PFA: Linux 페이지 폴트 서비스를 위한 통합된 성능 분석 도구

한승욱 O, 한상욱, 김지홍
서울대학교 컴퓨터공학부
{swhan, shanehahn, jihong}@davinci.snu.ac.kr

PFA: An Integrated Performance Profiling Tool for Linux Page Fault Service

Seungwook Han O, Sangwook Shane Hahn, and Jihong Kim
Department of Computer Science and Engineering, Seoul National University

요 약

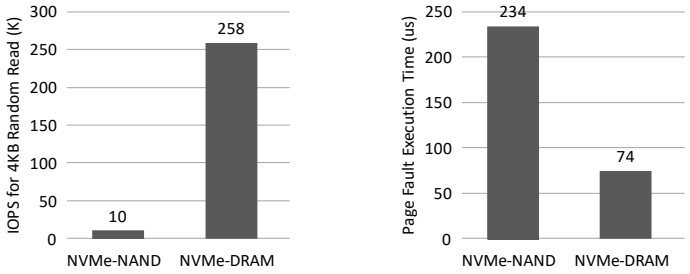
최근 응답시간이 빠른 저장장치의 등장으로 요구 페이징과 메모리 사상 입출력에서 활용되는 페이지 폴트의 수행시간이 대폭 감소될 것으로 예상된다. 하지만 기존 페이지 폴트 처리 방식의 비효율적인 구조로 인해 저장장치의 빠른 응답시간을 충분히 활용하지 못하는 문제점이 있다. 논문에서는 페이지 폴트 처리의 병목 지점을 분석하기 위해 메모리 관리 계층과 입출력 계층에서 호출되는 함수의 수행시간을 수집하는 PFA를 소개한다. PFA를 통해 저장장치의 성능에 따라 페이지 폴트의 병목이 바뀌는 모습을 관찰하고, 인터럽트 방식의 입출력 처리가 페이지 폴트 처리 시간의 새로운 병목이 된다는 것을 관찰하였다.

1. 서 론

현재 저장장치 시장에서는 NAND 기반 SSD보다 훨씬 빠른 응답시간을 제공하는 Z-NAND 기반 Samsung Z-SSD[1]가 차세대 저장장치로 소개되었다. Samsung Z-SSD의 읽기 응답시간은 12us로 NVMe SSD[2]의 읽기 응답 시간인 90us에 비해 약 8배 빨리 입출력을 처리하여 획기적으로 빠른 응답시간을 제공한다. 이러한 저장장치의 응답시간은 향후 기술 개발로 인해 더욱 감소할 것으로 보인다.

저장장치의 빠른 응답시간은 운영체제가 사용자의 메모리 공간에 존재하지 않는 데이터를 저장장치에서 읽어오는 페이지 폴트(Page Fault) 처리시간을 대폭 감소시킨다. 페이지 폴트의 처리 시간 감소는 프로그램 수행 도중에 필요한 데이터를 페이지 폴트를 통해 메모리에 적재하는 요구 페이징(Demand Paging)을 빨리 수행하여 프로그램의 실행 시간을 감소시키며, 페이지 폴트를 기반한 메모리 사상 입출력 (Memory-Mapped I/O) 역시 빨리 동작할 수 있게 한다. 메모리 사상 입출력은 기존의 파일 입출력보다 파일에 빠르게 접근하기 때문에 페이지 폴트에 대한 관심이 높아지고 있다[3].

하지만 느린 저장장치를 가정하여 설계된 운영체제의 비효율적인 구조로 인해 저장장치 성능을 최대한 활용하지 못하는 문제가 있었다[3,4]. 페이지 폴트에서도 동일한 문제가 있음을 관찰하였다. 그림 1은 NAND 기반 NVMe-NAND 저장장치와 DRAM 기반 NVMe-DRAM 저장장치의 읽기 성능과 페이지 폴트 처리 시간을 보이고 있다. 그림 1의 a는 4KB 크기의 임의 읽기일 때 NVMe-DRAM의 초당 입출력 처리 수가 258K로 NVMe-NAND의 10K에 비해 25.8배 빠르다. 하지만 그림 1의 b에서, 페이지 폴트 처리 시간은 NVMe-DRAM의 경우 74us로 NVMe-NAND의 234us에 비해 3.16배만 빠른 모습이다. 이는 페이지 폴트의 비효율적인 구조로 발생한 소프트웨어 오버헤드 때문에 25.8배 빠른 저장장치의 성능이 실제로는 3.16배의 효과만 보인 것이다.



(a) 임의 읽기 성능 (b) 페이지 폴트 처리 시간

그림 1. 저장장치 별 읽기 성능 및 페이지 폴트 처리 시간

이처럼 페이지 폴트 처리시 발생하는 소프트웨어 오버헤드를 분석하기 위해서는 페이지 폴트 처리를 위해 호출되는 각 함수들의 수행시간 정보를 수집할 수 있어야한다. 또한 페이지 폴트 처리가 시작되는 메모리 관리 계층부터 입출력을 요청하고 응답을 받는 입출력 계층까지를 모두 아우르는 계층 통합적 분석 도구가 필요하다. 하지만 기존 페이지 폴트 분석 도구인 perf[5]는 페이지 폴트 처리에 관련된 함수들의 호출 횟수만 수집할 뿐 각 함수들의 수행시간 정보를 확인할 수 없는 한계가 있다.

본 논문에서는 저장장치들의 응답시간이 빨라짐에 따라 페이지 폴트 처리 과정에서 새로운 병목 지점을 분석하기 위해 메모리 관리 계층부터 입출력 계층까지의 동작을 모두 관찰하고 수행시간을 수집하는 도구인 PFA(Page Fault Analyzer)를 소개한다. 또한 PFA를 통해 페이지 폴트를 분석한 결과 저장장치의 응답시간이 DRAM에 가까워질수록 입출력 요청으로 Sleep 상태에 있던 프로세스가 스케줄링을 기다린 시간이 새로운 병목이 된다는 것을 확인하였다.

논문의 구성은 다음과 같다. 2장에서는 PFA의 구조와 동작 원리에 대해서 설명한다. 3장에서는 PFA 도구의 관찰 대상함수들에 대해서 설명한다. 4장에서는 제안한 도구를 활용하여 페이지 폴트 처리의 병목 지점을 분석한 결과를 설명하고, 5장에서 결론을 맺는다.

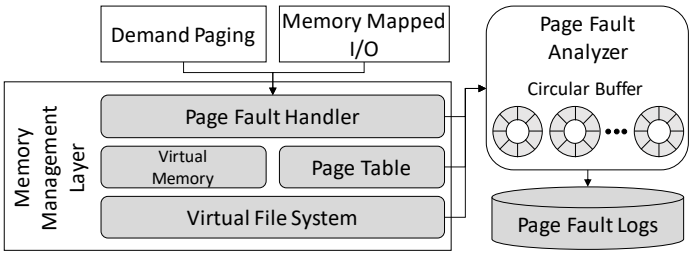


그림 2. PFA의 전체 구조

2. PFA 구조 및 동작 원리

그림 2는 페이지 폴트 분석 도구인 PFA의 전체 구조를 나타내고 있다. PFA는 요구 페이징과 메모리 사상 입출력으로 페이지 폴트가 발생하면 페이지 폴트 처리를 위해 호출되는 함수들의 시작 시간과 종료 시간 그리고 미리 정해놓은 함수의 번호와 페이지 폴트를 일으킨 프로세스 아이디를 순환 버퍼에 저장한다. 수집된 로그 정보들은 PFA가 종료되기 전까지 순환 버퍼에 위치해 있다가 PFA까 종료될 때 함수의 시작 시간을 기준으로 정렬하여 파일로 출력한다. 이처럼 메모리 공간에 위치한 순환 버퍼에 먼저 로그 데이터를 수집한 뒤 수집이 종료된 이후에 데이터를 출력하는 방식은 분석 도구인 PFA가 분석 대상인 페이지 폴트 처리에 미치는 영향을 최소화하기 위함이다. 하지만 순환 버퍼가 지나치게 크게 설정되면 페이지 폴트 처리 시간에 영향을 줄 수 있으므로 최대 80 MB 크기를 할당한다.

PFA는 특정 프로세스로가 발생시킨 페이지 폴트의 로그 정보만을 수집할 수 있게 하기 위해 Sysfs로 관찰 대상의 프로세스 아이디를 먼저 전달 받는다. 이후 Sysfs로 PFA을 시작 시키면 PFA는 페이지 폴트가 발생하였을 때 현재 프로세스의 아이디를 확인하여 관찰 대상 프로세스 인지를 판단한다. 만약 관찰 대상 프로세스의 아이디를 미리 설정하지 않았다면 모든 프로세스의 페이지 폴트 로그 정보를 저장하되 프로세스 아이디를 함께 수집하여 사용자가 직접 특정 프로세스의 로그를 선별 할 수 있다.

입출력 계층에서의 로그 수집을 위해서, 스마트폰의 저장장치 입출력 분석 도구인 AIOPro[6]를 PC환경으로 확장한 버전을 사용하였다. 확장된 AIOPro를 이용하여 블록 입출력 계층과 NVMe 디바이스 드라이버에서 호출되는 함수들의 수행시간을 수집할 수 있어 메모리 관리 계층과 입출력 계층 모두에 대해 페이지 폴트 처리 분석이 가능하다.

3. PFA의 관찰 대상 함수

PFA를 이용하여 페이지 폴트 처리의 병목을 분석하기 위해서는 관찰 대상 함수들이 페이지 폴트 처리 과정에서 어떤 역할을 수행하는지를 파악해야한다. 그림 3은 페이지 폴트 처리 과정과 호출되는 함수들을 나타내고 있다.

페이지 폴트가 발생하면 `Trap`을 통해 `do_page_fault`가 호출되고 페이지 폴트를 발생시킨 메모리 접근이 비정상적인 메모리 공간으로의 접근인지를 확인한다. 이후 `handle_mm_fault`에서는 페이지 폴트를 발생시킨 페이지가 페이지 캐시에 존재하는지를 확인한다. 만약 존재 한다면 입출력 필요 없는 Minor Fault가 된다. 하지만 페이지 캐시에 존재하지 않는다면 입출력을 통해 페이지 데이터를 읽어오는 Major Fault로 처리된다. Major

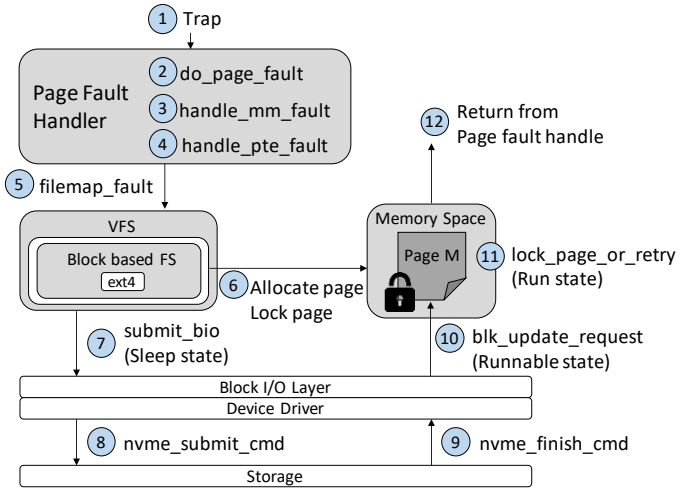


그림 3. 페이지 폴트 처리 과정

Fault로 처리되면 `handle_pte_fault`를 통해 페이지 폴트를 일으킨 메모리 주소에 대한 페이지 테이블 처리가 이루어진다. `filemap_fault`에서는 본격적으로 입출력을 통해 페이지 데이터를 메모리 공간으로 읽어오는 작업을 수행한다. 먼저 저장장치로부터 데이터를 읽기 위해 메모리 공간에서 페이지를 할당 받고 페이지에 Lock을 설정한다. 이후 페이지 폴트가 발생한 메모리 공간에 맵핑 되어 있는 파일 시스템 함수를 통해 `submit_bio`를 호출하여 입출력 요청을 하고 Lock을 걸었던 페이지의 Lock이 풀릴 때까지 Sleep 상태로 기다리게 된다. 입출력 처리를 위해 디바이스 드라이버가 `nvme_submit_cmd`로 저장장치에 읽기 요청을 전달하고, 저장장치의 입출력이 완료되어 인터럽트가 발생하면 `nvme_finish_cmd`와 `blk_update_request`로 페이지에 설정된 Lock을 해제한 뒤, 프로세스를 Runnable 상태로 바꾼다. 이후 스케줄링으로 인해 해당 프로세스가 선택되어 Run 상태가 되면 `lock_page_or_retry`로 읽어온 페이지가 교체되지 않았는지를 확인하고 페이지 폴트 처리가 완료된다.

4. PFA를 이용한 페이지 폴트 분석

본 실험의 목표는 저장장치의 응답시간이 빨라짐에 따라 페이지 폴트 처리의 병목 지점을 PFA를 통해 분석하는 것이다. 실험 환경으로 CPU E6550 Dual Core 2.33GHz와 8GB 메인 메모리를 사용하였고 리눅스 커널 버전 3.19에 PFA를 구현하였다. 저장장치로는 Samsung PM1725 모델의 NVMe- NAND[2]와 Taejin InfoTech에서 제작한 NVMe-DRAM을 사용한다. NVMe-DRAM은 호스트가 NVMe 인터페이스를 통해 디바이스에 부착되어 있는 DRAM에 접근하므로 그림 1에서 보이고 있듯이 매우 높은 읽기 성능을 가진다. 페이지 폴트 발생 시나리오는 1GB 크기의 실행 파일을 생성한 후, 가상 메모리 영역의 뒤부터 임의로 접근하여 요구 페이징으로 인한 페이지 폴트가 발생하도록 하였다. PFA로 수집한 함수들의 실행시간 정보를 명료하게 나타내기 위해, 표 1처럼 관찰 대상을 4개의 영역으로 나누어 결과로 표현한다.

그림 4는 NVMe-NAND 저장장치를 사용하였을 때 페이지 폴트 처리의 4개의 영역을 타임라인으로 나타내고 있으며, 각 영역이

표 1. 페이지 폴트 처리의 관찰 대상 영역

영역 이름	구간
페이지 폴트 전처리	do_page_fault ~ submit_bio
블록 입출력 처리	submit_bio ~ nvme_submit_cmd
디바이스 처리	nvme_submit_cmd ~ nvme_finish_cmd
페이지 폴트 후처리	nvme_finish_cmd ~ return_fault

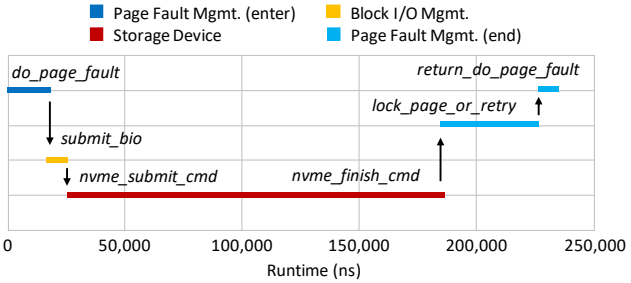


그림 4. NVMe-NAND에서의 페이지 폴트 처리 타임라인

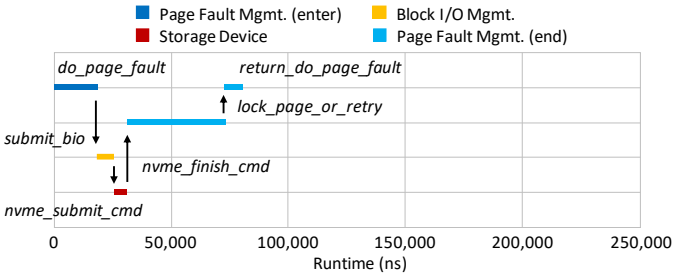


그림 5. NVMe-DRAM에서의 페이지 폴트 처리 타임라인

시작될 때 호출되는 함수들을 표시하였다. 분석 결과 NVMe-NAND의 디바이스 처리 시간이 전체 페이지 폴트 처리 시간의 대부분을 차지하는 것으로 나타났다. 이에 반해 소프트웨어 오버헤드인 페이지 폴트 전처리 및 후처리 시간과 블록 입출력 시간은 상대적으로 작은 부분을 차지하고 있다.

그림 5는 동일한 실험을 NVMe-DRAM을 대상으로 진행한 결과이다. 읽기 성능이 매우 빠른 NVMe-DRAM의 경우 디바이스 처리 시간이 대폭 감소하였다. 하지만 페이지 폴트 처리와 블록 입출력 처리 시간을 포함하는 소프트웨어 오버헤드가 전체 페이지 폴트 처리 시간의 대부분을 차지하게 되었고, 그 중 입출력의 응답 이후에 스케줄링을 기다린 시간인 페이 폴트 후처리 시간이 상당 부분을 차지하고 있다.

그림 6은 NVMe-NAND와 NVMe-DRAM 저장장치에 대해 페이지 폴트 처리의 영역별 수행시간 분포를 나타내고 있다. NVMe-NAND의 경우 디바이스 처리 시간이 약 68%로 전체 수행시간의 대부분을 차지하지만 NVMe-DRAM에서는 디바이스 처리 시간이 대폭 감소하여 약 7%를 차지하고 있다. 따라서 NVMe-DRAM과 같이 응답시간 빠른 저장장치에서는 더 이상 디바이스 처리 시간이 페이지 폴트 수행시간의 병목이 되지 않는다. 반면 인터럽트 방식의 입출력 처리로 인해 프로세스가 스케줄링 받기 위해 기다린 시간인 페이지 폴트 후처리 시간이 61%를 차지하여 새로운 병목으로 나타난 것을 PFA로 확인 할 수 있다.

이처럼 페이지 폴트 처리의 비효율적인 구조를 개선하기 위해 인터럽트 방식에서 폴링 방식으로 입출력을 처리하는 연구[4]와 페이지 테이블 관리를 효과적으로 수행하여 페이지 폴트 처리

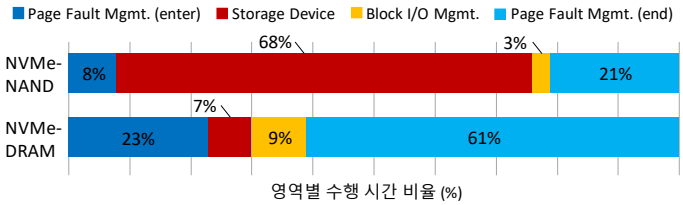


그림 6. 영역별 수행시간 비율 분포

시간을 줄인 연구[3]가 있었다. 향후 더 빠른 응답시간을 제공하는 저장장치들이 개발됨에 따라 페이지 폴트의 병목 분석이 더욱 중요해질 것이므로 본 논문에서 제안한 PFA가 페이지 폴트 최적화 연구에 많은 기여를 할 것으로 예상된다.

5. 결 론

본 논문에서는 기존 운영체제의 비효율적인 페이지 폴트 처리로 인해서 응답시간이 빠른 차세대 저장장치의 성능을 최대한 활용하지 못하는 문제가 있음을 지적하고 페이지 폴트 처리 과정에서 발생하는 소프트웨어 오버헤드를 분석하기 위한 도구인 PFA를 제안하였다. 제안한 도구를 통해 저장장치의 성능이 DRAM에 가까워 질수록 인터럽트 방식의 입출력 처리가 병목이 된다는 것을 확인하였으며, 향후 응답시간이 빠른 저장장치를 위한 페이지 폴트 연구에 적극 활용될 수 있을 것으로 기대한다.

6. 감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다. 이 논문은 한국연구재단의 재원으로 서울대학교 컴퓨터공학부 BK21플러스 컴퓨터미래인재양성사업단의 지원을 받아 수행된 연구임(21A20151113068). 이 논문은 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015M3C4A7065645). (교신저자: 김지홍)

참 고 문 헌

[1] Samsung Z-NAND SSD. http://samsung.com/us/labs/pdfs/collateral/Samsung_Z-NAND_Technology_Brief_v5.pdf.

[2] Samsung SSD PM1725. http://samsung.com/semiconductor/global/file/insight/2016/08/Samsung_PM1725a-1.pdf.

[3] J. Choi et al., “Efficient Memory Mapped File I/O for In-Memory File Systems,” in *Proceedings of the USENIX Workshop on Hot Topics in Storage and File System*, 2017.

[4] J. Yang et al., “When Poll is Better than Interrupt,” in *Proceedings of the USENIX Conference on File and Storage Technologies*, Vol. 12, 2012.

[5] perf-state. https://linux.die.net/ man/1/perf-stat.

[6] 한상욱, 이인혁, 류동욱, 김지홍, “AIOPro: 안로이드 스마트폰을 위한 통합된 스토리지 I/O 분석도구”, 한국정보과학회 논문지, 제44권, 제3호, pp. 232~238, 2017.