안드로이드 플랫폼의 스토리지 접근 소프트웨어 계층의 성능 부하 분석

김혁중, 안정철, 신동군
성균관대학교 정보통신대학
wangmir@skku.edu, luckyjc7@skku.edu, dongkun@skku.edu

Performance Analysis on Storage IO Software Layer of Android Platform

Hyukjoong Kim, Jeongcheol Ahn, Dongkun Shin
School of Information and Communication Engineering, Sungkyunkwan University

요 약

스마트폰 등의 임베디드 시스템에서는 낸드 플래시 기반 저장장치를 주로 사용한다. 하지만 지금까지의 운영체제의 블록 IO 시스템은 하드 디스크를 대상으로 설계되었기 때문에 낸드 플래시 메모리 기반의 저장장치의 특성을 고려하지 못하였다. 또한, 낮은 성능의 하드디스크에서는 운영체제에서 IO를 처리하는 소프트웨어 계층의 부하가 무시될 수 있었으나, 고성능의 낸드 플래시 메모리에서는 문제가 될 수 있다. 본 논문에서는 스마트 디바이스의 운영체제 중 하나인 안드로이드 플랫폼을 기반으로 IO 요청을 수행하는 소프트웨어 계층별 성능을 측정하였으며, 또한 멀티 프로세스상에서 IO 성능에 어떤 영향을 받는지 관찰했다. 실험 결과 IO 요청의 단위가 작은 경우는 운영체제에서의 부하가 저장장치에서 요청을 처리하는 지연 시간보다 압도적으로 크게 나타났으며, 16KB 단위의 IO 요청에 대해서 전체 지연 시간의 90%를 차지하였다. 또한, 멀티 프로세스 환경에서 IO를 처리하면서 인터럽트를 처리하는 시간이 증가하는 것을 확인했다.

키워드: 안드로이드, 낸드 플래시 메모리, 리눅스 커널, 스토리지, IO

1 서론

모바일 기기 시장이 성장함에 따라 내부 저장장치로 낸드 플래시 메모리의 사용 역시 증가하였다. 낸드 플래시 메모리는 하드 디스크보다 향상된 IO 속도를 통해 임베디드 시스템의 주요한 저장장치로 자리 잡았다. 하지만 지금까지의 운영체제 개발 방향이 하드 디스크에 최적화되는 방향으로 발전해 왔기 때문에 기존의 운영체제 요소 중 낸드 플래시 메모리의 특성을 고려하여 많은 수정이 요구되고 있다. 하드 디스크를 사용할 시에는 하드 디스크에서의 처리 시간이 커서 운영체제의 부하는 큰 문제가 아니었지만, 낸드 플래시 기반 저장장치는 빠른 응답 시간을 제공하므로 운영체제의 블록 IO 처리 부하가 이슈화되었다. 예를 들어 인터럽트를 수행하는 부하가 커지는 문제를 해결하기 위해 블록 IO 요청을 폴링으로 처리하는 기법 등이 제시되었다.[3]

안드로이드 기반 스마트폰의 수요가 증가하면서 안드로이드 운영체제에 대한 연구가 활발히 이루어지고 있다. 안드로이드 플랫폼은 기본적으로 리눅스 커널 위에 C와 C++, Java로 구현된 서비스 플랫폼을 제공하는 구조이다. 하지만 이런 다중 계층 구조와 Java를 사용하기 위하여 dalvik 머신을 사용하는 형태는 안드로이드의 주요 성능 취약점이다.

본 논문에서는 낸드 플래시 기반 저장장치에 대한 호스트단의 부하를 관찰하기 위해 안드로이드의 플랫폼과 리눅스 커널, 저장장치 자체에서 생기는 시간 지연에 대해 분석하고, 멀티프로세스 상에서 그 결과가 어떻게 변화하는지를 관찰하였다. 실험 결과 저장장치 자체의 소요 시간 대비 호스트 단에서의 지연이 매우 크며, 특히 요청되는 IO 크기가 작을수록 부하가 현저히 커질 수 있음을 확인했다.

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2011-0027613).

2 안드로이드 IO 성능 분석

안드로이드 기반 스마트 디바이스에서의 IO 성능을 분석하기 위해서 안드로이드의 각 계층별 IO가 처리되는 시간을 측정하여 계층별로 생기는 시간지연을 관찰하였다.

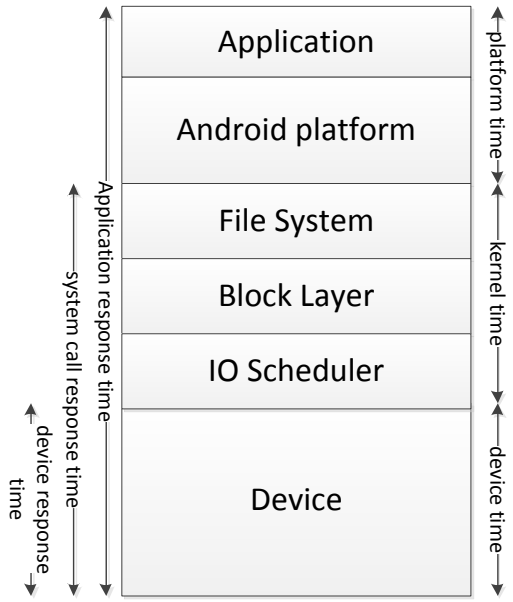

그림 1 안드로이드의 블록 IO 계층 구조와 성능측정 기법

2.1 안드로이드 IO 계층 분석

안드로이드는 그림 1에서 볼 수 있듯이 기본적으로 리눅스 커널과 커널 외의 안드로이드 서비스 플랫폼으로 그 구조를 구분 짓는다. 안드로이드 플랫폼은 다시 플랫폼과 Java 가상 머신을 대체하는 dalvik 머신으로 나눌 수 있으며, 리눅스 커널의 IO 계층은 파일시스템과 블록 IO 계층으로 구분 지을 수 있다. 본 논문에서는 이런 계층 구조를 기준으로 안드로이드의 IO 과정에서 생기는 부하를 크게 플랫폼, 커널, 저장장치 자체의 부하로 나누어 관찰하여 안드로이드에서 IO 시에 생기는 지연이 어느 부분에서 많이 생기는지, 또한 어느 영역이 최적화가 필요한지를 관찰했다.

2.2 성능 측정 방법

성능 측정은 실제 듀얼 코어 기반의 안드로이드폰을 사용하여 eMMC에 IO를 수행하는 형태로 진행하였다. 표 1은 실험에 사용된 기기의 사양을 보여준다. 계층별 지연되는 시간을 측정하기 위해서 파일 IO를 지속해서 요청하는 애플리케이션, 시스템콜을 관찰하기 위한 strace[1], 그리고 디바이스 자체의 소요 시간을 측정하기 위한 디바이스 드라이버 단에 추가한 트레이서를 사용했다.

표 1 실험에 사용된 안드로이드 폰 사양

운영체제	안드로이드 2.3.4 진저브레드
CPU	Exynos Dual-core 1.2GHz
메모리	1024MB RAM
내부 저장장치	eMMC 32GB
외부 저장장치	MicroSD 32GB

2.2.1 테스트 애플리케이션

안드로이드에서 IO 관찰을 하기 위해서 애플리케이션 단에서 쓰기 연산을 지속해서 발생시키는 테스트 애플리케이션을 자바 기반으로 구현하여 사용했다. 이 애플리케이션은 동기화 쓰기 연산을 통해 데이터가 저장장치에 쓰이는 것을 보장하도록 제작했다.

이 애플리케이션을 통해 쓰기 연산이 애플리케이션에서 요청되어서 응답하기까지의 지연시간을 관찰할 수 있다.

2.2.2 커널 지연 시간 측정

strace는 지정된 프로세스가 보내는 시스템콜들을 관찰할 수 있는 리눅스의 기본 제공 툴이다. Strace를 통해서 write() 시스템콜이 요청된 시점부터 응답이 된 시점까지의 지연시간을 관찰할 수 있으며 이를 통해서 파일 시스템으로 요청이 가기 직전으로부터 응답이 온 직후까지의 시간 측정이 가능하다.

```
command: adb shell strace -fF -c -p pid

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 44.82 2308.290093      179773     11051      1455 recv
 33.84 1742.784289      648840      2686        12 ioctl
 10.48  539.701314      149917      3600           semget
  5.89  303.205462        6012     50430           write
  0.09    4.633581          93     49899           msgget
  0.04    2.240565          68     32752           lseek
  0.01    0.504590         111      4539           mprotect
  0.01    0.294579          85      3476           gettimeofday
```

그림 2 strace 사용 예

그림 2는 strace의 사용 예이다. 프로세스 아이디를 통해 해당 프로세스에 대한 관찰이 가능하며, -c 옵션은 전체 시스템 콜에 대한 횟수, 평균 반응 시간 등을 출력해준다.

2.2.3 디바이스 지연시간 측정

파일 시스템과 블록 IO 계층을 거친 IO 요청은 장치 디바이스 드라이버를 거쳐서 저장장치에 전달된다.

본 논문에서는 저장장치에서 IO 처리를 위해 지연되는 시간을 측정하기 위해서 MMC 코어 드라이버에서 MSHCI(Mobile Storage Host Controller Interface) 드라이버로 요청을 보내기 직전과 IO 요청을 완료했다는 인터럽트가 발생한 직후 사이 시간을 관찰하였다. 그림 3는 MMC 디바이스 드라이버와 그에 연관된 계층의 구조를 나타내며 MMC 코어 드라이버와 MSHCI 드라이버의 사이에서 실제 시간을 측정하는 드라이버 모듈(mshci_debug)을 제작하여 시간을 측정하였다.

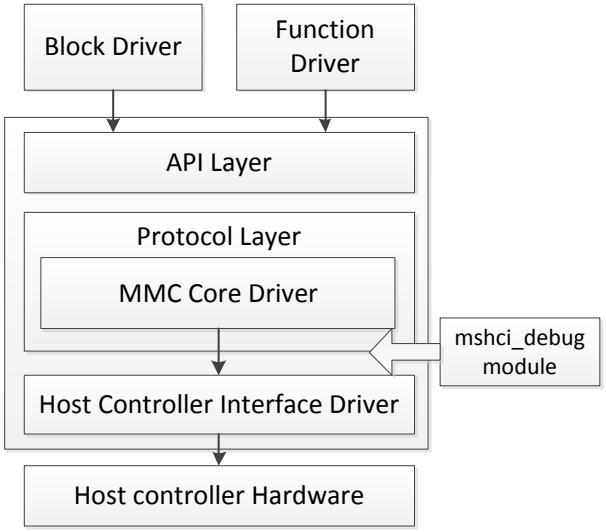

그림 3 MMC driver 구조와 debug 시점

2.3 실험 결과

실험은 두 가지 종류로 진행되었다. 첫 번째 실험은 각 IO 계층별로 생기는 부하를 측정하기 위한 실험으로 진행되었으며, 두 번째 실험은 멀티 프로세스 환경에서 IO 요청의 처리 시간을 분석하기 위한 실험을 진행하였다.

2.3.1 IO 계층별 성능 분석

그림 4은 테스트 애플리케이션을 IO 크기를 증가시켜 가면서 실행하였을 때 안드로이드의 계층별 부하를 보여준다. 그래프에서 볼 수 있듯이 플랫폼과 커널의 부하, 즉 호스트 운영체제의 IO 계층에서 생기는 부하가 IO 단위가 16KB인 경우 전체 수행 시간의 90% 이상을 차지하는 부하가 생기는 것을 관찰할 수 있었다. 플랫폼에서 생기는 부하는 애플리케이션에서 요청한 쓰기 연산이 dalvik 머신을 거치면서 생기는 지연시간으로 해석될 수 있으며, 커널에서 생기는 부하는 write() 시스템콜이 요청된 후 파일시스템과 블록 IO 계층에서 요청이 처리되어 저장장치로 들어가기 전까지의 지연시간이다. 이 결과를 통해서 dalvik 머신과 파일시스템, 블록 IO 계층에 대한 최적화가 필요함을 알 수 있다.

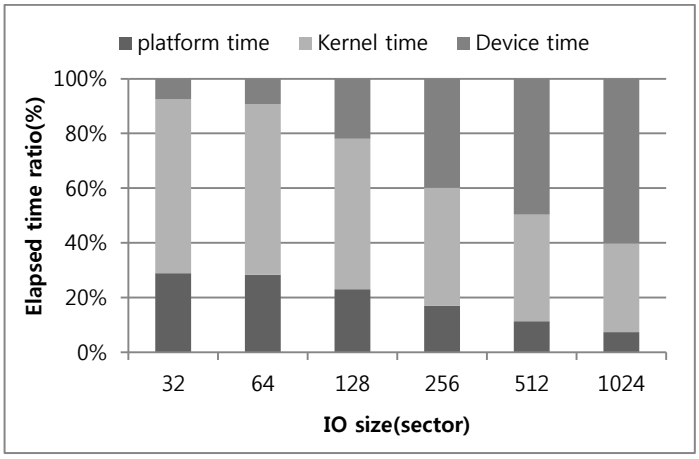

그림 4 안드로이드의 IO 계층별 소요 시간

2.3.2 멀티 프로세스 환경 성능 분석

멀티 프로세스가 동작하는 환경을 가정하기 위하여 2.3.1과 같은 실험을 CPU를 집약적으로 사용하고 IO 접근을 하지 않는 임의의 애플리케이션과 동시에 수행하면서 단일 애플리케이션 환경에서와 성능을 비교하였다. 테스트 애플리케이션만 동작하는 환경에서는 두 개의 코어 중 한 개의 코어가 유휴 상태에 들어가지만, 멀티 프로세스 환경에서는 두 개의 코어가 동시에 돌아가게 된다. 또한 듀얼 코어 환경이기 때문에, 인터럽트를 받는 코어와 요청된 인터럽트를 처리하는 코어가 다를 때 IPI(Inter-Processor Interrupt)를 사용한 코어간 통신을 통해 인터럽트를 전달해야 한다.

그림 5의 그래프에서 멀티 프로세스 환경에서의 성능은 단일 애플리케이션 환경보다 최대 20%까지 성능이 향상되었다. 이는 DVFS(Dynamic Voltage and Frequency Scaling)의 영향으로 테스트 애플리케이션만 사용하는 경우는 한 개의 CPU만 동작하며, CPU의 클록 속도가 200MHz에서 400MHz로 제한되는 것에 반해, 멀티 프로세스 환경에서는 CPU를 집약적으로 사용하기 때문에 CPU가 1GHz의 클록 속도로 동작하였기 때문이다.

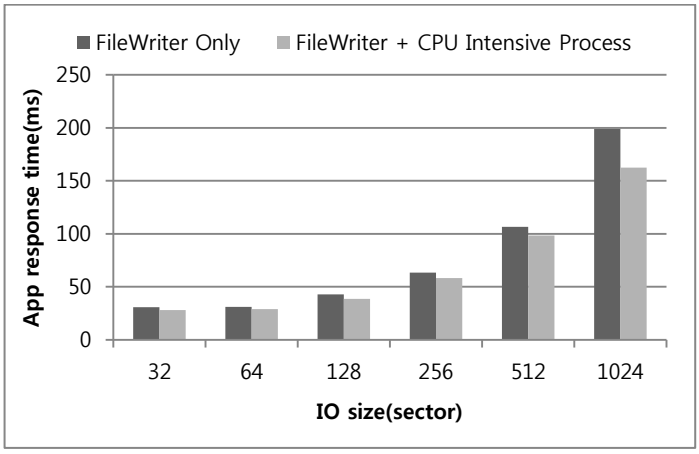

그림 5 멀티 프로세스 환경에서 애플리케이션 반응 시간

그림 6는 같은 실험에 대해서 저장장치에서의 부하만 따로 측정한 결과이다. 그래프의 결과는 두 개의 애플리케이션을 실행함으로써 두 개의 코어가 활성화 되면서 테스트 어플리케이션에서 요청한 쓰기 연산의 인터럽트가 코어 간 IPI를 유발하여 그림 5에서 전체 성능이 향상되었음에도 불구하고 디바이스의 반응 시간이 최대 7%까지 저하되는 것을 보여준다.

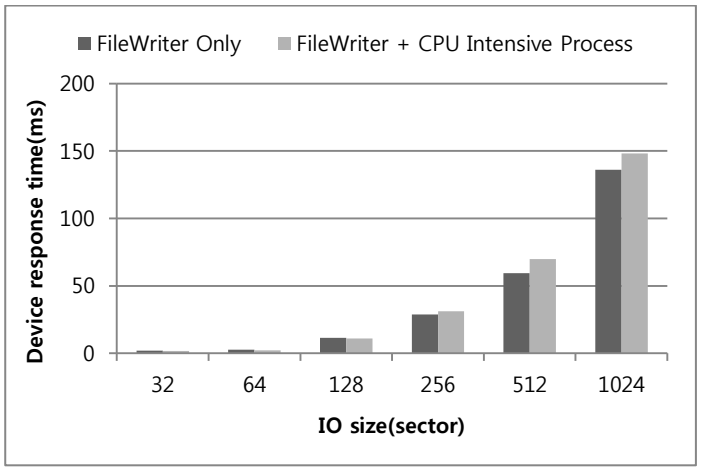

그림 6 멀티 프로세스 환경에서 디바이스 반응 시간

3 결론

본 논문에서는 안드로이드 플랫폼을 사용하는 스마트폰을 기반으로 호스트에서 생기는 부하에 대해 관찰하였다. 안드로이드 운영체제는 크게 안드로이드 서비스 플랫폼, 리눅스 커널의 두 부분으로 나누어서 플랫폼에서의 지연시간, 커널에서의 지연시간, 저장장치 자체에서 생기는 지연시간을 비교하였다. 실험 결과 호스트에서 생기는 부하가 저장장치의 처리 시간보다 더 크거나 동등하였다. 특히 IO 단위가 16KB일 때는 전체 시간의 90% 이상을 차지할 수 있음을 확인하였다.

또한, 멀티 프로세스 환경에서 동일한 실험을 수행하면서 멀티 프로세스 환경에서 IO 접근에 영향을 미칠 수 있는 요인을 분석하였으며, 디바이스에서 요청하는 소프트웨어 인터럽트를 관리하는 부하가 코어를 여러 개 사용하는 멀티 프로세스 환경에서는 증가할 수 있음을 확인하였다.

4 향후 연구

본 논문의 연구를 확장시키기 위해 앞으로는 멀티 코어 환경에서 소프트웨어 인터럽트를 빈번히 발생시킨다거나, 파일 시스템에 접근이 많은 등의 다양한 프로세스 모델을 만들어 IO 접근과 동시에 수행될 때 IO 접근이 어떤 간섭을 받는가에 대해서 관찰할 것이다. 또한 디바이스에서 소프트웨어 인터럽트를 요청하는 부하에 대해서 세부 분석을 수행하여 인터럽트를 처리하는 기법에 변화가 필요한지를 분석할 것이다.

궁극적으로 안드로이드 운영체제 단에서 IO 성능을 위한 최적화 기점을 찾아 안드로이드 운영체제의 교정을 통해 IO 성능을 향상하는 것을 목표로 한다.

➢ 참고 문헌

[1] strace, <http://sourceforge.net/projects/strace/>

[2] Android Developers, <http://developer.android.com/index.html>

[3] Jisoo Yang, Dave B. Minturn, “When Poll is Better than Interrupt”, *10th USENIX Conference on File and Storage Technologies*, February 2012