In-Storage Processing을 위한 SSD 소프트웨어 플랫폼

박대동○, 이재수, 홍성수
서울대학교 전기컴퓨터공학부
{ddpark, jslee, sshong}@redwood.snu.ac.kr

SSD Software Platform for In-Storage Processing

Daedong Park○, Jaesoo Lee, Seongsoo Hong
School of Electrical Engineering and Computer Science, Seoul National University

요 약

In-storage processing(ISP)은 I/O 인터페이스의 병목으로 인한 데이터 전송 성능 제약 문제를 해결하기 위한 효과적인 기법이다. 특히 플래시 메모리를 저장 매체로 사용하는 SSD는 I/O 인터페이스에 비해 내부 데이터 전송 속도가 훨씬 빠르게 발전하고 있어 ISP의 필요성이 더욱 증대되고 있다. 그러나 기존의 ISP에 관한 연구는 특정 파일 시스템이나 응용에만 제한되어 SSD에 적용하기 어렵고 다양한 응용을 개발하기 어렵다는 문제가 존재한다. 이 논문에서는 이러한 문제를 해결하기 위해 ISP를 위한 SSD 소프트웨어 플랫폼을 제안한다. 제안된 소프트웨어 플랫폼은 응용의 동적 설치 및 제거가 가능할 뿐만 아니라 이벤트 서비스를 통한 동적 기능 확장이 가능하다. 우리는 먼저 프로그래밍 모델을 제안하고 다음으로 이를 지원하는 SSD 소프트웨어 플랫폼을 설계한다. 마지막으로 제안된 SSD 소프트웨어 플랫폼을 구현하고 실험을 통해 성능 향상 및 기능 확장을 검증한다. 실험 결과 SSD를 사용하는 응용의 응답 속도가 47.3%로 감소하였으며 동적으로 새로운 기능을 SSD에 추가하는 것이 가능함을 확인하였다.

1. 서 론

최근 저장 장치의 발달로 SSD(Solid State Drive)의 사용이 크게 증가하고 있다. SSD는 하드디스크에 비해 매우 빠른 읽기 및 쓰기 속도를 제공한다. 더구나 플래시 칩과 버스의 다양한 병렬화 기법에 힘입어 현재도 그 속도가 지속적으로 증가하고 있다. 반면 저장 장치와 호스트 시스템 사이의 I/O 인터페이스 속도는 SSD의 발전 속도를 따라가지 못하고 있다. 그 결과 I/O 인터페이스가 데이터 전송의 병목 구간이 되어 SSD가 장착된 시스템의 성능 향상을 제한하게 된다.

그런데 일반적으로 대량의 데이터 전송이 필요한 프로그램의 실제 연산은 간단한 경우가 많다. 만약 이런 연산을 저장 장치 내에서 수행할 수 있다면 전송되는 데이터의 양을 크게 줄일 수 있다. 이를 위해 저장 장치 내에서 응용의 일부를 수행하는 ISP 기법의 연구가 이루어졌다[1-3]. 그러나 기존 연구는 특정 파일 시스템이나 응용에만 제한되어 SSD에 적용하기 어렵고 다양한 응용을 개발하기 어렵다는 문제가 존재한다.

이 논문에서는 이러한 문제를 해결하기 위해 ISP를 위한 SSD 소프트웨어 플랫폼을 제안한다. 이를 위해 개발자를 위한 SSD 프로그래밍 모델을 제안하고 이를 지원하는 SSD 소프트웨어 플랫폼을 설계한다. 제안된 프로그래밍 모델은 저장 장치에서 동작하는 응용뿐만 아니라 저장 장치 내의 특정 이벤트 발생 시 동작할 처리기의 개발을 가능하게 한다. 또한 사용자는 제안된 소프트웨어 플랫폼을 사용하여 쉽게 3rd 파티 응용을 저장 장치 내에 설치하고 관리할 수 있다.

우리는 SSD 소프트웨어 플랫폼을 실제로 구현하고 실험을 통해 처리 성능 향상 및 동적 기능 확장을 검증하였다. 실험 결과 데이터베이스 응용의 select 명령 응답 시간이 47.3%로 감소하였다.

이 논문의 나머지 부분은 다음과 같이 구성된다. 2장에서는 문제를 정의한다. 3장과 4장에서는 SSD 프로그래밍 모델과 소프트웨어 플랫폼에 대해 각각 설명한다. 5장에서는 실험을 통한 검증을 보이고 마지막으로 6장에서 결론을 내린다.

2. 문제 정의

2.1 시스템 모델

전체 시스템은 그림 1과 같이 호스트 시스템과 대상 시스템으로 구성된다. 호스트 시스템은 사용자와 직접 상호작용 하는 시스템으로 데스크톱 PC가 그 예이다. 호스트 시스템은 대상 시스템으로 데이터 읽기 및 쓰기 명령을 요청한다. 대상 시스템은 ISP 응용이 수행 가능한 SSD 시스템이다. 두 시스템은 SATA와 같은 I/O 인터페이스를 통해 서로 연결된다.

2.2 문제 정의 및 해결 방안

최근 개발되고 있는 SSD 내부의 버스 속도는 6 Gbit/s

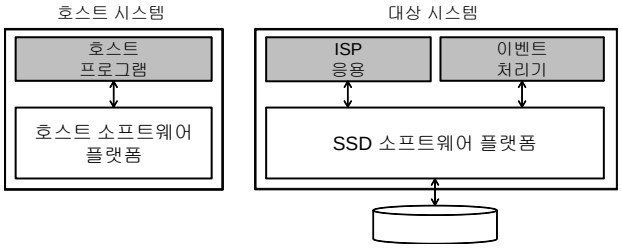

그림 1. 시스템 모델

이상이다. 하지만 현재 많이 사용되는 SATA2 인터페이스의 최대 속도는 3 Gbit/s일 뿐만 아니라 실제 동작 속도는 이보다 낮은 경우가 많다. 이로 인해 데이터 전송이 많은 프로그램의 동작 시 호스트 시스템과 SSD 사이의 I/O 인터페이스가 병목 구간이 된다.

우리는 그림 2와 같이 ISP를 통해 I/O 인터페이스를 통한 데이터 전송을 줄임으로써 문제를 해결한다. 화살표의 굵기가 데이터 전송 크기를 의미한다. 그림에서와 같이 병목 구간이 SSD 내부로 변경되기 때문에 전체 시스템의 성능 향상을 가져올 수 있다. 이어지는 장에서 이를 위한 SSD 프로그래밍 모델을 제안하고 SSD 소프트웨어 플랫폼 설계를 기술한다.

3. SSD 프로그래밍 모델

본 장에서는 우리가 제안하는 SSD 프로그래밍 모델을 설명한다. SSD를 사용하는 프로그램은 그림 1의 회색 블록과 같이 호스트 프로그램, ISP 응용 그리고 이벤트 처리기의 세 종류로 분류된다.

3.1 호스트 프로그램 API

호스트 프로그램은 호스트 시스템에서 수행되며 SSD에 데이터를 저장하고 사용하는 프로그램이다. 호스트 프로그램에는 다음 두 종류의 API가 제공된다.

- 대상 시스템의 ISP 응용 관리 API

  호스트 프로그램은 이들 API를 통해 ISP 응용을 설치, 제거, 실행, 중지할 수 있으며 또한 ISP 응용의 상태를 감시할 수 있다. 이들 API는 대상 시스템과의 통신을 위하여 RPC를 사용한다.

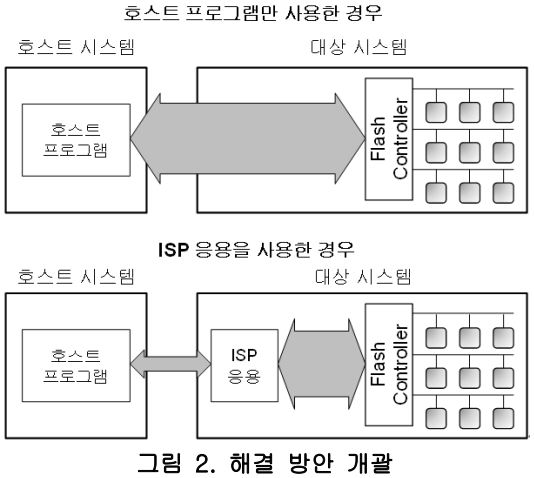

그림 2. 해결 방안 개괄

- SSD에 저장된 데이터 접근을 위한 API

  호스트 프로그램은 이들 API를 통해 SSD의 데이터를 읽고 쓸 수 있다. 이들 API는 호스트 시스템의 운영체제에 의해 정의되고 파일 시스템을 통해 제공된다. 본 연구에서는 Linux의 파일 시스템 관련 시스템 콜들을 사용한다.

3.2 ISP 응용 API

ISP 응용은 대상 시스템에서 수행되며 호스트 프로그램의 기능 일부를 나누어 담당한다. 이 응용은 저장 장치의 데이터에 접근하기 위한 API와 호스트 프로그램과 통신하기 위한 API를 필요로 한다. 우리는 이들 두 종류의 API들을 별도로 구현하지 않기 위해 저장 장치에 데이터를 기록하여 호스트 프로그램과 ISP 응용이 통신하는 방법을 개발하였다. 따라서 ISP 응용은 다음 API를 사용한다.

- 대상 시스템의 데이터 접근 인터페이스

  ISP 응용은 이들 API를 통해 SSD에 저장된 데이터에 접근할 수 있다. 우리가 제안하는 SSD는 OSD(Object-based Storage Device)파일 시스템[4]을 사용하므로 모든 데이터는 object로 저장되고 OSD-2 표준[5]이 정의하는 명령어를 사용하여 object에 접근한다.

3.3 이벤트 처리기 API

마지막으로 이벤트 처리기는 대상 시스템에서 수행되며 SSD 내에서 특정 이벤트가 발생했을 때 자동으로 수행되는 프로그램이다. SSD 내부의 이벤트는 파일 시스템의 동작에 의해 발생하는 이벤트와 FTL(Flash Translation Layer)의 동작에 의해 발생하는 이벤트가 있다. 이들 이벤트 처리기가 사용하는 API는 다음 두 종류로 분류된다.

- 이벤트 처리기 등록 및 해제를 위한 인터페이스

  이벤트 처리기는 이들 API를 통해 자신을 특정 이벤트 발생시 호출되도록 등록하거나 반대로 해제할 수 있다. 이들 API는 유틸리티 프로그램에 의해 호출될 수도 있다.

- 대상 시스템의 데이터 접근 인터페이스

  이벤트 처리기는 이들 API를 통해 SSD에 저장된 데이터에 접근할 수 있다. ISP 응용을 위한 API와 동일하게 OSD-2 표준 명령어를 사용한다.

4. SSD 소프트웨어 플랫폼

본 장에서는 제안하는 SSD 소프트웨어 플랫폼에 대해 설명한다. 그림 3은 전체 시스템의 소프트웨어 블록 다이어그램이다. 그림에서 볼 수 있듯이 소프트웨어 플랫폼은 다음 두 종류의 서브 시스템 블록들로 구분된다.

- 호스트 시스템 소프트웨어 플랫폼의 서브 시스템
- 대상 시스템 소프트웨어 플랫폼의 서브 시스템

본 장의 나머지 부분에서는 이들 두 서브 시스템들의

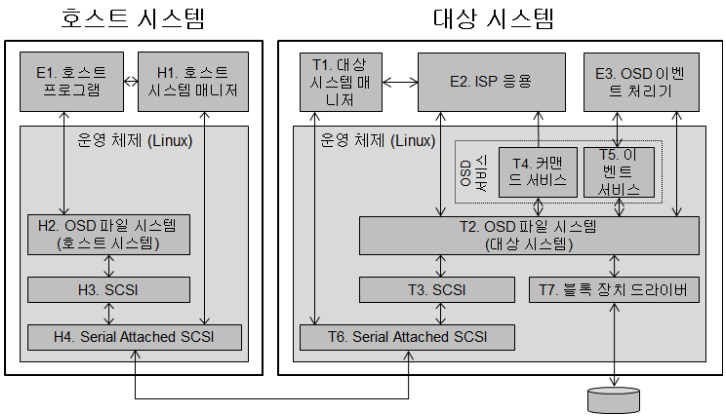

그림 3. 소프트웨어 블록 다이어그램

정의들을 설명한다.

4.1 호스트 시스템

본 절에서는 호스트 시스템 소프트웨어 플랫폼을 구성하는 서브 시스템들을 설명한다. 총 4개의 서브 시스템으로 구성된다. 각각에 대한 설명은 다음과 같다.

- H1. 호스트 시스템 매니저
  호스트 프로그램에게 ISP 응용을 관리하는 API를 제공하고 ISP 응용을 관리하는 명령을 대상 시스템으로 한다. 대상 시스템과의 통신은 RPC를 사용한다.
- H2. OSD 파일 시스템(호스트 시스템 부분)
  호스트 시스템에 위치하는 OSD 파일 시스템으로 Linux 표준 파일 입출력 인터페이스를 유저 레벨 프로그램들에게 제공한다. Linux 파일 입출력 명령이 호출되면 이를 알맞은 OSD-2 표준 명령어로 변환하여 SCSI 드라이버로 전송한다.
- H3. SCSI 드라이버
  OSD 표준 명령어를 SCSI CDB(Command Descriptor Block)으로 변환하여 하단에 위치한 Serial Attached SCSI로 전송한다.
- H4. Serial Attached SCSI
  SCSI CDB를 SATA 명령어로 변환하여 SATA 인터페이스를 통해 전송한다.

4.2 대상 시스템

본 절에서는 대상 시스템 소프트웨어 플랫폼을 구성하는 서브 시스템들을 설명한다.

- T1. 대상 시스템 매니저
  호스트 시스템 매니저에게 ISP 응용을 관리하는 인터페이스를 제공하고 ISP 응용을 관리하는 명령을 실제로 처리한다. 호스트 시스템과의 통신은 RPC를 사용한다.
- T2. OSD 파일 시스템(대상 시스템 부분)
  대상 시스템에 위치하는 OSD 파일 시스템으로 OSD-2 표준 명령어 인터페이스를 제공한다. SCSI 드라이버로부터 OSD-2 표준 명령어를 입력 받고 object에 직접 접근하여 알맞은 동작을 수행한다. Object를 생성, 수정, 제거하고 object의 attribute를 설정하는 작업 등이 이루어진다.
- T3. SCSI 드라이버
  Serial Attached SCSI로부터 전송 받은 SCSI CDB패킷을 분해하여 OSD-2 표준 명령어를 찾아낸다. 분해된 OSD-2 표준 명령어에 따라 OSD 파일 시스템의 해당 인터페이스를 호출한다.
- T4. 커맨드 서비스
  호스트 프로그램과 ISP 응용간의 데이터 교환을 지원한다. 대상 시스템 초기화 시에 두 개의 특수 object를 생성하여 데이터 교환에 사용한다. 하나는 호스트 프로그램이 ISP 응용으로 명령을 전송할 때 쓰는 명령어 수신 object이며 다른 하나는 ISP 응용의 처리 결과를 호스트 프로그램으로 전송할 때 쓰는 응답 object이다. 주기적으로 명령어 수신 object를 감시하여 새로운 명령어가 들어왔는지 검사하고 새로운 명령어가 수신 되었으면 알맞은 ISP 응용을 찾아서 명령어를 전달한다.
- T5. 이벤트 서비스
  SSD 내의 이벤트 발생을 감시하고 이벤트 발생 시 해당 이벤트에 등록된 이벤트 처리기를 자동으로 수행한다. 이벤트 처리기의 등록과 해제를 위한 인터페이스를 제공한다.
- T6. Serial Attached SCSI
  SATA 인터페이스를 통해 들어온 ATA 표준 명령어를 SCSI CDB로 변환하고 SCSI 드라이버로 전송하는 서브 시스템이다.
- T7. 블록 장치 드라이버
  블록 장치에 읽기 및 쓰기 명령을 내린다. 본 연구에서는 SSD에 특화된 장치 드라이버가 사용된다. 이 장치 드라이버는 블록 캐시, FTL 및 기타 기능을 구현한 소프트웨어를 포함한다.

5. 구현 및 검증

본 장에서는 먼저 소프트웨어 플랫폼의 구현에 대해 설명한다. 다음으로 성능 향상 및 동적 기능 확장을 검증한 실험을 설명한다.

5.1 SSD 소프트웨어 플랫폼 구현

우리는 그림 4와 같은 하드웨어 환경 위에 SSD 소프트웨어 플랫폼을 구현하였다. 이때 기존의 상용 SSD는 범용 프로세서와 메모리가 없기 때문에 그대로 사용이 불가능하다. 따라서 우리는 ISP가 수행 가능한 SSD를 에뮬레이션하기 위해 프로세서 및 메모리를 탑재한 NAS(Network Attached Storage)를 사용했다. 실험 환경과 제안한 시스템은 병목

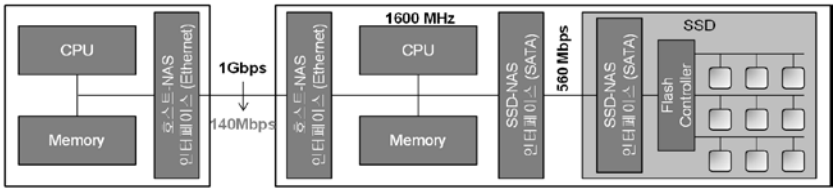

그림 4. 실험 하드웨어 환경

구간과 연산 성능에 차이가 존재한다. 우리는 다음 방법을 통해 차이를 극복하였다.

- 병목 구간

  실험의 병목 구간은 NAS와 SSD 사이의 인터페이스이다. 우리는 TCP flow control 기술을 통해 호스트 시스템과 NAS 사이의 속도를 조절하여 병목 구간을 변경하였다.

- 연산 성능

  제안된 시스템은 400 MHz의 프로세서와 병렬 연산을 위해 특별히 설계된 하드웨어 가속기를 사용한다. 실험 환경에서는 고속의 하드웨어 가속기가 없는 대신 1600 MHz의 프로세서를 사용하였다.

5.2 성능 및 기능 검증

우리는 성능 및 기능 검증을 위해 두 가지 실험을 수행하였다. 첫째로 ISP를 통한 성능 향상을 검증하기 위해 데이터베이스 프로그램을 사용하였다. {계좌, 이름, 나이, 성별, 잔고}를 임의로 생성하여 대용량의 데이터베이스를 구축하고 select 명령을 수행하여 응답 시간을 측정했다. 비교 대상은 호스트 프로그램만으로 구현된 경우와 select 명령 처리를 ISP 응용으로 분리해 구현한 경우다. ISP 응용은 일반적인 select 명령을 처리할 수 있도록 구현하였다. 실험 결과는 그림 5와 같다. 가로 축은 데이터베이스의 크기(MB)이고 세로 축은 응답 시간(sec)이다. 실험 결과 응답 시간이 평균 47.3%로 감소하였다. 이를 통해 병목 구간이 프로세서 및 메모리 부분으로 변경된 것을 확인할 수 있다. 향후 간단한 연산을 위한 하드웨어 가속기를 추가함으로써 병목 구간을 하드웨어 가속기와 플래시 컨트롤러 사이로 변경할 수 있으며 이 경우 추가적인 성능 향상을 기대할 수 있다.

다음으로 이벤트 처리기의 추가를 통한 동적 기능 확장을 검증하였다. 실험에서는 SSD에 저장되는 mp3 데이터의 품질을 자동으로 변환하는 이벤트 처리기를 사용하였다. 이벤트 처리기는 OSD-2의 WRITE_OBJECT 명령어에

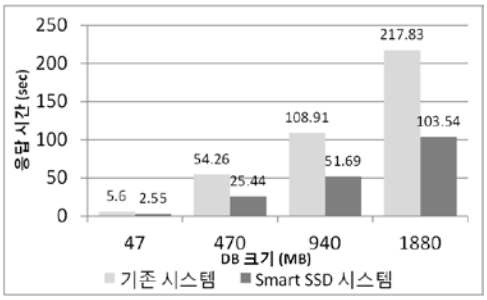

그림 5. DB 응용의 select 명령 응답 시간

등록되며 데이터가 기록될 때 헤더를 검사하고 mp3 데이터인 경우 bitrate를 128 Kbps로 변경하여 기록한다. 이벤트 처리기의 등록 후 320 Kbps의 mp3 데이터를 SSD로 복사하면 잠시 후에 128 Kbps의 mp3 데이터로 변환되어 기록되는 것을 확인할 수 있었다.

6. 결 론

본 연구에서 우리는 SSD 프로그래밍 모델을 제안하고 이를 지원하는 SSD 소프트웨어 플랫폼을 설계하였다. 또한 제안된 SSD 소프트웨어 플랫폼을 구현하고 실험을 통해 그 유용성을 검증하였다. 제안된 SSD 소프트웨어 플랫폼은 3rd 파티 ISP 응용의 동적 다운로드 및 상태 관리가 가능할 뿐만 아니라 3rd 파티 이벤트 처리기의 등록을 통해 동적 기능 확장이 가능하다. 이 연구 결과를 활용하여 고성능의 SSD 개발 및 SSD를 위한 다양한 응용 개발이 이루어질 수 있을 것으로 기대한다.

Acknowledgement

본 연구는 삼성전자 메모리 사업부의 지원을 받아 수행하였음. (No. 0421-20100068, SSD를 위한 컴포넌트 기반의 내장형 고성능 소프트웨어 플랫폼 연구)

참고 문헌

[1] A. Acharya, M. Uysal, and J. Saltz, “Active Disks: Programming Model, Algorithms and Evaluation”, ACM SIGPLAN Notices, vol. 33(11), pp. 81~91, 1998.

[2] R. Reambasu, A. A. Levy, T. Kohno, A. Krishnamurthy, and H. M. Levy, “Comet: An Active Distributed Key-Value Store”, In Proc. Of OSDI 2010.

[3] S. Ahn, J. Choi, D. Lee, S. H. Noh, S. Min, and Y. Cho, “Design, Implementation, and Performance Evaluation of Flash Memory-based File System on Chip”, Journal of Information Science and Engineering, vol. 23(6), pp. 1865~1887, 2007.

[4] M. Mesnier, G. R. Ganger, and E. Riedel, “Object-Based Storage”, IEEE Communications Magazine, vol. 41(8), pp. 84~90, 2003.

[5] T10, “The Object-Based Storage Device Commands-2”, INCITS, 2009.