SSD 에서의 리눅스 readahead 성능 분석

윤명식º 주용수 임성수 임은진

국민대학교 소프트웨어융합대학

myoungsikyoon@kookmin.ac.krº,
ysjoo@kookmin.ac.kr, sslim@kookmim.ac.kr, ejim@kookmin.ac.kr

Analysis of Linux readahead technique on SSD

Myoungsik Yoonº, Yongsoo Joo, Sung-Soo Lim, Eun-Jin Im
School of Computer Science Kookmin University

요 약

리눅스에서는 파일 읽기 작업의 최적화를 위해 prefetching 기술 중 하나인 readahead 기법을 채택하고 있다. Readahead 기법은 파일 읽기 요청이 발생했을 때 이후에 발생할 파일 읽기 요청들을 예측하여 해당 데이터들을 prefetching 하는 기법이고, 미래의 파일 읽기 요청을 예측하기 수월한 순차 파일 읽기 요청에서 효과가 극대화 된다. 이러한 Readahead 기법은 HDD의 물리적인 특성과 맞물려 그 효과가 더욱 상승하였다. 한편, 최근 NAND flash 기반 SSD가 널리 보급됨에 따라 저장 장치의 특성이 크게 바뀌게 되었다. 그럼에도 불구하고 SSD에서의 readahead 성능에 대한 분석은 충분히 이루어지지 않고 있다. 이에 본 논문에서는 현재 리눅스에서 사용되고 있는 readahead 기법을 SATA 기반 SSD에서 분석하고 readahead window의 크기를 변화시키며 실제 성능을 평가한다.

1. 서 론

파일 입출력 처리 성능은 애플리케이션 실행 속도의 상당 부분을 차지하는 요소이며 이러한 중요도로 인해 파일 입출력 성능 최적화에 많은 연구가 수행되어 왔다. 파일 입출력의 유형은 크게 파일 쓰기와 파일 읽기로 나뉘며 이 중 파일 읽기 작업 최적화의 경우 prefetching 기술이 주로 논의되어 왔다.

이 prefetching 기술 중 하나인 readahead 기법[1]은 파일 읽기 요청이 발생했을 때 이후에 발생할 파일 읽기 요청들을 예측하고, 해당 데이터들을 prefetching하여 page cache hit 비율을 증가시키는 기법으로 리눅스는 이 readahead 기법을 채택하고 있다. Readahead 기법은 미래의 파일 읽기 요청을 예측하기 수월한 순차 파일 읽기 요청에서 그 효과가 극대화 된다. 이러한 readahead 기법은 HDD의 물리적인 특성과 맞물려 그 효과가 더욱 상승하였다.

이러한 리눅스 readahead 기법은 2010년경 있었던 demand-readahead 알고리즘의 업데이트가 마지막으로 큰 변화 없이 현재까지 사용되고 있다.

한편, 파일 입출력 성능의 개선을 위해 하드웨어적으로도 발전이 있었다. 과거에는 디스크 방식의 저장 장치인 HDD가 주로 사용되었는데 최근에는 NAND flash 기반 SSD, NVME, 3D-Xpoint SSD 등의 저장 장치가 널리 보급되어 파일 입출력 성능이 HDD에 비해 비약적으로 향상되었고 저장 장치의 작동 원리도 크게 바뀌어 HDD의 물리적 한계를 벗어날 수 있었다.

하지만 demand-readahead 알고리즘이 도입된 2010년경에는 HDD가 주로 사용되었기 때문에 저장 장치를 SSD로 사용하는 환경에 대한 분석이 충분히 이루어지지 않았다. 실제로 한 연구[2]에는 “SSD 스토리지에서 전체 성능을 얻는데 필요한 최적의 파일 입출력 크기는 HDD와 다르며 장치마다 다르다”고 언급되어 있으나 이에 대한 정량적인 분석을 수행한 연구는 알려져 있지 않다.

이에 본 논문에서는 저장 장치를 SSD로 사용하는 리눅스 시스템에서 readahead의 구조를 분석하고 readahead의 성능이 최고가 되는 최적의 파일 입출력 크기 탐색을 위해 readahead window 크기를 변경해가며 readahead의 실제 성능을 측정 및 평가한다.

* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2018R1D1A1B05044558).

2. Readahead 구조 분석

2.1 리눅스 파일 입출력 흐름

리눅스 시스템에서 파일 읽기 요청의 처리 과정은 그림 1과 같다. 애플리케이션이 특정 파일의 offset에 대해 파일 읽기 요청을 발생시키면 우선 page cache에 요청한 내용이 캐싱 되어 있는지의 여부를 확인한다. 이때 page cache에 해당 내용이 캐싱되어 있는 경우인 page cache hit과 캐싱되어 있지 않은 경우인 page cache fault 두

가지로 나뉘며 파일 읽기 요청의 이후 처리 방법이 각각 다르다. 첫째로 만약 page cache hit이 발생한다면 CPU cache에 해당 내용을 캐싱하여 빠르게 파일 읽기 요청을 종료한다. 둘째로, 만약 page cache fault가 발생한다면 해당 요청의 file level 정보를 기반으로 Block I/O의 내용을 담은 BIO 구조체를 구성하고 이 BIO 구조체를 통해 저장 장치에 파일 입출력을 요청한다. 이후 이 요청의 응답으로 받은 데이터를 page cache와 CPU cache에 캐싱하고 파일 읽기 요청을 종료한다.

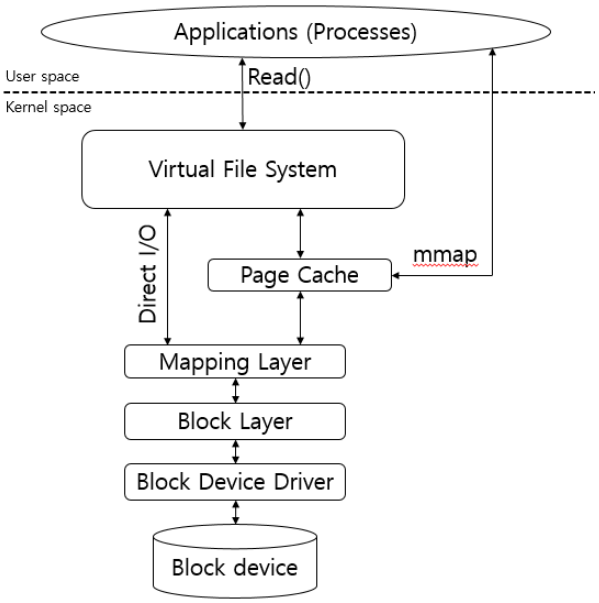

그림 1 리눅스 파일 입출력 흐름

2.2 리눅스 v2.6 이전의 readahead 구조

리눅스 v2.6 이전의 readahead 기법에서는 파이프라이닝을 수행하기 위해 그림 2와 같이 두 개의 window를 관리하였다. 이 두 개의 window는 current window와 ahead window인데 응용 프로그램이 current window에서 진행되는 동안 readahead의 파일 읽기 작업은 ahead window에서 비동기적으로 진행된다. 파일 읽기 요청이 ahead window로 넘어갈 때마다 current window가 되고, 새로운 ahead window를 만든 후 readahead가 이를 비동기적으로 읽는다. Readahead 파이프라이닝은 모두 비동기 readahead를 수행하는 것이다. 하지만 이 방법은 다음 구간의 readahead를 얼마나 일찍 비동기적으로 시작해야 하는지에 대해 정확한 답을 제시할 수 없어 파일 읽기 작업의 “비동기 정도”를 제어할 수 없는 한계를 가진다.

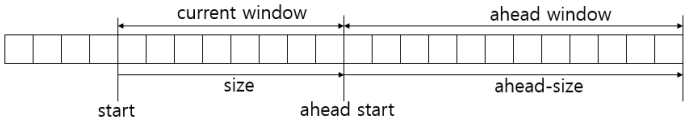

그림 2 current window & ahead window

2.3 리눅스 v2.6 이후의 readahead 구조

리눅스 v2.6 이후에서는 이전 readahead의 한계점을 보완하기 위해 demand-readahead[2] 알고리즘이 도입되었다. Demand-readahead 알고리즘에서는 파이프라이닝을 수행하기 위해 그림 3과 같이 한 개의 window만을 관리한다. 또한, "비동기 정도"를 제어하기 위해 명시적 매개변수인 async_size를 추가하였다. 아직 소비되지 않은 readahead 페이지 수가 이 임계값 아래로 떨어지면 바로 다음 readahead를 시작한다. 이를 통해 한 개의 window를 사용하여 readahead를 관리할 수 있게 되었다. 이 한 개의 readahead window 안에 start + size - async_size 위치에 있는 페이지에 PG_readahead flag를 추가하여 파일 읽기 작업의 "비동기 정도"를 제어한다. 이러한 리눅스 readahead 기법은 2010년경 있었던 demand-readahead 알고리즘의 개선 이후 큰 변화 없이 현재까지 사용되고 있다.

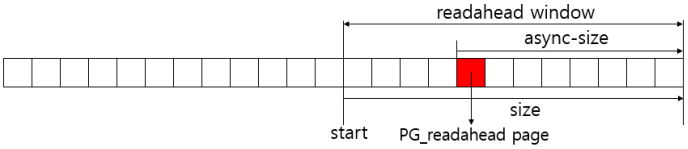

그림 3 readahead window & PG_readahead page

3. Readahead 크기에 따른 파일 읽기 성능

파일 입출력 성능을 향상시키기 위해 저장 장치가 디스크 기반의 저장 장치인 HDD에서 최근 NAND flash 기반 SSD, NVME, 3D-Xpoint SSD 등의 저장 장치로 발전하였다. 이러한 저장 장치들이 널리 보급되기 시작하면서 파일 입출력 성능이 과거에 비해 비약적으로 향상되었고 저장 장치의 작동 원리도 크게 바뀌었다. 이에 따라 HDD의 물리적 특성과 한계를 고려할 필요성이 줄어들게 되었다.

하지만 demand-readahead 알고리즘이 도입된 2010년경에는 HDD가 주로 사용되었기 때문에 저장 장치를 SSD로 사용하는 환경에서 readahead 기법의 효율에 대한 분석이 충분히 이루어지지 않았다. 한 연구[2]에서 “SSD 스토리지에서 전체 성능을 얻는데 필요한 최적의 I/O 크기는 HDD와 다르며 장치마다 다르다”고 언급되어 있으나 이에 대한 정량적인 분석을 수행한 연구는 알려져 있지 않다.

현재의 리눅스 커널에서 readahead 기법의 주요 변인은 file_ra_state 구조체로 관리된다. 이 중 ra_pages 변수는 readahead window 크기가 도달할 수 있는 최대 크기를 저장하는 변수이며 이를 통해 readahead 기법에서 prefetch하는 데이터의 양을 조절할 수 있다. 리눅스 시스템에서 한 개 페이지에 대한 파일 입출력이 발생했을 때, readahead 기법에 의해 4KB 크기의 파일 입출력이 발생하는 것이 아닌 128KB(32*4KB) 크기의 파일 입출력이 발생하게 된다. 그 이유는 일반적인 리눅스 시스템에서 ra_pages 변수의 기본값이 32인 상수이기 때문이다. 즉, readahead window 크기의 결정자는 ra_pages 변수이다. 이 변수의 기본값을 정할 때 휴리스틱이 적용되었으며 그 결과 기본값이 32가 되었다. 하지만 이를 정했을 당시에는 주로 HDD가 저장 장치로 사용되었기 때문에 이 값이 최근의 NAND flash 기반 SSD에서도 최적의 값 인지에 대하여 살펴볼 필요가 있다. 이에 따라 본 논문에서는 저장 장치를 SSD로 사용하는 리눅스 환경에서 readahead의 성능이 최고가 되는 최적의 파일 입출력 크기 탐색을 위해 readahead window 크기를 변경해가며 readahead의 실제 성능을 측정 및 평가한다.

4. 실험 구성 및 결과

4.1 실험 구성

본 논문에서는 SSD를 저장 장치로 사용하는 리눅스 시스템에서 readahead window 크기를 4KB~384KB 사이로 변화시키며 최적의 readahead window 크기의 값을 탐색한다.

표 1

OS	Ubuntu 16.04.6 LTS
Kernel	5.3.0-rc8
CPU	Intel i7-8086K (4.00GHz)
RAM	32G
저장 장치	삼성전자 860 PRO (2T)

4.2 실험 결과와 분석

4.2.1 APP launch time 관점

Readahead window 크기를 4KB~384KB까지 변화시키며 5개의 앱에 대해 구동 시간을 측정한 결과는 그림 4와 같다. Readahead window 크기가 16KB 미만일 때 모든 앱들의 구동 시간이 최대 약 1.35배 증가하며 16KB 이상일 때에는 구동 시간에 큰 변화가 없다. 이는 적은 readahead window 크기로 인하여 page cache hit rate가 낮아져 파일 입출력 작업이 저장 장치까지 접근하는 횟수가 늘어났기 때문이다.

그림 4 APP launch time

4.2.2 page cache hit ratio 관점

Readahead window 크기를 4KB~384KB까지 변화시키며 5개의 앱에 대해 page cache hit ratio를 측정한 결과는 그림 5와 같다. Readahead window 크기가 작아질수록 page cache hit ratio 또한 감소한다. Readahead window 크기가 16KB 미만일 때 page cache hit ratio가 급격히 줄어드는데 이는 readahead window 크기가 작아졌기 때문에 추가로 가져오는 데이터의 양이 너무 적어졌기 때문이다.

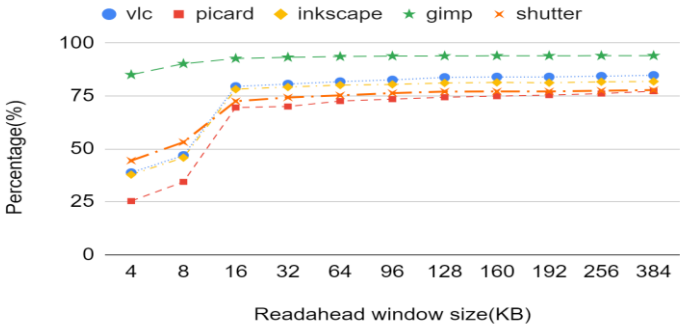

그림 5 Page cache hit ratio

4.2.3 readahead hit ratio 관점

Readahead window 크기를 4KB~384KB까지 변화시키며 5개의 앱에 대해 readahead hit ratio를 측정한 결과는 그림 6과 같다. Readahead hit ratio란 리눅스 시스템에서 특정 파일의 offset에 파일 입출력이 발생했을 때 page cache에 캐싱 되는 page들 중 해당 offset의 정보가 포함되어 있는 page를 제외한 나머지 page들의 hit ratio를 말한다. 즉, readahead 기법에 의해 반입된 데이터들 중 실제로 사용된 데이터의 비율을 나타내는 것이다. Readahead window 크기가 16KB 미만일 때에는 비슷한 비율을 보이다가 16KB 이상이 되면 꾸준히 감소한다.

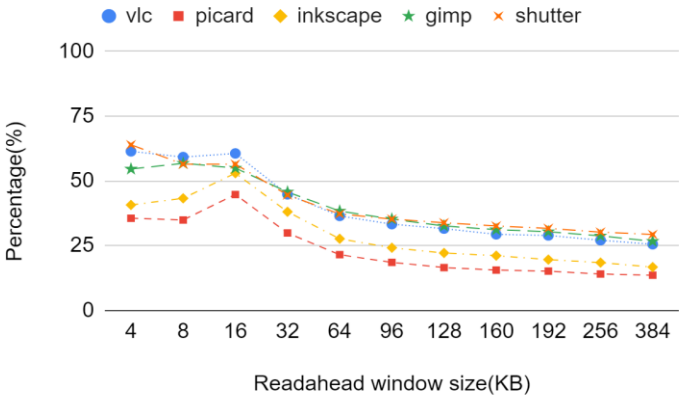

그림 6 Readahead hit ratio

5. 결론 및 향후 계획

SSD 저장 장치를 사용하는 리눅스 시스템에서 readahead는 readahead window 크기를 최소 16KB 이상으로 설정해주면 파일 입출력 성능을 크게 저하시키지 않는 선에서 시스템 운용이 가능하다. 또한, 앱 마다 최적의 readahead window 크기에 차이가 있음을 확인할 수 있었다. 이를 통해 현재 리눅스에서 사용하고 있는 readahead window 크기의 기본값 128KB는 전체 시스템 상에서 보았을 때 최적에 근접하긴 하지만 앱 단위로 그 범위를 좁혔을 때에는 최적이 아니라는 것을 확인하였다.

향후 계획으로는 리눅스 파일 입출력 과정을 일반 파일 입출력과 메모리 매핑 기반 입출력으로 세분화하여 분석하고, HDD와 비교 분석할 계획이다. 또한, readahead window 크기는 파일마다 한 개의 file_ra_state 구조체를 관리하고 있는데 이를 앱 별로 따로 관리하게 하여 각기 다른 앱들이 모두 자신에게 최적화된 readahead window 크기를 구성할 수 있도록 file_ra_state 구조체를 관리하게 할 계획이다.

6. 참고 문헌

[1] Fengguang, Wu Hongsheng Xi, Jun Li, Nanhai Zou, Linux readahead: less tricks for more, Proceedings of the Linux Symposium, Volume Two, 273-286p, 2007

[2] Fengguang Wu, Sequential File Prefetching in Linux, Advanced Operating Systems and Kernel Applications, 218-261p, 2010