리눅스 I/O Throttling 기법에 의한 사용자 쓰기 요청의 지연시간 증가 현상 분석

오준택○ 원유집
한양대학교 컴퓨터·소프트웨학과
na94jun@hanyang.ac.kr, yjwon@hanyang.ac.kr

The Effect of Linux I/O Throttling on User Write Latency

Joontaek Oh○ Youjip Won
Department of Computer and Software, Hanyang University

요 약

리눅스는 dirty 페이지가 생성되는 속도와 dirty 페이지가 처리되는 속도를 맞추기 위해 I/O Throttling 이라는 기법을 사용한다. dirty 페이지가 처리되는 속도는 dirty 페이지가 생성되는 속도보다 상대적으로 느리다. 이러한 특징으로 인해 페이지 캐시 내의 free 페이지가 고갈되는 문제가 발생할 수 있다. 이를 방지하기 위해 리눅스는 dirty 페이지를 생성하는 프로세스가 sleep 하도록 하여 dirty 페이지 생성 속도를 늦추는 I/O Throttling이라는 기법을 사용한다. 하지만 I/O Throttling 기법으로 인해 write() 시스템 콜의 반환 시간이 느려질 수 있다. 본 논문에서는 I/O Throttling 기법으로 인해 write() 시스템 콜이 느려지는 경우를 분석하고, 향후 해결법을 논의한다.

1. 서 론

HDD(Hard disk drive), SSD(Solid State Drive)와 같은 스토리지의 입출력 속도는 주기억 장치인 DRAM의 입출력 속도보다 느리다. 리눅스는 이러한 입출력 장치 별 속도 차이를 응용 프로그램으로부터 감추기 위해, 사용자 프로세스가 O_DIRECT나 O_SYNC 옵션 없이 쓰기를 수행할 때 스토리지에 데이터를 직접 쓰지 않고 DRAM에서 관리되는 페이지 캐시에 쓰기를 수행하고 반환하는 Buffered 쓰기 방식을 사용한다[1].

사용자 프로세스에 의해 내용이 쓰여져서 파일의 내용과 달라진 페이지를 dirty 페이지라고 한다. dirty 페이지는 스토리지에 쓰여져서 clean 페이지가 되어야 해제될 수 있는 페이지이다. 페이지 캐시내의 dirty 페이지들은 kworker라는 쓰레드에 의해 스토리지에 쓰여진다. kworker 쓰레드는 sleep 상태에 있다가 주기적으로 깨어나서 dirty 페이지들을 스토리지에 쓴다.

이 때, 사용자 프로세스가 수행하는 입출력은 페이지 캐시, 즉, 메인 메모리에 대한 입출력이고, kworker가 수행하는 입출력은 스토리지에 대한 입출력이다. 응용 프로그램이 최고 속도로 쓰기 연산을 수행할 경우, dirty 페이지가 생성되는 속도는 dirty 페이지가 스토리지에 쓰여지는 속도보다 빠르다. 다수의 사용자 프로세스가 각각 다른 파일의 dirty 페이지를 많이 생성하게 되면 사용자 프로세스와 kworker의 작업 속도 차이가 생기면서 페이지 캐시에서 free 페이지가 고갈되고, free 페이지 확보를 위해 Foreground Writeback을 수행한다. 이 때, 다수의 Foreground Writeback을 수행하면서 스핀 락 경쟁이 발생한다.

이를 방지하기 위해서 리눅스는 I/O Throttling 기법을 사용한다. I/O Throttling 동작은 write() 시스템 콜 동작 도중에 호출되며, dirty 페이지를 생성하는 사용자 프로세스가 sleep하도록 하는 작업을 수행한다. 따라서 I/O Throttling 동작은 write() 시스템 콜의 응답 지연을 악화시킬 수 있다.

본 논문에서는 I/O Throttling으로 인해서 생긴 write() 성능의 하락을 분석한다. 먼저 write() 시스템 콜과 I/O Throttling의 동작에 대해서 분석한 후, write() 시스템 콜을 사용하여 파일에 쓰기를 수행하는 실험을 진행한 결과를 통해 I/O Throttling이 write() 시스템 콜 성능 하락에 미치는 영향을 확인한다.

2. 배 경

2.1. 리눅스 write() 시스템 콜 동작 분석

```
function generic_perform_write(filep, data, size)
  for i ← 0 to size step 4096 do
    page ← 페이지 캐시로부터 페이지 할당;
    offset ← 페이지 내 오프셋;
    memcpy(page+offset, data, size);
    page를 dirty 페이지로 변경;
    balance_dirty_pages_ratelimited();
  end for
end function
```

의사 코드 1. generic_perform_write()함수의 동작

write() 시스템 콜은 open() 시스템 콜로 열기를 한 파일에 쓰기를 하는 시스템 콜이다. write() 시스템 콜은 일

반적인 Buffered 쓰기일 때, 커널 함수인 generic_perform_write()를 호출한다. generic_perform_write()는 할당해야 하는 페이지의 수만큼 반복문을 수행하며, 페이지에 쓰기를 수행한다.

먼저, 쓰기를 수행할 페이지를 할당한다. 이 때, 파일 시스템 함수인 write_begin() 함수를 호출하는데, write_begin() 함수는 파일 포인터와 오프셋 등을 인자로 받아서 쓰기를 할 페이지를 반환한다. write_begin() 함수는 먼저 페이지 캐시에서 쓰기를 수행할 페이지를 찾는다. 쓰기를 수행할 페이지가 캐싱되어 있으면 해당 페이지를 반환하고, 캐싱되어 있지 않으면 페이지 캐시에서 free 페이지 할당 후, 파일의 내용을 페이지에 쓰고, 페이지를 반환한다.

다음에는 write_begin() 함수가 반환한 페이지에 쓰기를 하는데, iov_iter_copy_from_user_atomic() 함수를 사용한다. iov_iter_copy_from_user_atomic() 함수는 내부적으로 memcpy()를 사용함으로써 페이지에 대한 쓰기를 수행한다.

쓰기 동작의 마지막으로, 쓰기를 완료한 페이지를 dirty 페이지로 변경한다. 이 동작은 파일 시스템 함수인 write_end() 함수를 사용한다.

위 과정을 write() 함수로 쓰고자 하는 데이터를 전부 쓸 때까지 반복한다. dirty 페이지 하나를 생성할 때마다 balance_dirty_pages_ratelimited() 함수를 호출한다. balance_dirty_pages_ratelimited() 함수는 시스템의 dirty 상태를 확인하고, I/O Throttling을 수행하는 함수이다.

위와 같이 write() 시스템 콜이 Buffered 쓰기를 수행한다. I/O Throttling이 없을 때는 페이지 캐시에서 페이지를 할당하는 비용과 메모리에 쓰는 비용, 그리고 페이지를 dirty 페이지로 만드는 비용만 있을 것이다. 하지만 dirty 페이지의 양이 일정 량을 넘어서게 되면 balance_dirty_pages_ratelimited() 함수에서 일정 시간동안 sleep을 하여 write() 시스템 콜의 성능이 하락한다. 2.2절에서는 이러한 I/O Throttling을 수행하는 함수인 balance_dirty_pages_ratelimited() 함수에 대해서 분석해 볼 것이다.

2.2. I/O Throttling 동작 분석

I/O Throttling은 페이지 하나에 대한 쓰기를 수행할 때마다 dirty 페이지의 개수를 확인하고, dirty 페이지의 개수가 일정량보다 많다면 해당 프로세스를 sleep하게 하여 dirty 페이지의 생성 속도를 늦추는 기법이다.

먼저 Buffered 쓰기를 하는 프로세스는 페이지 하나에 대한 쓰기를 완료할 때마다 balance_dirty_pages_ratelimited() 함수를 호출한다. balance_dirty_pages_ratelimited() 함수는 현재 프로세스가 갖고 있는 dirty 페이지의 양을 확인한다. 만약 현재 프로세스가 갖고 있는 dirty 페이지의 양이 일정량을 넘어서게 되면 balance_dirty_pages() 함수를 호출한다.

balance_dirty_pages() 함수는 먼저 전체 시스템에서의 dirty 페이지의 양과 페이지를 쓰려고 했던 스토리지에서의 dirty 페이지의 양을 확인한다. 전체 시스템에서의 dirty 페이지의 양은 많지만 해당 스토리지에서의 dirty 페이지의 양은 적을 수 있고, 전체 시스템에서의 dirty 페이지의 양은 적지만 스토리지에 dirty 페이지가 몰려있을 수 있기 때문이다. 확인한 전체 시스템에서의 dirty 페이지의 양과 스토리지에서의 dirty 페이지의 양을 이용해서 얼마만큼 sleep을 할지를 pause라는 변수에 저장한다. pause 값은 최대 200ms, 최소 20ms로 고정되며, 최대값을 넘을 경우에는 최대값으로 고정되고, 최소값을 넘지 않을 경우에는 I/O Throttling 동작을 종료한다. pause 값 계산이 완료된 후, 프로세스는 계산된 pause값만큼 sleep한다. sleep이 끝난 후, 위 동작을 반복한다.

```
function balance_dirty_pages()
  for true do
    global dirty 페이지의 양 확인;
    writeback 쓰레드가 sleep 상태라면 깨움;
    bdi dirty 페이지의 양 확인;
    두 dirty page의 양을 확인하여 pause 값 계산;
    if (200 < pause)
      pause ← 200;
    else if (pause < 10)
      break;
    sleep(pause);
  end for
end function
```

의사 코드 2. balance_dirty_pages()의 동작

위 동작을 수행함으로써 dirty 페이지를 생성하던 사용자 프로세스는 sleep 상태에 들어가 dirty 페이지를 생성하지 않게 되고, dirty 페이지를 스토리지에 쓰는 kworker는 dirty 페이지의 양을 줄인다. 하지만, 사용자 프로세스가 write() 시스템 콜 실행 중 sleep 상태로 전환되기 때문에, write() 시스템 콜의 성능이 하락된다. 본 논문에서는 I/O Throttling이 write() 시스템 콜의 성능에 끼치는 영향을 확인하기 위해 I/O Throttling을 수행하는 커널과 그렇지 않도록 수정된 커널에서 실험을 진행하였고, 이에 대해 분석을 하였다.

3. I/O Throttling으로 인한 성능저하 확인 및 분석

CPU	Intel(R) i7-4770 (3.4GHz)
DRAM	8 Gbyte (4 Gbyte * 2)
운영체제	Ubuntu 14.04 LTS (3.18.1)
스토리지	Samsung 840 Evo SSD 120 Gbyte

표 1 실험 환경

I/O Throttling이 write()에 끼치는 영향을 확인하기 위해 실험을 진행하였다. 먼저 120 Gbyte SSD 파티션을 EXT4[2]로 포맷한 후, 50 Gbyte Cold 파일과 30 Gbyte Hot 파일을 생성하고, Hot 파일에 70 Gbyte만큼 Buffered 임의 쓰기를 하고, Buffered 임의 쓰기의 총 실행 시간을

측정하였다.

[그림 1]에서 EXT4는 I/O Throttling 기법이 적용되어 있는 기존의 커널을 사용한 결과이고, EXT4_mod는 I/O Throttling 기법을 사용하지 않도록 수정한 커널을 사용한 결과이다.

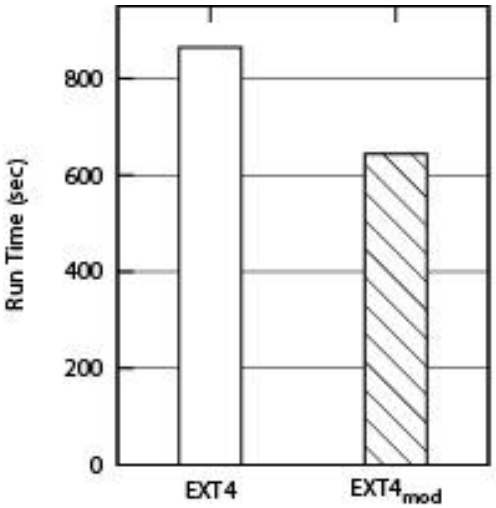

그림 1. I/O Throttling 활성 유무에 따른 성능 차이

I/O Throttling 기법이 적용된 커널을 사용한 EXT4는 실행 시간이 약 864초가 걸렸고, I/O Throttling을 하지 않도록 수정된 커널을 적용한 EXT4_mod는 그보다 약 34% 감소한 644.28초가 걸렸으며, 약 220초의 차이가 있었다. 메인 메모리가 8 Gbyte로 적지 않은 용량이었기 때문에 스핀 락 경쟁 상태에 빠지지는 않았다. 하지만 EXT4에서 I/O Throttling에 대한 회피를 중요시한 나머지 쓰기 성능이 하락하는 문제가 생겼다.

또, I/O Throttling은 dirty 페이지의 수를 줄이기 위해 다수의 dirty 페이지가 줄을 때까지 사용자 프로세스가 sleep 하도록 한다. 하지만 kworker가 dirty 페이지를 스토리지에 쓰는 양보다 적은 양의 dirty 페이지가 생성되도록 사용자 프로세스를 적은 시간씩만 sleep시켜도 dirty 페이지의 양은 시간이 지나면서 줄어들 것이다. 이와 같은 I/O Throttling의 개선 사항에 대해서는 향후 과제로 남기고 다뤄보도록 한다.

4. 결론

본 논문에서는 Buffered 쓰기 동작 도중 사용자 프로세스를 sleep하게 하는 I/O Throttling 기법에 대해서 분석해보고, I/O Throttling 기법이 Buffered 쓰기 동작에 끼치는 영향을 실험을 통해 알아보았다. 실험 결과, 70 Gbyte Buffered 임의 쓰기를 할 때, I/O Throttling 기법을 사용하지 않는 커널에서 약 34%의 실행 시간 감소가 있었다. 하지만 I/O Throttling 기법을 사용하지 않으면 시스템이 교착 상태에 빠지게 될 수 있으며, 특히 메인 메모리 크기가 작은 환경에서 더 쉽게 교착 상태에 빠질 수 있다. 즉, I/O Throttling은 메인 메모리 크기가 작은 환경, 혹은 사용자 프로세스가 대용량의 Buffered 쓰기를 수행할 때, 교착상태에 빠지는 것을 회피하기 위한 기법이다.

하지만 현대 메인 메모리의 비용은 점점 저렴해지고 있고, 한 프로세스가 Buffered 쓰기를 교착 상태에 빠질 정도로 수행하는 경우는 드물다. I/O Throttling은 이에 맞게 개선될 필요가 있는 기법이며, 향 후 개선 방안에 대해서 논의해 볼 것이다.

4. 사사

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터 육성지원사업의 연구결과(IITP-2016-H8501-16-1006)와 정보통신•방송 연구개발사업의 일환[R0601-15-1063, ICT 장비용 SW 플랫폼 구축]으로 수행되었음.

참고 문헌

[1] Pai, Vivek S., Peter Druschel, and Willy Zwaenepoel. "IO-Lite: a unified I/O buffering and caching system." ACM Transactions on Computer Systems (TOCS) 18.1 (2000): 37-66.

[2] Mathur, Avantika, et al. "The new ext4 filesystem: current status and future plans." Proceedings of the Linux symposium. Vol. 2. 2007.