 또한 리셋을 아예 적용하지 않은 경우에 비하여 약 2.8% 정도의 수명 단축이 측정되었다.

다음 그림 13 은 각 기법의 읽기 응답시간을 누적 분포 그래프로 나타낸 것이다. 결과를 보면, 기존에는 해결 할 수 없었던 사용자 읽기 요청에 우선순위가 부여됨에 따라 기존 기법보다 읽기 꼬리응답시간이 SOS+RR 에 비해 약 12.5%가 줄어든 것을 확인할 수 있다.

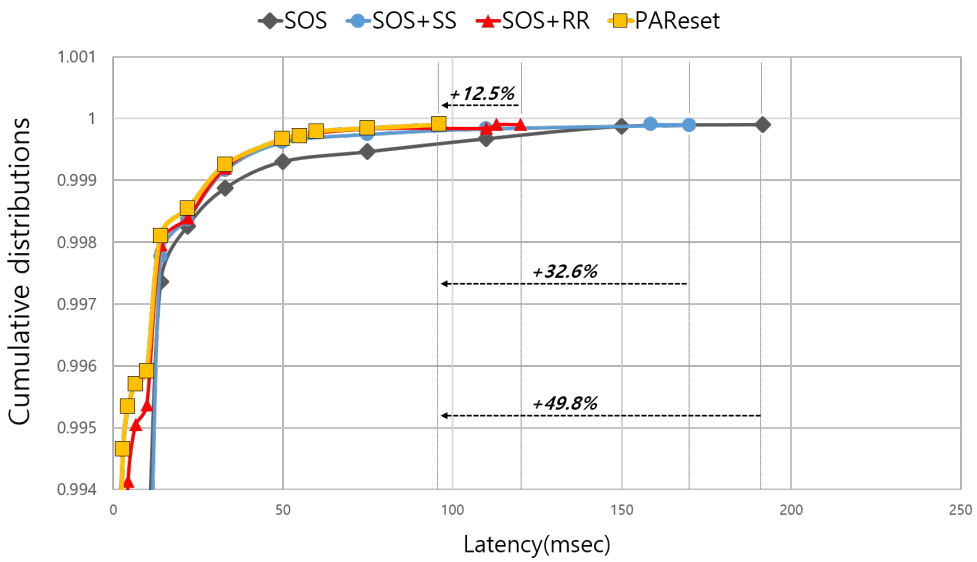

그림 13 PAReset 읽기 응답시간 누적 분포 그래프

제 6 장 결론

6.1 결론

본 연구에서는 기존의 읽기 응답시간의 최적화 기법들을 소개하고 기존 기법들에 대한 한계점을 밝혔다. 선점 가비지 컬렉션의 경우 가비지 컬렉션 동안에 사용자 요청을 받을 수 없었던 문제를 해결하였으나, 저장장치 내에서 읽기 요청이 큐에 의해 지연되는 부분은 여전히 문제로 남아 있었다.

소프트웨어 기반 무순서 중첩 실행 스케줄링 기법에서는 의존성이 없는 사용자 읽기 요청에 대해서 가장 먼저 실행하고, 읽기 요청이 삽입되어 큐에서 대기하기 전에, 큐의 버퍼에서 동일한 데이터가 존재하면 해당 데이터를 읽고 바로 요청을 완료하였다. 또한 중복된 쓰기 연산의 제거 및 쓰기 재할당을 통하여 전체적인 응답시간을 향상시켰다. 하지만 읽기 요청이 진행중인 쓰기/소거 연산을 선점하지 못하고 대기해야 하는 한계가 존재하였다.

쓰기/소거 일시정지 재개 기법은 진행중인 쓰기/소거 연산에 대하여 해당 칩을 읽기 요청이 선점하도록 하여 추가적으로 읽기 응답시간을 향상시키는 노력을 하였다. 하지만 쓰기/소거에 대하여 일시정지를 하는 시간이 읽기 요청에 상대적으로 길기 때문에 이러한 일시정지에 대한 오버헤드를 해결할 필요가 있었다.

따라서 본 연구에서는 쓰기 일시정지에 비하여 5 배 빠르고, 소거 일시정지에 비하여 10 배가 빠른 리셋 명령을 제안하였다. 다만, 리셋 명령의 경우 수명을 단축시키는 문제가 존재하기 때문에 중요한 읽기에 대해서만 선택적으로 적용할 것을 제안하였다.

이러한 문제를 해결하기 위해, SOS 에서 가비지 컬렉션 동안에 유효 페이지 읽기와 사용자 읽기가 서로 다른 우선순위를 갖고 실행되는 것을 SSD 저장장치 밖의 호스트 시스템에서도 이러한 가비지 컬렉션 읽기와 사용자 읽기에 서로 다른 우선순위를 부여하여 선택적으로 처리되도록 하였다.

평가 결과, 사용자 읽기와 쓰기/소거 충돌이 발생하는 모든 경우와 선택적으로 리셋을 적용하는 경우, 평균 읽기 응답시간, 99.99^{th} 읽기 꼬리응답시간, 수명이 각각 9%, 6.9%, 20% 향상되었다. 모든 충돌에 대하여 리셋을 수행하는 경우에 비하여 PAReset 이 더 좋아지는 이유는 기존에 해결할 수 없었던 읽기 요청 간의 큐 지연시간이 해결되기 때문이다. 또한 수명이 리셋을 전혀 사용하지 않는 경우와 비교하여 단 2.8% 정도의 수명 단축이 발생함을 관찰하였다.

6.2 향후 연구

본 논문에서 제안한 PAReset 의 경우 사용자가 우선순위를 정하여 저장장치에 전달해 주며, 이러한 우선순위 정보를 제공받은 저장장치는 짧은 응답시간을 보장하면서 낸드 플래시 메모리의 수명 단축을 최소화하기 위해서, 선택적으로 리셋을 적용한다. 따라서 읽기 우선순위가 존재하는 응용을 찾아야 하며 해당 응용의 수정이 불가피하다. 따라서 사용 범위가 제한적이며, 해당 응용에 대한 추가적인 분석이 필요하다. 이러한 한계를 해결하기 위해서 중요한 읽기 요청을 저장장치 내에서 동적으로 판단하여 리셋을 적용하게 된다면 범용적으로 사용이 가능할 것으로 기대된다.

읽기 응답시간을 보장하면서 수명 단축을 최소화 하는 다른 한가지 방법은 사용자의 우선순위를 기반으로 하는 대신에 읽기와 진행중인 쓰기/소거 요청의 충돌을 최소화하는 방향이다. 만일 저장장치가 특정 칩에서의 읽기와 쓰기/소거 충돌을 완벽하게 예상하고 피할 수 있다면, 리셋을 적용하지 않아도 읽기 성능을 최대로 올릴 수 있을 것으로 기대된다.

참고문헌

[1] Xiangfeng Lu, et al., “Optimizing SSD Latency and Performance in Enterprise Applications” Flash Memory Summit, 2016.

[2] Steven Wells, et al., “Avoiding Costly Read Latency Variations in SSDs Through I/O Determinism” Flash Memory Summit, 2017.

[3] AJOUX, P., BRONSON, N., KUMAR, S., LLOYD, W., AND VEERARAGHAVAN, K. "Challenges to adopting stronger consistency at scale." In HotOS (2015).

[4] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim. A semi-preemptive garbage collector for solid state drives. In ISPASS, pages 12–21. IEEE Computer Society, 2011.

[5] S. S. Hahn, S. Lee, and J. Kim. “SOS: Software-based Out-of-order Scheduling for High-performance NAND Flash-based SSDs.” In IEEE 29th Symposium on Mass Storage Systems and Technologies, 2013.

[6] G. Wu and X. He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In Proceedings of FAST’2012.

[7] J. Gantz et al., “Digital University in 2020: Big Data, Bigger Digital Shadow, and Biggest Growth in the Far East-United States,” IDC, 2013.

[8] RocksDB, https://github.com/facebook/rocksdb/blob/master/docs/_docs/faq.md

[9] RocksDB USERS.md, https://github.com/facebook/rocksdb/blob/master/USERS.md

.

Abstract

Reset-Based Read Optimization for Flash Memory Storage Devices

Jaehoon Lee

Dept. of Computer Science & Engineering

College of Engineering

The Graduate School

Seoul National University

In this paper, we introduce the optimization techniques of read response time. And we propose a reset command to overcome the limitation of existing technology. In addition, we analyze the advantages and disadvantages of each technique by experimenting with both existing techniques and newly proposed reset commands.

The reset command is faster than the suspend command for the read request to preempt the chip. However, reset command has a problem that deteriorates the life of the NAND flash memory. Therefore, it is suggested to reset selectively only for important read requests.

The proposed technique assigns a different priority to garbage collection reads and user reads from the host system and selectively applies the reset.

The priority-based selective reset technique is based on the fact that valid page read requests and user read requests are executed with different priorities during garbage collection in a storage device.

We compared the case of applying a reset on all read requests and the case of applying selective reset on priority basis. In result, average read latency, 99.99th read tail latency, and lifetime improved by 9%, 6.9%, and 20%, respectively.

The reason why the selective reset technique is better than when performing a reset for all read requests is that the queuing delay between read requests is solved. Also, we observed that the life time of the storage device was shortened by only 2.8% compared with the case where the reset was not used at all.

keywords :User read request, Read response time, Selective reset, Priority
Student number : 2016-21226