공학석사학위논문

NVM기반 I/O캐시 레이어에 발생하는 소프트웨어 오버헤드에 관한 실증적인 실험

Empirical Study on Software Overhead in NVM-based I/O Cache Layer

2018 년 6 월

서울대학교 대학원

컴퓨터공학부

이건희

초록

기술발전에 따라 응용들이 처리하는 데이터의 양이 급격히 증가하였고, 이에 따라 빠르게 요청들을 처리해야 하는 필요성이 강조되고 있다. 최근 활발히 연구되고 있는 비휘발성 메모리는 메인메모리와 비슷한 처리 속도와 짧은 응답속도를 지원하여, 메모리 도메인이나 저장장치 계층 에서 중요한 역할을 담당하게 될 것이라고 평가되고 있다. 하지만 기존에 SSD와 하드디스크를 위하여 디자인된 I/O 레이어 모듈들은, 단순히 비휘발성 메모리를 사용하기에는 많은 소프트웨어 스택 오버헤드가 발생하여, 비휘발성 메모리의 속도를 온전히 활용할 수 없다. 본 논문에서는 다양한 실증적인 실험들을 통해 비휘발성 메모리에게 미치는 기존 I/O레이어의 소프트웨어 오버헤드의 정도를 측정해보았다. 또한, 그 결과들을 기반으로 비휘발성 메모리에 최적화된 내부 자료구조 운영 방법과 NVM의 빠른 처리를 도와주는 요청의 특성들을 제안한다.

주요어 : NVM, NVDIMM, I/O 캐시 레이어, 저장장치

학번 : 2016-25737

목차

초록 i

목차 ii

그림 목차 v

표 목차 vi

제 1장 서론

1.1 연구 배경……1

1.2 연구 내용……4

1.3 논문의 구성……5

제 2장 배경 지식

2.1 I/O캐싱 레이어 지원 모듈……6

2.1.1 bcache ……7

2.1.2 dm-cache……8

2.2 비휘발성 메모리 모듈 ……9

제 3장 관련 연구 11

제 4장 비휘발성 메모리 기반 I/O 캐싱 레이어

4.1 실험 환경……14

4.2 기준 실험 ……16

4.3 메타데이터를 읽을 때 일어나는 룩업 오버헤드 실험……17

4.4 스레드 개수에 따른 성능 변화……20

4.5 읽기/쓰기 비율에 따른 성능 변화……23

4.6 NVM기반의 저장장치 시스템 구조 제안……25

제 5장 결론 27

참고문헌 28

Abstract 32

그림 목차

그림 1. bcache와 dm-cache의 내부구조 ……8

그림 2. I/O 캐시레이어의 기본 성능 ……16

그림 3. (a) bcache의 블록/파일 크기 변화에 따른 성능 ……18

그림 3. (b) dm-cache의 블록/파일 크기 변화에 따른 성능 ……18

그림 4. (a) 스레드 개수에 따른 무작위 읽기 요청 성능 ……21

그림 4. (b) 스레드 개수에 따른 순차적 읽기 요청 성능 ……21

그림 5. 읽기/쓰기 비율에 따른 각 모듈의 성능……24

그림 6. 고정된 읽기/쓰기 비율과 스레드 개수 변화에 따른 성능 ……24

표 목차

표 1 시스템을 구성하는 장치들의 평균 응답 속도……10

표 2 실험이 진행된 시스템의 구성요소……15

제 1 장 서론

1.1 연구 배경

기술발전에 따라, 응용들이 처리하는 데이터의 양은 급격하게 증가하고 있으며, 넓은 범위의 도메인에 따른 다양한 형태의 데이터 집약적인 응용들이 나오고 있다. Cassandra[17]나 Redis[15]같은 잘 알려진 빅데이터 관련 응용을 포함하여 초고성능 컴퓨터(High Performance Computing) 내 버스트 버퍼(Burst Buffer)[14]와 같은 모듈 들에서도 단시간에 몰려드는 요청들을 빠르게 처리해야 하는 필요성이 강조되고 있으며 이에 따른 높은 처리량(High-throughput)과 빠른 응답시간(Low-latency)을 지원하는 저장장치의 요구는 점점 증가하고 있다.

저장장치의 발전속도는 점점 응용프로그램 단의 발전속도를 따라가지 못하게 되었고, 저장 장치에 집중되는 병목현상이 계속 되는 것처럼 보였다. 그래서 다음 세대의 저장장치가 나오기 전까지, 기존에

존재하던 저장장치들을 이용하여 위에서 요청되는 I/O들을 빠르게 처리하려는 노력들이 이루어져 왔다. 구동 환경의 I/O 특성에 맞추어 저장장치를 최적화를 하거나, 하나의 SSD에 성격이 다른 칩을 쓰거나, 빠르지만 비교적 용량이 작은 SSD와 느리지만 비교적 용량이 큰 HDD들을 이용하여 가상화를 하는 방법 등으로 저장장치의 병목현상을 극복해왔다[11][12]. 성격이 다른 저장장치들이 저장장치 계층을 구성할 때 이들의 장점만을 활용하는 노력의 흔적들은 리눅스 커널에서도 찾아볼 수 있다. 리눅스 커널 코드에 이미 합병되어 있는 bcache, dm-cache를 포함해서, 커널에 합병 되어있지는 않지만 Facebook에서 만든 flashcache와 그것으로부터 갈라져 나온(forked) 프로젝트인 enhanceio 모두 내부 메커니즘은 다름에도, 하나의 목적을 가지고 있다 [3][5][6][7]. 이는 리눅스 저장장치 스택 다이어그램 상에서 파일시스템과 블록 디바이스 계층 중간에 위치하여, 물리적인 저장장치들을 묶어 하나의 논리적인 저장장치만 상단에 보여주며, 내부적으로 캐싱 하는 등의 방법으로 저장장치의 성능을 높이는 것이다

최근, 비휘발성 메모리 모듈들이 시장에 등장하였고, 메인메모리와 비슷한 처리량과 짧은 대기시간을 지원하기 때문에 비휘발성 메모리를 메모리로서 이용하는 연구들이 활발히 이루어지고 있고, 또한 블록 장치로 잡히는 특성으로 인해 비휘발성 메모리를 저장장치로 이

용하는 연구도 활발히 진행되고 있다. 응용이 다양한 만큼 여러 요구 사항을 만족시키기 위하여 비휘발성 메모리 기반의 저장장치는 다양한 형태로 개발이 되고 있으며 그 중 Intel에서 제시한 Optane memory와 3D Xpoint memory가 활발히 연구가 진행되고 있으며, 그 외 차세대 메모리 기술이라 불리는 STT-MRAM(Spin Transfer torque magnetoresistive RAM)을 포함하여 NVDIMM(Non-volatile Dual In-line Memory Module), Memristor, PCM(Phase Change Memory) 등 많은 종류의 비휘발성 메모리 모듈들이 동시다발적으로 개발이 되고 있다 [1][4][8][9][10][20]. 하지만 위에서 소개된 비휘발성 메모리 기반의 저장장치들은 아직 연구 초기 단계이므로 시중에 판매되고 있는 비휘발성 메모리 모듈들은 여전히 용량당 가격이 아주 비싸다는 단점이 있다. 그래서 비휘발성 메모리만을 사용하여 시스템을 구성하기에는 여전히 가격 부담이 존재한다.

1.2 연구 내용

이 논문에서는 커널 에서 지원하는 I/O캐싱 모듈인 bcache[3]와 dm-cache[5] 를 이용하여 여러 실증적인 실험을 통해 고성능의 비휘발성 메모리가 기존 I/O 캐싱 레이어의 캐싱 장치로 사용되었을 때 발생하는 소프트웨어 오버헤드를 측정 해 볼 것이다. 진행된 실험들은 다음과 같다.

1. 메타데이터를 읽을 때 일어나는 오버헤드

2. 워킹 세트 크기 변화에 따른 성능 변화

3. 스레드 개수에 따른 성능 변화

4. 요청의 읽기/쓰기 비율에 따른 성능 변화

또한, 위 실험들의 결과를 기반으로 고성능의 비휘발성 메모리를 이용하여 I/O 캐싱 레이어를 구축할 때 더 좋은 성능을 낼 수 있는 가능한 방안들을 제시한다.

1.3 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 다수의 저장장치를 묶어서 하나의 논리적인 장치만 보여 주며 저장장치의 성능향상을 도와주는 I/O캐싱 레이어의 모듈들과 실험에 사용된 비휘발성 메모리인 NVDIMM 에 대해서 간단히 소개한다. 3장에서는 본 논문과 관련된 연구들을 소개하고, 4장에서는 여러 실증적인 실험들을 통해 저장장치 각각의 독립적인 성능과 비휘발성 메모리를 캐싱장치로서 기존 I/O 캐싱 레이어를 이용하였을 때의 소프트웨어 스택의 오버헤드의 정도와 요청의 특성에 따른 성능 변화에 대해서 알아본 뒤, 5장에서 결론을 맺는다.

제 2 장 배경지식

2.1 I/O 캐싱 레이어 지원 모듈

서론에 언급되었듯이, bcache, dm-cache, flashcache 그리고 enhancio 등은 낸드 플래시 기반의 SSD나 하드디스크 같은 다수의 물리적 저장장치를 하나의 논리적 저장장치로 묶어서 상단에 보여주는 모듈이며, 캐싱 메커니즘 이용하여 보다 효율적이고 빠른 I/O 처리를 도와준다. 또한 파일시스템과 독립적으로 작동하기에 파일시스템과 상관없이 투명하게(Transparent) I/O 처리성능을 높여준다. 조금씩은 서로 방법이 다르지만, 공통으로 위 모듈들을 사용하기 위해서는 캐싱 용도를 위한 속도가 빠른 저장장치를 지정하고 속도는 느리지만 용량이 큰 백엔드 저장장치를 지정해야 한다. 그 후, 내부 알고리즘에 의해, 가까운 미래에 접근될 확률이 높은 데이터들은 캐싱 장치로 지정된 저장장치에 캐싱시켜서 추후 요청에 대해 빠르게 서비스할 수 있게 하고, 가까운 미래에 접근될 확률이 낮은 데이터들은 백엔드 저장장치(Back-end Storage)에 저장하게 된다.

2.1.1 bcache

bcache는 리눅스 커널 3.10버전 이후로 커널 코드와 합병되어 리눅스 사용자들에게 제공된다. 그림 1은 bcache의 간단한 내부 구조를 보여준다. bcache는 내부적으로 캐시 데이터를 B+ tree 자료구조를 기반으로 관리하게 된다. 요청이 올 때마다 요청되는 논리 주소가 key값이 되어 btree의 버킷들을 탐색하게 되고, 캐시된 데이터가 없으면 캐시 미스로 처리되어 새로운 버킷을 할당받은 뒤 데이터를 입력하고 copy on write(COW) 방식으로 버킷이 교체된다. 또한, 사용 가능한 버킷이 없으면 버킷을 재사용하기 위해 쓰레기 수거(Garbage Collection) 동작이 진행되게 된다. 그 외 읽기나 쓰기 요청 간의 락킹(Locking)이나 업데이트에 대한 저널링(Journaling), 그리고 btree 노드 내부 탐색 시 순차적 일관성(Sequential Consistency) 정책 등의 복잡한 메커니즘을 포함하여 설계되었다.

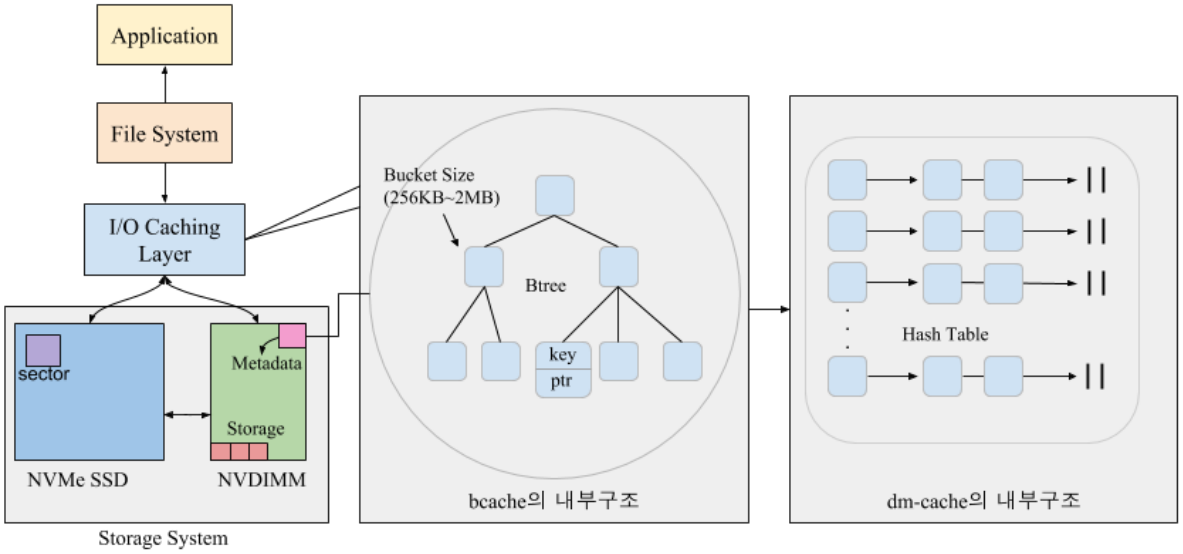


그림 1 – bcache와 dm-cache의 내부구조

2.1.2 dm-cache

dm-cache는 리눅스 커널의 디바이스 매퍼의 component 중 하나이며 커널 3.9버전 이후부터 소스코드에 포함되어있다. dm-cache 내부적으로 해시 연결 리스트로 관리하지만, 정책들을 살펴보면 CPU 캐시와 비슷한 모델로 캐시 데이터를 관리한다. 캐시 블록의 크기는 모듈 셋업 단계에서 지정하게 되며 32KB부터 1GB까지 32KB 단위로 자유롭게 지정할 수 있다. 한번 지정된 캐시 블록은 중간에 조정할 수 없기에 워크로드 패턴에 따른 상황대처능력은 떨어지게 된다. bcache에서는 쓰레기 수거 작업

을 통해 유효하지 않은 블록들을 재사용 하였지만, dm-cache에서는 clean한 블록과 dirty한 블록을 보관하는 다중 큐를 유지하여 clean한 블록들을 먼저 희생블록으로 선택하며 공간을 확보한다. bcache는 내부적으로 복잡한 자료구조와 정책들을 사용하던 반면 dm-cache는 전체적으로 간단하고 효율적인 내부 구조로 구성 되어 있다.

2.2 비휘발성 메모리 모듈

본 논문에서 사용된 NVDIMM(Non-Volatile Dual In-line Memory Module)은 Netlist, Smart, Agiga 혹은 Micron 등의 벤더에서 활발히 생산하고 있는 비휘발성 메모리 모듈이다 [2][16][20]. 대표적인 특징은 이름에서도 알 수 있듯이 DIMM 소켓에 장착되어 메모리 버스를 통해 CPU와 통신을 하는데, 이는 여태 존재했던 어떤 저장장치보다 CPU와 가까운 거리에서 통신한다는 이점이 있다. 그로 인해 높은 처리량과 빠른 응답시간을 보장해주게 되고, 그 뿐만 아니라 직접 접근(Direct Access)을 지원하여 불필요한 동작들의 오버헤드를 줄이고, 블록 디바이스로 인식이 되며 유연성을 높일 수 있기에, 빅데이터 관련 작업이나 초고성능컴퓨터 응용 등 신속한 I/O처리가

모듈 종류	평균 응답 속도(ns)
SRAM	2-3
DRAM	10-100
NVDIMM-N (DAX)	200-900
NVDIMM-N (BLOCK)	5,000-7,000
NVMe SSD	60,000-70,000
SSD	90,000-300,000

표 1 – 시스템을 구성하는 장치들의 평균 응답 속도(ns)

필요한 환경에서 중추적인 역할을 할 것으로 기대하고 있다.

표 1을 참조하면 NVDIMM의 평균 응답 속도는 메인메모리와 NVMe SSD 중간에 위치하여 두 모듈 간 I/O 속도 차이를 메워주고 있는 것을 확인할 수 있다. NVDIMM은 대표적으로 3가지 종류가 있는데, 일반 메인메모리과 비슷한 용량과 속도를 지원하며 전원공급이 중단되어도 데이터를 유지할 수 있게 일시적으로 전력을 공급해주는 커패시터(capacitor)가 달린 N 타입, 속도는 기존 메인메모리보다 느리지만 크기는 일반적인 SSD만큼의 용량을 지원하는 스토리지 형식의 F 타입, 그리고 N 타입과 F 타입의 장점만을 흡수한 메인메모리와 같은 속도를 가지며 일반 저장장치 크기의 큰 용량을 지원하는 특성을 가진 P 타입으로 나누어져 있다[18]. 이 논문에서는 비휘발성 메모리 중 하나인 NVDIMM-N 타입을 이용하여 실험하였다.

제 3 장 관련연구

본 논문과 관련하여 비휘발성 메모리를 이용한 실증적인 실험을 통해 기존 시스템에 아무 변경 없이 비휘발성 메모리가 편입되었을 때, 어떠한 현상이 나타나는지에 대해 연구한 논문들을 소개하려 한다.

[13] 은 비휘발성 메모리가 저장장치로 이용되었을 때, 운영체제의 관점에서 어떠한 문제가 발생하는지 다양한 실험을 통해 설명한다. 비휘발성 메모리를 버퍼캐시로 이용하는 것과 Direct I/O를 요청하는 방식의 비교를 포함하여, 미리 읽기(Read Ahead) 기능이 비휘발성 메모리 기반의 저장장치에 미치는 영향, 같은 크기의 데이터를 요청할 때, 횟수를 몇 번으로 나누어서 줘야 제일 효율적인지, 그리고 데이터를 얼만큼의 크기로 주어야 비휘발성 메모리에 제일 적합한지 등에 관련된 실험들로 구성되어 있어, 비휘발성 메모리와 통신할 때 적합한 값을 찾는 것을 도와준다.

[21]는 비휘발성 메모리가 메인메모리로 사용되었을 때 어떻게 더 잘 활용할 수 있는지 실험을 통해 설명한다. 논문의 중점은, 업데

이트된 데이터에 대해서 permanent 저장장치로 flush 동작을 할 때의 오버헤드에 관련된 것이다. 보통 백엔드 저장장치와의 동기화를 위한 sync 요청은 msync 나 fsync같은 명령어들로 이루어지는데 그 과정에서 순서를 보장해주는 sfence나 barrier 같은 동작이 중첩되면 더욱 오버헤드가 심해진다고 설명한다. 그래서 순서 보장의 정도를 약하게 해서 비휘발성 메모리의 속도를 극대화 할 수 있는 Weakly-ordered clflush(WOF) 방법을 제시한다. 그뿐만이 아니라, WOF방법을 포함한 6가지 다른 방법으로 파일서버, 웹서버, 메일서버, MongoDB, MySQL 그리고 Memcached 등 다양한 워크로드에 대해서 성능을 측정하게 된다. 또한, 다양한 도메인에서 생성되는 워크로드에 대해 각각 어떤 flush 메커니즘을 이용해야 가장 비휘발성 메모리를 잘 사용할 수 있는지 많은 실험을 통해 제안한다.

[19] 또한 비휘발성 메모리를 메인메모리로 이용할 때, 파일시스템의 입장에서 어떻게 잘 활용할 수 있을지에 관해 설명한다. 이 논문에서 실험한 주제들은 다음과 같다. 처음으로, 파일시스템의 방식 중 제자리 업데이트(In-place Update)를 지원하는 파일시스템과 로그 구조의 파일시스템 (Log-structured) 중 어떤 종류의 파일시스템이 비휘발성 메모리와 더 조화가 잘 되는지에 대한 실험으로 시작된다. 그 이후, XIP(Execute-In-Place)의 방식과 버퍼 캐시를 사용하는 방식의 비교, 병렬적으로 할당(Parallel Allocation)하는 것과 그렇

지 않은 방식에 대한 비교, 고정된 블록 크기와 가변 블록크기에 대한 비교 등 흥미로운 실험들이 진행된다. [15] 논문과 비슷하게 XFS, PMFS, 그리고 ext2,3,4를 포함한 7개의 파일시스템에 대하여 파일 서버, 웹 서버, 데이터베이스 관련 등 다양한 도메인의 워크로드를 이용하여 실험이 진행된다.

[13][19][21] 과 본 논문의 차이점은, 위 논문들에서는 메인메모리나 램디스크(RamDisk) 등을 이용하여 비휘발성 메모리의 응답속도와 처리량 등을 유사하게 설정 후 시뮬레이션하여 실험들을 진행하였고, 본 논문은 비휘발성 메모리 모듈 중 하나인 NVDIMM을 이용하여 실험들을 진행하여서 데이터들이 비교적 정확 하다는 점이다. 하지만 비휘발성 메모리 모듈을 기존 시스템에 사용하였을 때, 어떤 현상이 발생하는지 실증적인 실험을 통해 확인해보려는 점은 공통적이라고 볼 수 있다.

제 4 장 비휘발성 메모리 기반 I/O 캐싱 레이어의 성능 평가

4.1 실험 환경

이 섹션에서는 하드디스크와 SSD, 그리고 비휘발성 메모리를 저장장치로 이용했을 때 각각의 독립적인 성능과 비휘발성 메모리 모듈인 NVDIMM이 리눅스 커널에 포함된 I/O캐싱 레이어 모듈인 dm-cache와 bcache에서 캐싱 장치로 사용되었을 때의 성능을 확인해보려 한다. 실험이 진행된 환경의 시스템 구성은 표2와 같다.

NVDIMM모듈을 지원하는 슈퍼마이크로 사의 X11DPI-N 모델을 메인보드로 이용하였고, 메인보드와 호환되는 CPU를 선정하였는데, 인텔사에서 제조한 제온 확장 프로세서 패밀리(Xeon Scalable Processor Family) 중 실버 4114 모델을 사용하였다. 그리고 비휘발성 메모리는 Smart사에서 판매하는 16GB 크기의 NVDIMM-N타입 모듈 2개를 장착하였고, 메인메모리 또한 삼성에서 판매하는 16GB 크기의 모듈 2개를 장착하였다.

CPU	Intel Xeon Silver 4114 Processor (13.75M Cache, 2.20 GHz, 10 core)
Mainboard	SUPERMICRO X11DPI-N
Memory	16GB * 2 of REG. ECC DDR4 SDRAM (2666Mhz), Product of Samsung
NVDIMM	16GB * 2 of NVDIMM-N type DDR4 (2400MHz), Product of SMART
SSD	250GB MZ-V6E250 960 Evo NVMe 1.2 SSD M.2, Product of Samsung
HDD	1.0TB WD BLUE WD10EZEX SATA3, Product of Western Digital

표 2 – 실험이 진행된 시스템의 구성 요소

백엔드 저장장치는 SSD와 HDD를 장착하였는데, SSD는 PCIe 버스를 이용하는 고성능 SSD인 NVMe SSD를 이용하였고, HDD는 웨스턴 디지털사의 1TB크기의 하드디스크를 이용하여 시스템을 구성하였다.

실험이 진행된 환경의 운영체제는 CentOS 7 이며 3.10 버전의 커널을 사용하였으며, bcache는 3.10버전, 그리고 합성 워크로드를 생성하는 프로그램인 FIO(Flexible I/O)[23] 는 3.1버전을 이용하였다.

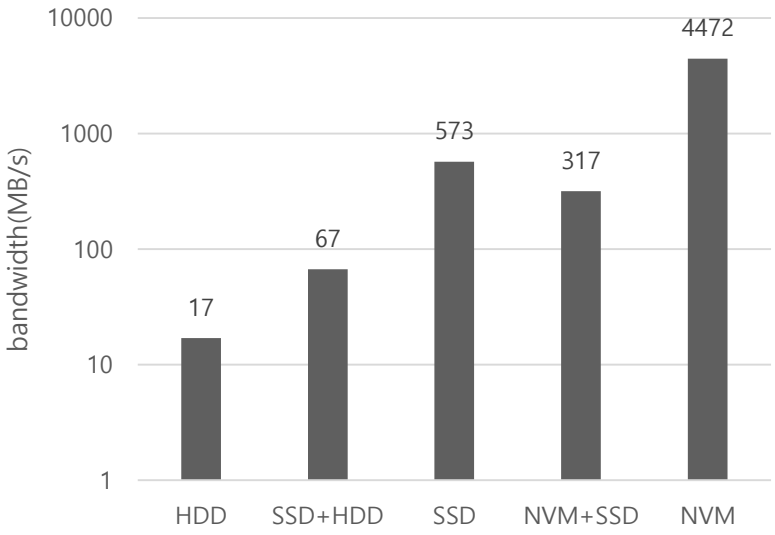


그림 2 – 각 저장장치와 I/O캐싱 모듈들(SSD+ HDD, NVM+ SSD)의 로그 스케일 대역폭

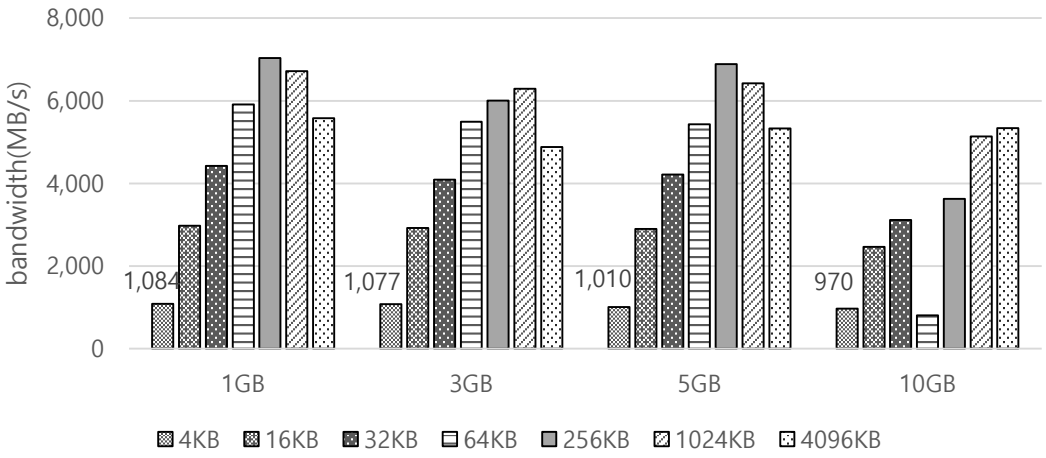
4.2 기준 실험

그림 2는 각 독립적인 저장장치와 두 개 이상의 저장장치들이 I/O캐싱 모듈을 구성할 때의 대역폭을 로그스케일로 나타낸다. 무작위 한 패턴과 순차적인 패턴이 합쳐진 요청들을 생성하였고, 동기적(synchronous)인 요청을 보냈으며 페이지 캐시를 우회하기 위해 direct 특성을 포함하여 설정하였다. 각 저장장치가 독립적으로 요청들을 처리할 때는 정상적인 속도 범위 내에서 요청이 처리되었음을 확인하였다. 또한, 원래의 dm-cache와 bcache의 목적대로 SSD를 캐싱장치로 이용하고 HDD를 backing 장치로 이용하였을 때, 단일

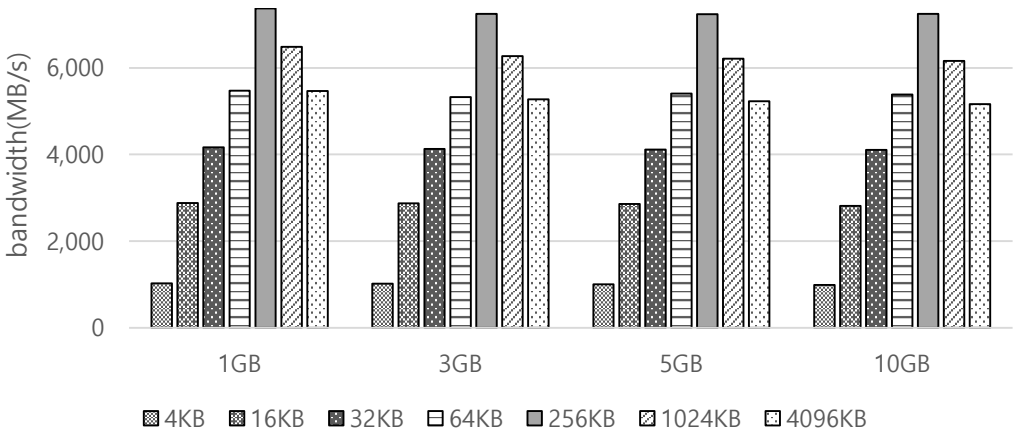
HDD의 성능보다는 좋고 SSD 성능에는 미치지 못하는 성능을 확인할 수 있었다. 하지만 같은 상황에 대해 비휘발성 메모리 기반의 저장장치가 캐싱 장치로 선택되었을 때, 오히려 단일 비휘발성 메모리의 처리장치보다 좋지 않은 성능이 나오는 것을 볼 수 있었다. 이는 캐싱 장치가 SSD가 사용될 것을 가정하고 설계된 I/O캐싱 레이어들의 내부 설계에 존재하는 복잡성 때문에 발생하는 오버헤드이다.

4.3 메타데이터를 읽을 때 일어나는 룩 업(look-up) 오버헤드 실험

다음 실험에서는 NVDIMM 독립적인 장치와 I/O 캐싱 모듈인 bcache와 dm-cache에 대해 다양한 블록 크기와 다양한 파일 크기로 요청을 보낼 때 발생하는 look up 오버헤드를 측정하기 위해 실험이 진행되었다. 4KB부터 4MB까지 크기의 블록을 요청하였으며 총 파일의 크기가 1GB, 3GB, 5GB 그리고 10GB 일 때의 성능을 각각 측정하였다.



(a) bcache 모듈의 블록 크기와 파일 크기에 변화에 look-up 오버헤드 측정



(b) dm-cache 모듈의 블록 크기와 파일 크기에 변화에 look up 오버헤드 측정

그림 3. 각 모듈이 비휘발성 메모리를 캐싱장치로 이용하고, SSD를 백엔드 저장장치로 이용할 때 요청 블록 크기와 파일 크기의 변화에 따른 look up 오버헤드 측정 실험 결과

그림 3은 비휘발성 메모리의 캐싱장치와 SSD가 백엔드 저장장치로 지정되었을 각 bcache와 dm-cache의 성능을 보여준다. 실험은 하나의 스레드에서 무작위의 주소에 대해서 동기적으로 읽는 요청으

로 진행하였다.

그림 3(a)을 참조하면 위 상황에 대한 bcache의 성능을 확인할 수 있다. bcache는 요청의 블록 크기가 4KB일 때 가장 안 좋은 성능을 보였다. 이는 4KB의 작은 요청들이 들어오게 되면 B+ tree 구조를 따라 단계적인 탐색을 통해 key를 확인하며 매칭되는 버킷을 찾고, 그 이후 해당 버킷에 대해 순차적으로 확인하게 되면서 발생하는 look up 오버헤드 때문이다. 이는 bcache를 사용할 때 나타나는 대표적인 소프트웨어 오버헤드이며 비휘발성 메모리의 빠른 성능을 온전히 사용하지 못하게 하는 원인이 된다. 그 이후 요청의 블록 크기가 커질수록 성능은 향상하며 파일의 크기가 5GB를 넘어서게 되면서 성능이 떨어지는 것을 확인하였다. 5GB 이하의 작은 파일에 대해서는 성능의 변화가 미미하였다.

그림 3(b)은 위와 같은 상황에 대해 dm-cache 의 성능을 나타낸다. bcache와 마찬가지로 요청의 블록 크기가 4KB정도로 작을 때 잦은 look up으로 인한 오버헤드가 발생하여 성능이 느려지는 것을 확인하였고, 파일의 크기에 상관없이 요청의 블록 크기가 256KB일 때 가장 좋은 성능을 보이는 것을 확인하였다. 이는 실험을 위해 dm-cache를 셋업 할 때 메타데이터 블록 크기를 설정하지 않아서 기본값인 256KB의 블록 크기로 설정이 되었고, 이를 통해 요청의

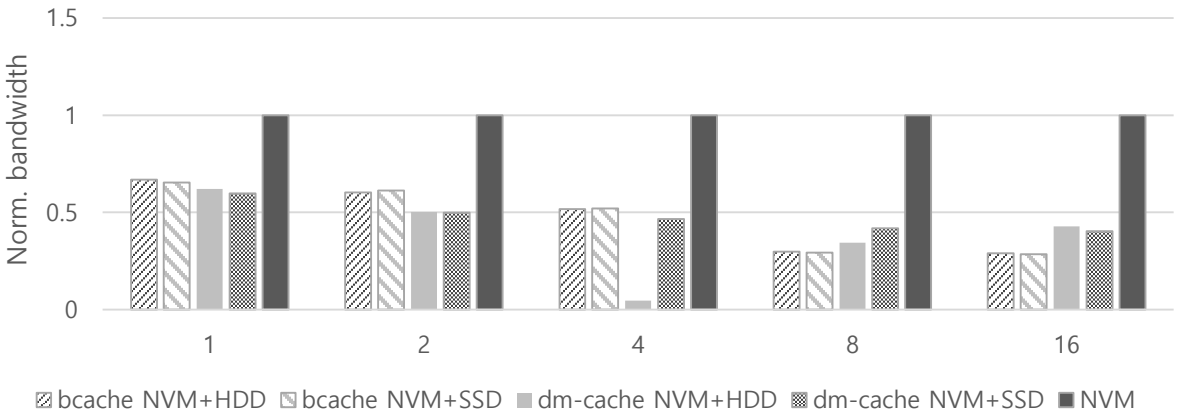
블록 크기가 메타데이터가 관리하는 블록의 크기가 같을 때 가장 좋은 성능을 내는 것을 확인할 수 있었다.

위 실험을 통해 요청의 블록 크기와 파일 크기에 따른 두 개 모듈의 성능 변화를 살펴보았다. 두 모듈 전부 다수의 작은 크기의 요청이 오게 되면 잦은 테이블 룩 업이 일어나게 되고, 내부적으로 캐시된 데이터를 탐색하는 과정에서 발생하는 오버헤드로 인한 성능 저하가 발생하는 것을 확인하였다.

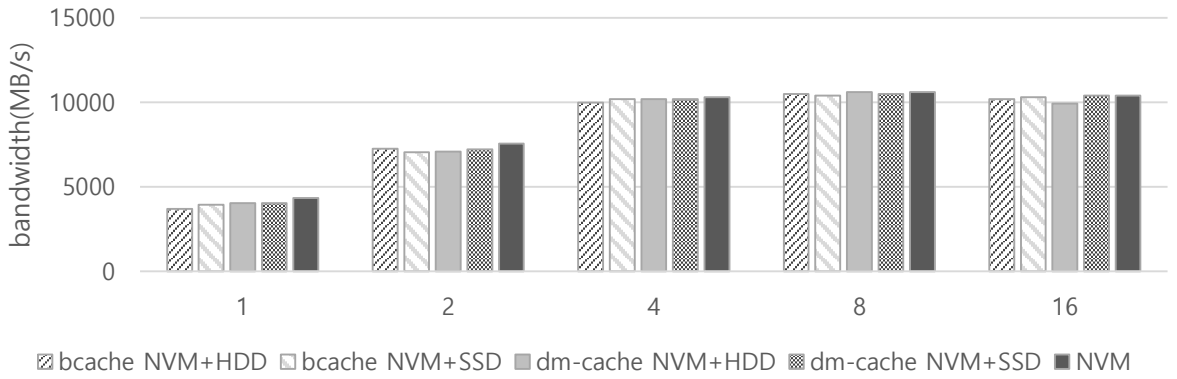
4.4 스레드 개수에 따른 성능 변화

다음 실험에서는 NVDIMM 독립적인 장치와, I/O 캐싱 모듈인 bcache와 dm-cache에 대해 스레드 개수에 따른 성능 변화를 확인한다. 4KB의 무작위 읽기 요청과 4MB의 순차적인 읽기 요청에 대해 각각 실험이 진행되었다.

그림 4(a)는 NVM, bache, 그리고 dm-cache의 무작위 읽기 요청에 따른 대역폭을 나타내며 단일 NVM의 성능으로 정규화 (Normalized) 되어 있다. 스레드 하나가 요청을 보낼 때의 성능은 bcache 그리고 dm-cache 둘 다 NVM 성능의 60% 조금 넘는 수치



(a) NVM, bcache, 그리고 dm-cache의 스레드 개수에 따른 무작위 읽기 요청의 대역폭



(b) NVM, bcache, 그리고 dm-cache의 스레드 개수에 따른 순차적 읽기 요청의 대역폭

그림 4. 스레드 개수에 따른 NVM, bache 그리고 dm-cache의 읽기 요청에 대한 대역폭

를 보인다. 그 후 스레드가 2개 일 때는 bcache의 성능이 dm-cache의 성능을 추월하였고, 스레드가 8개 일 때부터는 dm-cache가 bcache보다 좋은 성능을 보였다. 두 I/O 캐싱 레이어 모두 내부적으로 lock을 사용하는데 스레드의 개수가 많아질수록 bcache가 성능이 나빠지는 이유를 살펴보니, bcache는 내부적으로 큰 범위에(Coarse-

grained) 대해 lock을 잡는 것을 확인하였다. B+ tree에 작업이 일어나게 되면 트리를 탐색 하는 과정에서 루트 노드부터 lock을 걸고 내부를 탐색하는 메커니즘으로 되어있다. 그로 인해 스레드 개수가 늘어날수록 성능은 계속 낮아지는 트렌드를 보이며, 스레드 개수가 16개 일 때는 단일 NVM에 비해 성능이 30% 도 나오지 않는 것을 확인 하였다.

그림 4(b)는 크고 순차적인 읽기 요청이 왔을 때의 성능 변화를 나타낸다. 이전 무작위 읽기보다 스레드 개수가 늘어날수록 NVM의 성능과 비슷해지는 결과를 보였다. 스레드가 4개 일 때 대역폭은 포화(saturated)된 것을 볼 수 있는데, 비휘발성 메모리의 한계 대역폭은 그것보다 높지만, 커널의 소프트웨어 스택 오버헤드로 인해 한계 대역폭이 10GB/s 조금 넘는 수치를 보이는 것을 확인 할 수 있었다.

이번 실험에서는, 스레드 개수에 따른 bcache와 dm-cache의 성능 변화를 살펴보았고, 두 모듈의 비교적 큰 범위의 lock으로 인해 스레드의 개수가 늘어날수록 성능이 저하되는 것을 볼 수 있었다.

4.5 요청의 읽기/쓰기 비율에 따른 성능 변화

다음 실험은 요청의 읽기/쓰기 비율의 변화에 따른 각 I/O캐싱 모듈의 성능 변화를 알아본다. 실험은 읽기의 비율이 100%, 99%, 95%, 90%, 70%, 50%, 30%, 그리고 10% 인 상황에 대해 진행 되었다. 스레드 1개가 요청을 보내며, 블록 크기는 4MB로 설정하였고 순차적인 읽기/쓰기 패턴으로 실험이 진행되었다.

그림 5는 각 NVM과 각 모듈의 저장장치 조합에 대하여 다양한 읽기/쓰기 비율에 따른 성능을 보여준다. 읽기의 비율이 100% 일 때는 모든 케이스가 NVM에 가까운 대역폭을 보였다. 하지만 쓰기 요청이 1% 라도 섞이기 시작하는 순간 bcache의 성능이 NVM의 성능의 58%정도까지 떨어지는 것을 확인할 수 있었다. 읽기 비율이 낮아질수록 dm-cache는 꾸준히 NVM과 비슷한 성능을 내는데 반면, bcache는 읽기 비율이 떨어질 때 마다 성능도 같이 하락하는 트렌드를 보였다. 읽기의 비율이 10%가 됐을 때, bcache의 성능은 NVM 대비 10% 밖에 내지 못하는 수치를 보였다.

이 성능 하락의 원인은 이전 실험과 마찬가지로 bcache의 넓은 lock의 범위(granularity)와 관련이 있는데, bcache는 내부적으로 아주 조심스러운 lock 운영을 하며 B+ tree의 자식 노드로 이동을 해

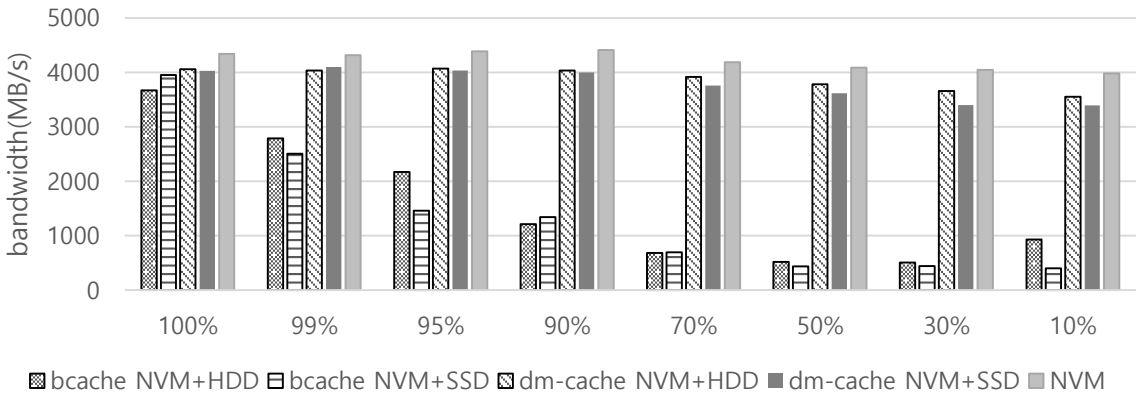


그림 5. 읽기의 비율이 100%부터 점차 내려갈 때의 각 모듈의 성능

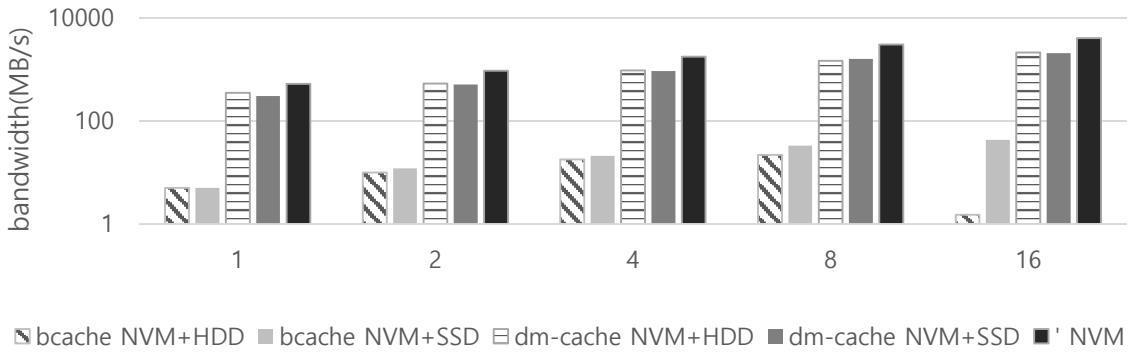


그림 6. 4KB 의 무작위 패턴의 읽기/쓰기 요청이 5:5 비율로 고정되어 있고, 스레드의 개수가 늘어갈 때 각 모듈의 성능

야할 때도 부모 노드와 루트 노드까지 넓은 범위에 대하여 lock을 잡고 동작이 이루어지게 된다.

이와 관련되어 추가적인 실험을 진행해보았다. 이전에 진행된 스레드의 개수 별로 성능의 변화를 확인하였었는데, 이번에는 두 모듈에 대해 읽기와 쓰기의 비율이 50:50일 때, 스레드 수가 증가함에 따라 어떤 성능 변화를 보일지 확인해보았다. 그림 6은 무작위 패턴

의 읽기와 쓰기 요청이 5:5 비율로 고정이 되어있고, 스레드 개수가 늘어날 때의 성능변화를 보여준다. 이번 실험에도 알 수 있듯이, bcache의 경우 조금의 강도 높은 쓰기 요청만 들어와도 lock 문제로 인해 심각한 오버헤드를 만들고 있다는 것을 확인하였다.

4.6 NVM기반의 저장장치 시스템 구조 제안

여러 실증적인 실험들을 통해 비휘발성 메모리가 I/O 캐싱 레이어의 캐싱 장치로 이용이 될 때 겪는 소프트웨어 오버헤드와 요청의 특성에 따라 일어나는 성능변화에 대하여 알아보았다. 실험 결과를 통해 얻은 정보를 바탕으로, 미래에 NVM기반의 저장장치 시스템을 구축할 때 중요하게 생각되어야 하는 디자인 이슈 몇 가지를 제안한다.

첫째는 간단한 캐시 메타데이터 자료 구조와 관리 메커니즘이다. 실험을 통해 알아보았듯이, 기존의 저장장치들을 위해 설계된 I/O 캐싱 모듈들은 복잡한 내부 자료구조와 관리 메커니즘으로 인해 고성능의 비휘발성 메모리의 성능을 완전히 활용하지 못한다는 것을 확인하였다. NVM의 속도를 극대화하려면, 최소한의 복잡성을 가지고

있는 자료구조를 사용하여 디자인 하면 NVM 최적화에 많은 도움이 될 것이라 믿는다.

두 번째는, lock 관련 문제이다. 실험에 사용된 I/O 캐시 모듈은 스레드의 개수가 늘어남에 따라 급격한 성능저하를 보였다. 큰 범위의 lock 잡으며 발생하는 오버헤드는 적지 않은 수치였으며, NVM을 위한 구조를 디자인할 때 꼭 중요하게 다루어야 할 부분이라 믿는다. Lock 오버헤드를 최소화하기 위해서 더욱 좁은 범위의(fine-grained) lock을 활용하거나 lock-free 디자인을 이용하면 NVM의 성능을 극대화할 수 있을 거라 확신한다.

제 5 장 결론

최근 다양한 연구가 활발히 진행 중인 비휘발성 메모리는, 메인메모리와 비슷한 빠른 성능을 가지고 있어서 기존 저장장치의 I/O 메커니즘을 따르게 되면 소프트웨어 스택 오버헤드 때문에 완전한 성능을 기대하기 어렵다. 본 논문에서는, 기존에 SSD와 HDD를 묶어 하나의 논리적인 저장장치로 상위 레이어에 보여주는 bcache모듈과 dm-cache 모듈을 이용하여 비휘발성 메모리가 I/O 캐싱 레이어의 캐싱 장치로 이용되었을 때 나타나는 소프트웨어 오버헤드들에 대해서 실증적인 실험들을 통하여 알아보았고, 또한 워킹 세트 크기, 스레드 개수, 그리고 읽기 쓰기 비율에 따른 NVM기반 캐싱 레이어의 성능 변화를 알아보았다.

논문에서 연구된 비휘발성 메모리를 I/O캐싱 레이어에 편입시켜 성능을 향상시키는 것과 관련된 향후 연구 과제는 다음과 같다. 복잡하지 않은 자료구조들과 그에 따른 정책들을 구현하여 메타데이터를 관리 할 때 비휘발성 메모리가 느끼는 소프트웨어 오버헤드를 최소화 할 수 있는 방향으로 연구가 진행되면 의미 있는 결과를 얻을 수 있을 것 같다.

참고 문헌

[1]3D XPoint. (2015). Retrieved from http://www.micron.com/products/advanced-solutions/3d-xpoint-technology

[2]AGIGARAM NVDIMMs. (2018). Retrieved from http://agigatech.com/products/agigaram-nvdimms/

[3]Bcache. (2018). Retrieved May, 2018, from https://bcache.evilpiepirate.org/

[4]Breitwisch, M. J. (2009). Phase Change Random Access Memory Integration. Phase Change Materials, 381-408. doi:10.1007/978-0-387-84874-7_17

[5]Dm-cache. (2018). Retrieved from https://www.kernel.org/doc/Documentation/device-mapper/cache.txt

[6]EnhanceIO. (2018). Retrieved May, 2018, from https://github.com/stec-inc/EnhanceIO

[7]Flashcache. (2018). Retrieved May, 2018, from https://github.com/facebookarchive/flashcache

[8]Ho, Y., Huang, G. M., & Li, P. (2009). Nonvolatile memristor memory. Proceedings of the 2009 International Conference on Computer-Aided Design - ICCAD 09. doi:10.1145/1687399.1687491

[9]Intel® Optane™ Memory. (n.d.). Retrieved May, 2018, from http://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html

[10]Kawahara, T. (2011). Scalable Spin-Transfer Torque RAM Technology for Normally-Off Computing. IEEE Design & Test of Computers, 28(1), 52-63. doi:10.1109/mdt.2010.97

[11]Kgil, T., Roberts, D., & Mudge, T. (2008). Improving NAND Flash Based Disk Caches. 2008 International Symposium on Computer Architecture. doi:10.1109/isca.2008.32

[12]Lee, C., Sim, D., & Hwang, J. (2015). F2FS: A New File System for Flash Storage. 13th USENIX Conference on File and Storage Technologies (FAST '15)., 273-286.

[13]Lee, E., Bahn, H., Yoo, S., & Noh, S. H. (2014). Empirical Study of NVM Storage: An Operating Systems Perspective and Implications. 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems. doi:10.1109/mascots.2014.56

[14]Liu, N., Cope, J., Carns, P., Carothers, C., Ross, R., Grider, G., Maltzahn, C. (2012). On the role of burst buffers in leadership-class storage systems. 012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST). doi:10.1109/msst.2012.6232369

[15]MongoDB. (2018). Retrieved from https://www.mongodb.com/

[16]Netlist Inc. NVDIMM. (n.d.). Retrieved from http://www.netlist.com/products/vault-memory-storage/nvvault-ddr4-nvdimm/

[17]Redis. (2018). Retrieved from https://redis.io

[18]Sainio, A. (2016, May 23). NVDIMM: Changes are Here So What's Next? Speech presented at In-Memory Computing

Summit 2016 in Grand Hyatt, San Francisco.

[19]Sehgal, P., Basu, S., Srinivasan, K., & Voruganti, K. (2015). An empirical study of file systems on NVM. 2015 31st Symposium on Mass Storage Systems and Technologies (MSST). doi:10.1109/msst.2015.7208283

[20]SMART's NVDIMM with SafeStor™ Technology. (n.d.). Retrieved from http://www.smartm.com/products/dram/NVDIMM_products.asp

[21]Zhang, Y., & Swanson, S. (2015). A study of application performance with non-volatile main memory. 2015 31st Symposium on Mass Storage Systems and Technologies (MSST). doi:10.1109/msst.2015.7208275

[22]Joos, M., & Zipf, G. K. (1936). The Psycho-Biology of Language. Language, 12(3), 196. doi:10.2307/408930

[23]Fio. (2018). Retrieved from https://github.com/axboe/fio

Abstract

An Empirical Study on NVM based I/O Cache Layer

As technology advances, the amount of data processed by applications is growing exponentially, and various types of data intensive applications are emerging across a number of domains. Accordingly, the demand for storage devices supporting high-throughput and low-latency is rapidly increasing. Emerging non-volatile memory based storage devices, they are considered to be an important device that can resolve the storage bottleneck by supporting significantly higher throughput and shorter latency compared to existing I/O devices. However, applying NVM in system without a modification makes significant software overhead due to I/O layers that are originally made for SSD and HDD. Thus, this paper presents an empirical study on Non-volatile memory based I/O Cache layer when combining the performance benefits of emerging NVM storages and the cost-effectiveness

of secondary storages. With fast NVM storages, locating data must be handled efficiently, but the sophisticated yet complex data structures used impose significant overheads by substantially increasing the hit time. As this design approach is suboptimal for accessing fast I/O devices, we suggest several architectural designs to exploit the performance of NVM storages.

Keywords: NVM, NVDIMM, I/O Cache Layer, Storage Stack

Student Number: 2016-25737