SOAR : 저장장치를 기반으로 하는 시스템의 신뢰성 분석도구 개발

(SOAR : Storage Reliability Analyzer)

김 영 진 † 원 유 집 †† 김 락 기 †
(Youngjin Kim) (Youjip Won) (Rakie Kim)

요 약 대용량 멀티미디어 파일의 증가와 개인의 디지털 정보의 중요성이 날로 증가하면서 저장장치는 고용량화, 고집적화 되는 방향으로 발전하고 있다. 따라서 저장장치에 발생되는 물리적인 오류는 단순히 작은 영역의 손상이 아닌 매우 넓은 영역에 대한 정보 손실로 이어진다. 이를 방지하기 위해서는 시스템을 사용하기 전에 물리적인 오류에 대한 시스템의 강인성과 대처 수준을 검증하고 사용해야 한다. SOAR(Storage Reliability Analyzer)는 검증의 핵심이 될 수 있는 물리적인 오류 발생 기능과 복구 기능을 가지고 있으며 이것은 시스템에 대한 신뢰성과 강인성을 검증 할 수 있는 유용한 도구이다. 이 기능을 보다 편리하게 사용하기 위해서 SOAR는 3가지의 특수한 오류 적용 기법과 파일시스템에 특화된 2가지 기법을 가지고 있다. 본 논문에서는 SOAR를 이용해서 어플리케이션부터 파일시스템까지 물리적인 오류에 대한 검증을 실제 수행하고 결과 분석을 진행하였다. 그러므로 SOAR는 물리적인 오류에 대한 시스템의 많은 문제점을 발견하였고 동시에 그 기능을 증명하였다.

키워드 : 신뢰성, 강인성, 저장장치, 물리적인 오류, 베드섹터

Abstract As the number of large size multimedia files increases and the importance of individual's digital data grows, storage devices have been advanced to store more data into smaller spaces. In such circumstances, a physical damage in a storage device can destroy large amount of important data. Therefore, it is needed to verify the robustness of various physical faults in storage device before certain systems are used. We developed SOAR(Storage Reliability Analyzer), Storage Reliability Analyzer, to detect physical faults in diverse kinds of HDD hardware components and to recover the systems from those faults. This is a useful tool to verify robustness and reliability of a disk. SOAR uses three unique methods of creating physical damages on a disk and two unique techniques to apply the same feature on file systems. In this paper, we have performed comprehensive tests to verify the robustness and reliability of storage device with SOAR, and from the verification result we could confirm SOAR is a very efficient tool.

Key words : reliability, robustness, storage, physical faults, badsector

· 본 연구는 KOSEF(R0A-2007-000-20114-0)의 한양대 National Research Lab 지원을 받아 수행하였습니다.

† 학생회원 : 한양대학교 전자컴퓨터통신학과 연구원
kyj79@paran.com
daybreak24@ece.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자컴퓨터통신학과 교수
yjwon@ece.hanyang.ac.kr

논문접수 : 2007년 5월 29일
심사완료 : 2008년 3월 19일

정보과학회논문지: 시스템 및 이론 제35권 제6호(2008.6)

1. 서 론

1.1 연구동기

현대 사회의 디지털 정보는 간단한 콘텐츠부터 금전적으로 살 수 없을 정도의 중요한 콘텐츠에 이르기까지 날로 그 종류가 증가하고 있다. 요즘 디지털 정보 중 가장 중요시 여기는 개인 정보는 외부에 알려졌을 경우 개인의 프라이버시를 침해할 수 있고, 사라졌을 경우 큰 곤경에 빠질 수 있어 학계에서는 디지털 정보의 보호를 위해 많은 연구가 진행되고 있다. 그리고 이와 같은 디지털 정보는 일반적으로 개인 PC나 노트북에 저장하게 되는데 이때의 저장장치는 대부분 하드디스크를 사용하고 있다. 이러한 저장장치에 논리적인 오류가 아닌 물리

적인 오류가 발생되었고, 해당 위치가 중요한 디지털 정보가 담긴 영역이라면 되돌릴 수 없는 상황이 된다. 또한 저장장치의 대용량화, 디스크의 소형화, 고속화 위주로 연구 개발되면서 디스크의 집적도는 증가하여 보다 많은 정보를 잃어버릴 수 있게 되었다.

기존의 어플리케이션이나 파일시스템 검증은 저장장치의 물리적인 오류를 고려하지 않은 채 시스템의 논리적인 오류에 대한 검증만을 하고 있다. 하지만 저장장치는 수많은 물리적인 오류가 발생할 수 있는 불안전한 공간이기 때문에 어플리케이션이나 파일시스템이 물리적인 오류에 대해 적절한 대처를 하고 있는지 검증할 필요가 있다. 이와 같은 검증을 위해서는 검증을 위한 도구 개발이 선행 되야 하며 도구의 신뢰성 또한 갖추어져야 검증의 신뢰성을 얻을 수 있다.

1.2 관련연구

기존에 저장장치에 대한 물리적인 오류를 발생시켜주는 대표적인 도구는 IDE 디스크 기반의 물리적인 오류 발생 도구인 Fault Injection Tool[1]이 있다. 이 도구는 IDE디스크 기반의 저장장치에 물리적인 오류를 발생 시켜 주는 도구로서 실제 저장장치에 물리적인 오류를 발생 시키는 기법이 아닌 디바이스 드라이버 영역을 이용해서 가상의 물리적인 오류를 발생시키는 기법을 가지고 있다. 이 기법은 물리적인 오류를 가상으로 발생 시켜 파일시스템, 디스크 시스템, I/O인터페이스 사이의 신뢰성을 검증하기 위한 소프트웨어 기반의 오류 발생 도구로 리눅스의 IDE디스크 드라이버와 그 상위의 시스템 사이에서 논리적인 접근으로 원하는 디스크의 영역에 물리적인 오류를 발생시킨다. 이것은 커널 내의 자료구조와 디바이스 드라이버 영역의 소스 코드를 수정해서 개발한 도구이기 때문에 커널에 존재하는 Request 구조체가 핵심 자료구조 이다. 커널에 존재하는 Request 구조체란 디스크 접근 요청이 물리적 블록 장치마다 있는 요청 큐[2]에 들어오면 엘리베이터 알고리즘에 의해 디스크 헤드 이동을 최소화 하는 방향으로 알맞게 정렬하여 시스템 성능을 높이는데 관여되는 자료구조이다. 이와 같은 Request구조체를 조작한다면 저장장치로부터 읽거나 쓰는 등의 작업을 조작할 수 있기 때문에 물리적인 오류가 발생한 것과 같은 효과를 줄 수 있다. 이와 같은 방식의 장점은 현재 저장장치의 디지털 정보를 훼손하지 않으면서 시스템의 물리적인 오류에 대한 검증이 가능하다. 하지만 단점으로는 실제 저장장치에 발생한 것이 아닌 가상의 물리적인 오류이기 때문에 검증을 위한 범위에 한계가 있고 재 부팅 된 후에는 오류 정보가 사라진다. 이와 같이 파일시스템에 대한 강인성 및 무결성을 검증하는 연구[3]뿐만 아니라 시스템의 신뢰성과 강인성을 검증하기 위한 연구[4]도 함께 진행 중이다. 하지만 아직 미흡한 수준이다.

이와 같은 연구가 보다 활발히 이루어지기 위해서는 저장장치 벤더와의 밀접한 교류와 상세한 저장장치의 정보가 제공되어야 한다. 하지만 저장장치 벤더의 입장에서 보면 자신의 제품에 대한 결점을 노출하는 상황이 되기 때문에 연구에 많은 어려움이 있는 실정이다. 여러 가지 어려움 속에서 저장장치의 물리적인 오류를 실제 발생 시킬 수 있는 지능형 도구 SOAR를 개발하였다.

2. 입출력 시스템 구조의 한계

일반적인 시스템이라면 그림 1과 같이 입출력에 관련한 시스템 구조가 유저 어플리케이션, 파일시스템, 디바이스 드라이버, 저장장치로 각각 나누어져 있으며 각각 명확한 경계로 이루어져 있다. 디바이스 드라이버 영역은 저장장치에서 발생 되는 물리적인 오류 및 저장장치 관련 정보의 대부분을 알 수 있는 영역이다. 왜냐하면 저장장치 관련 여러 레지스터 조작을 독립적으로 수행하고 있기 때문이다. 물론 다른 영역에서도 저장장치 관련 레지스터[5]에 접근하여 값을 확인할 수 있지만 그렇게 해서 얻어지는 값은 올바른 레지스터 값이 아니다. 파일시스템과 유저 어플리케이션간에도 단순히 에러 코드만을 전달할 뿐 더 이상의 어떠한 정보도 공유하지 않는다. 따라서 각 계층간의 명확한 구분[6]으로 인해 계층간 필요로 하는 중요한 정보를 공유하지 못하는 시스템 구조임을 확인할 수 있다.

이와 같은 문제는 저장장치에 발생된 물리적인 오류 정보가 상위 계층으로 전달되지 않아 하위 계층인 하드웨어는 점점 심각한 상태로 진행되고 있어도 상위 계층은 튼튼한 하드웨어를 가졌다고 생각하여 정상적인 작업을 계속 수행하게 된다. 결국 문제를 인지하지 못한 채 어느 순간 데이타를 모두 손실할 수 있는 위기상황에 봉착할 수 있다. 더군다나 연구학자들 또한 각각의 영역에 대해서 독립적인 연구 활동을 할 뿐 통합적인 접근은 아직 미흡한 상태이다.

과거의 저장장치에 비해 현대의 저장장치는 집적도가 매우 향상되어 작은 데이타의 손실이 아닌 막대한 양의 디지털 정보가 손실될 수 있다. 따라서 현재의 시스템이

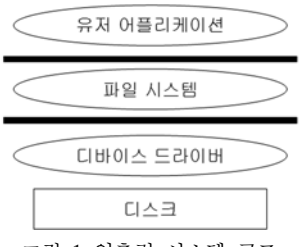

그림 1 입출력 시스템 구조

물리적인 오류에 대한 강인성을 어느 수준으로 가지고 있는지 확인할 필요가 있다. 이러한 이유로 개발된 SOAR는 ATA[7] COMMAND를 이용하는 SATA/IDE 디스크 저장장치에 실제 물리적인 오류를 발생시켜 물리적인 오류에 대한 시스템 검증뿐만 아니라 검증의 신뢰성을 향상시켰다. 또한 실제 물리적인 오류로부터 디지털 정보를 보호가 반드시 필요할 경우를 대비해서 가상의 물리적인 오류를 발생시킬 수 있게 SOAR를 구성하였다. 이는 디바이스 드라이버를 이용해서 실제 물리적인 오류와 같은 효과를 줄 수 있을 만큼의 정교한 기능이다.

3. SOAR의 물리적인 오류 발생

SOAR의 물리적인 오류 발생 기능은 2가지가 있다. 첫째는 ECC-CREATOR(Error Correcting Code-CREATOR) 기능으로 사용자가 원하는 저장장치에 정확한 위치를 파일시스템에서 구분하는 블록번호 또는 LBA (Logical block addressing)주소를 통해서 접근한 후 읽기 실패 오류의 대표적인 베드섹터를 물리적으로 실제 발생시키는 기능이다. 두 번째는 ERROR-CREATOR 기능으로 디바이스 드라이버에서 정의 하고 있는 모든 물리적인 오류에 대해서 ECC-CREATOR기능과 같이 원하는 위치를 블록번호 또는 LBA주소를 통해서 선정 가능하고, 읽기 실패뿐만 아니라 쓰기 실패, 인터페이스 오류 등을 최대한 실제 현상과 비슷하게 구축한다.

이와 같은 2가지 기능으로 ATA COMMAND를 이용하는 저장장치에 대해서 물리적인 오류에 대한 시스템의 반응을 쉽게 확인할 수 있으며, 그로 인해 시스템의 강인성과 신뢰성을 간단한 프레임워크를 통해 검증하였다.

3.1 저장장치 오류를 발생 시킬 수 있는 기능 및 기법

표 1은 위와 같은 SOAR에서 핵심이 되는 2가지 오류 발생 기능을 기존의 도구와 비교 분석하였다.

표 1에서 분석한 Fault Injection Tool[1]과 같이 저장장치에 물리적인 오류를 발생 시키기 위한 연구의 대부분은 가상의 물리적인 오류를 발생시키는 방식일 뿐 실제 저장장치를 손상시킬 수 있는 기능은 가지고 있지 않다. 하지만 SOAR의 ECC-CREATOR기능은 실제 저장장치에 베드섹터를 발생시켜 읽기 실패를 유도할 수 있다. 왜냐하면 저장장치로 접근할 수 있는 ATA COMMAND를 이용해서 원하는 섹터에 정확히 ECC (Error Correcting Code) 오류를 발생시킬 수 있는 기능을 가지고 있기 때문이다.

SOAR가 가지고 있는 또 하나의 물리적인 오류 발생 기능은 ERROR-CREATOR로 기존에 연구 개발되고 있는 디바이스 드라이버를 이용한 오류 발생 기법을 적용하고 있다. 하지만 Fault Injection Tool과 같은 기존 도구에서는 물리적인 오류를 디바이스 드라이버 영역에서 가상으로 만들었기에 현실성이 매우 떨어진다. 왜냐하면 오염된 섹터에 읽기 접근이든 쓰기 접근이든 어떠한 접근이 이루어지면 디바이스 드라이버 영역에서 무조건 저장장치 오류 메시지를 발생시키기 때문이다. 그래서 SOAR의 ERROR-CREATOR는 원하는 섹터에 원하는 물리적인 오류 타입을 디바이스 드라이버 영역에서 정의하는 모든 타입으로 설정가능하며 오류 타입에 따른 특성을 최대한 반영하여 실제 해당 오류가 발생한 것과 같은 효과를 발생시켰다. 이는 기존 도구와는 다르게 최대한 현실성과 도구의 신뢰성을 향상시킨 것이다.

지금까지 SORE의 ECC-CREATOR기능과 ERROR-CREATOR기능의 특징에 대해서 살펴보았으며 표 2에서 중요한 특징 3가지를 정리해보았다. ECC-CREATOR 기능은 ATA COMMAND[7]를 이용해서 원하는 섹터에 베드섹터를 발생시켜 섹터의 디지털 정보를 실제 손상시키는 기능이고 ERROR-CREATOR기능은 실제 섹터의 디지털 정보는 손상시키지 않으면서 물리적인 오류를 발생시키는 기능이다. 이것은 디바이스 드라이버 영역에서 에러 레지스터 값의 조작을 통해 원하는 섹터

표 1 SOAR의 오류 발생 기능과 기존 도구와의 비교분석

물리적인 오류 발생 도구	기능명칭	오류발생기법
SOAR	ECC-CREATOR 기능	ATA COMMAND[7]를 이용한 실제 베드섹터 발생
	ERROR-CREATOR 기능	디바이스 드라이버 영역에서의 필터링 기법을 이용한 가상의 물리적인 오류 발생
Fault Injection Tool[1]	물리적인 오류 발생기능	디바이스 드라이버 영역을 이용한 가상의 물리적인 오류 발생

표 2 ECC-CREATOR 와 ERROR-CREATOR 기능 비교

항목 및 기능	ECC-CEATOR 기능	ERROR-CREATOR 기능
역할	실제 베드섹터 발생 및 복구	원하는 물리적 오류 발생 및 복구
기법	ATA COMMAND 이용	에러 레지스터 값 조작 및 가상환경 구축
활용	읽기오류 발생	읽기/쓰기/인터페이스 오류 발생

의 디지털 정보는 손상시키지 않으면서 섹터 접근 유형(읽기/쓰기/인터페이스 접근)에 따라 설정된 물리적인 오류를 발생시킬 수 있다. ECC-CREATOR기능은 현실성이 강하면서 읽기 접근 오류만을 발생시키고 ERROR-CREATOR 기능은 실제 섹터의 디지털 정보를 손상시키지 않아 현실성은 ECC-CREATOR보다는 낮지만 여러 가지 접근 유형과 물리적인 오류 타입을 정할 수 있는 특징을 가진다.

SOAR의 ECC-CREATOR기능을 이용해서 실제 베드섹터가 발생되었음을 명확하게 확인할 필요성이 있다. 리눅스는 2가지 방법을 제공하는데 첫째는 공개 소스로 제공되어지는 S.M.A.R.T도구[8]를 이용해서 베드섹터 관련한 항목을 통해 확인 가능하고, 두 번째는 읽기 시스템 콜을 이용해서 베드섹터가 발생된 부분을 직접 읽어 커널 메시지를 확인하는 방법이 가능하다. 실제 저장장치에 베드섹터가 발생한 것이기 때문에 재 부팅 되거나 다른 시스템에 저장장치를 옮기더라도 베드섹터로 인한 읽기 실패는 계속 된다.

ECC-CREATOR기능은 실제 베드섹터를 발생하기 때문에 이를 치료할 수 있는 방법이 필요하다. 일반적으로 베드섹터를 치료하는 방법은 하드디스크의 리맵핑[9] 기능을 활용하는 방법이 있으며 이는 베드섹터가 발생한 부분에 쓰기 작업을 수행해서 하드디스크의 스페어 섹터와 맵핑 시키는 기능이다. SOAR도 이와 같은 기법을 이용해서 발생시킨 베드섹터를 복구한다. 이때 치료하고자 하는 섹터의 위치선정을 파일시스템에서 구분하는 블록단위로 계산한다면 2가지 단점이 발생한다. 첫 번째는 블록 하나당 섹터가 8개라면 베드섹터가 하나 발생했음에도 불구하고 8개 섹터 모두 쓰기 작업을 해야 하기 때문에 보다 많은 디지털 정보가 손실 된다는 것과 두 번째는 SOAR의 자체 기능이 아닌 파일시스템의 쓰기 시스템 콜을 사용해야 한다는 것이다. 하지만 SOAR는 이 두 가지 단점을 해결하였다. ATA COMMAND중 WRITE COMMAND를 이용해서 블록 단위가 아닌 섹터 단위의 쓰기를 가능하도록 만든 것이다. 이로 인해 특정 파일시스템의 도움 없이 섹터 단위의 쓰기를 SOAR의 독립적인 기능으로 탑재 하였다. 이와 같이 SOAR 기능 하나만으로도 어플리케이션, 파일시스템, 디바이스 드라이버의 강인성을 측정할 수 있는 기능을 갖춘 것이다.

디지털 정보의 손상을 막고 싶고, 읽기 오류뿐만 아니라 다른 오류 타입을 발생시키고 싶을 경우를 위해 SOAR는 ERROR-CREATOR 기능을 탑재 하였다. 이 기능은 디바이스 드라이버 영역에서 정의하고 있는 물리적인 오류[10]를 모두 발생 시킬 수 있다. 하지만 실제 저장장치에 물리적인 오류를 발생 시키는 방식이 아니기 때문에 최대한 해당 오류의 성질을 반영해야 한다. 따라서 해당 섹터에 접근 하는 방식에 따라 오류의 적용유무 및 행동을 구분 할 수 있도록 개발하였다. 이때의 행동 구분은 읽기 오류, 쓰기 오류, 인터페이스 오류로 구분되며, 원하는 디바이스 오류와 행동 구분을 선택하는 방식이다. 예를 들어 원하는 위치의 섹터에 행동구분을 쓰기 오류로 하고, 특정 오류 구분을 설정하였다면, 해당 섹터에 대한 읽기 수행은 가능하지만 쓰기 수행을 하고자 할 경우 설정된 특정 오류를 발생시킬 수 있도록 하였다.

ERROR-CREATOR기능의 오류 복구 방식은 설정된 오류 정보 리스트에서 오류 정보를 삭제 시키는 복구 인터페이스를 가지고 있다. 하지만 읽기 오류와 인터페이스 오류는 오류의 특성을 반영한 복구 방식을 추가적으로 가지고 있다. 읽기 오류 같은 경우 ECC-CREATOR 기능에서와 같이 해당 섹터에 쓰기 작업을 수행하게 되면 쓰기 작업이 수행된 영역에 대한 오류 정보를 자동으로 삭제시켜 리맵핑 된 것과 같은 효과를 주었으며, 일시적인 오류인 인터페이스 오류 같은 경우 첫 번째 접근하여 오류가 발생되었다면 자동으로 설정된 오류 정보를 삭제시켜 이후 접근에 대해서는 오류가 발생되지 않도록 하여 일시적인 오류 현상을 유도해냈다. 이는 최대한 오류의 특성을 반영하여 보다 현실에 가까운 상황을 유도한 것이다.

ERROR-CREATOR 기능은 디바이스 드라이버 영역에서 필터링 기법을 이용한 가상의 물리적인 오류 발생이기 때문에 ECC-CREATOR 기능과 같이 S.M.A.R.T 도구를 이용한 오류발생확인은 할 수 없다. 하지만 파일시스템을 통해서 오류가 발생한 섹터를 직접 접근하여 커널 메시지를 확인하는 방법은 가능하다. 실제 저장장치의 디지털 정보를 훼손하지 않고, 많은 종류의 오류를 발생 시킬 수 있기 때문에 시스템의 강인성을 검증하기에 매우 편리한 기능이다.

그림 2와 그림 3은 SOAR의 ECC-CREATOR기능과 ERROR-CREATOR 기능에 대한 시스템 구조상의 흐름도이다. 그림 2는 하드웨어 입장에서 각각의 오류 발생 흐름도를 나타낸 것으로 ECC-CREATOR기능은 운영체제 및 디바이스 드라이버 영역을 거치지 않고 ATA COMMAND를 이용해서 저장장치에 직접 접근한 후 실제 베드섹터를 발생시킨다. 그리고 ERROR-CREATOR기능은 디바이스드라이버 영역의 조작으로 인해 운영체제를 거쳐 들어오는 요청에 대해 해당 물리적인 오류를 디바이스 드라이버 영역에서 가상으로 발생시킨다.

그림 3은 소프트웨어 입장에서 오류 발생 흐름도를 나타낸 것으로 ECC-CREATOR기능은 운영체제와 디

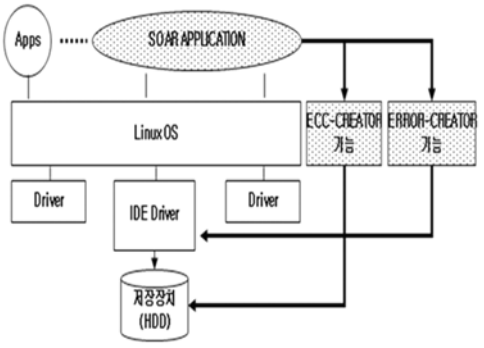

그림 2 오류 발생 기능에 대한 시스템 흐름도 (외적)

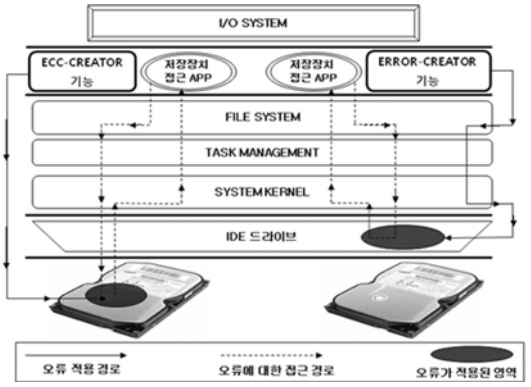

그림 3 오류 발생 기능에 대한 시스템 흐름도 (내적)

바이스 드라이버를 거치지 않고 직접 저장장치에 접근하여 물리적인 오류를 발생시키는 구조 이지만 ERROR-CREATOR기능은 파일시스템, 커널, 디바이스 드라이버를 모두 거쳐 가상의 물리적인 오류를 발생시킴을 알 수 있는 흐름도이다.

3.2 오류 발생 위치 선택 기법

ECC-CREATOR기능과 ERROR-CREATOR기능을 효과적으로 사용하기 위해서는 오류 발생 영역의 접근이 사용자 중심에 있어야 한다. 물론 사용자가 저장장치의 위치선택을 용이하게 하기 위해서 파일시스템이 구분하는 블록 번호와 LBA주소로 하고 있으나, 그 외에 특정 영역에 대해서는 간단한 선택 만으로 원하는 영역을 설정할 수 있도록 구성하였다.

저장장치에 물리적인 오류를 발생시킬 수 있는 위치 선정 기능은 표 3에서와 같이 4가지 방법이 있다. 일반적으로 생각할 수 있는 방법은 파일시스템에서 구분하는 블록번호와 계산된 LBA주소를 통해서 사용자가 직접 위치를 지정해주는 방식이다. 이 방식뿐만 아니라 3가지 특화된 기능을 SOAR에 적용해서 보다 사용자 편의성을 도모했다. 첫 번째 위치선정 방식은 RANDOM 기법이다. 이 위치선정 기능은 개발자나 SOAR의 사용자가 저장장치의 오류 발생을 원하는 일정 범위를 입력하고 이 범위 내에서 오류 발생 패턴과 발생 빈도를 입력 받아 임의의 위치에 오류를 발생 시켜 주는 기능으로 정형화되지 않은 오류 발생 패턴을 만들어준다. 예를 들어 저장장치의 정보와 함께 오류를 발생하고자 하는

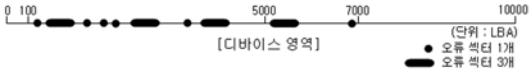

그림 4 랜덤 기법을 이용한 위치선정 예시

범위의 시작 위치를 LBA 100, 끝 위치를 LBA 7000으로 입력하고 오류발생 패턴의 묶음 수에 10을 입력한다. 그리고 발생 패턴의 개수를 4와 발생 패턴을 3이라고 주었을 경우, 오류는 저장장치의 LBA주소 100에서 7000번지 내에서만 발생 되며 총 10개의 묶음으로 발생된다. 이때 10개의 묶음 중 발생 패턴을 적용할 4개의 위치를 임의로 선택되게 된다. 임의로 선택된 4개는 연속된 3개의 섹터에 오류를 적용하게 된다. 오류가 적용된 섹터의 총 개수를 S라고 하고 물리적인 오류를 발생하고자 하는 시작 섹터위치를 i, 마지막 섹터위치를 j로 표시하였을 경우 $S(i,j)$는 다음과 같이 계산할 수 있다.

$$S(i,j) = (1 * (\text{오류발생 패턴의 묶음 수} - \text{발생패턴의 개수}) + \text{발생패턴} * \text{발생패턴의 개수})$$

이와 같은 방식으로 오류 발생 섹터의 개수를 계산하면 18개가 된다. 랜덤방식의 이용은 물리적인 오류 발생 패턴을 알 수 없는 상황을 재연하기 위해 유용한 방식이다.

저장장치에 물리적인 오류를 적용하기 위한 위치 선정의 두 번째 방법은 FAULT LIBRARY 방식이다. SOAR에는 파일시스템에서 구분하는 블록번호와 LBA 주소를 통해서 원하는 영역에 접근이 가능하도록 설계

표 3 오류 발생 위치선정 기법

	기법 명칭	설 명
오류발생 위치 선정 기능	일반	사용자의 위치 지정으로 오류 발생 및 복구 수행
	RANDOM	정형화되지 않은 오류 발생 (랜덤한 위치선정기법)
	FAULT LIBRARY	오류 정보를 담은 파일을 이용한 오류 발생
	FAULT FILE	원하는 파일의 원하는 블록에 오류 발생

표 4 FAULT LIBRARY기능 적용방법

기능	ECC-CREATOR 기능	ERROR-CREATOR 기능
파일 작성 규칙	device:/dev/hda1 sector:111,222,333 block:100,200,300 command:crash	device:/dev/hda1 sector:111,222,333 block:100,200,300 errortype:64 action:1 command:crash

가 되어 있다. 하지만 SOAR는 블록번호와 LBA주소를 통해서 여러 부분에 동시적으로 오류를 발생시키지는 못한다. 따라서 FAULT LIBRARY방식을 통해서 블록번호와 LBA주소를 동시에 입력한다면 2가지 인터페이스를 이용해서 여러 부분에 오류를 발생 시킬 수 있다. 이 방식은 TEXT 파일을 생성한 후 표 4에서와 같이 여러 정보를 담아 SOAR에 적용시키는 방식이다.

ECC-CREATOR을 이용하고자 할 경우 물리적인 오류를 발생시키고자 하는 디바이스 명과 다수의 섹터 번호, 블록번호를 입력한다. 마지막으로 오류발생인지 오류 복구인지를 명시하여 SOAR를 실행시키면 다수의 섹터와 다수의 블록번호에 물리적인 오류가 발생된다. ERROR-CREATOR인 경우는 디바이스 드라이버 영역에서 정의하고 있는 물리적인 오류 타입을 십진수 형태로 작성하고 읽기오류인 경우 1, 쓰기 오류인 경우 2, 읽기/쓰기 오류인 경우는 3으로 ACTION정보를 부여하여 SOAR를 실행시키면 된다.

이와 같이 FAULT LIBRARY 기능을 이용하면 ECC-CREATOR 기능과 ERROR-CREATOR 기능 모두 사용하면서 2개 이상의 위치를 블록번호와 LBA를 이용해서 선택할 수 있어 번거롭게 여러 번 SOAR를 수행하지 않아도 된다.

세 번째 위치선택 방식인 FAULT FILE 은 실제 파일의 경로와 블록의 인덱스를 통해 파일의 원하는 위치에 물리적인 오류를 발생시킬 수 있다. 이 방식을 이용하기 위해서는 SOAR의 독립적인 기능만으로는 불가능하다. 왜냐하면 특정 파일이 저장장치에 정확히 어느 위치에 있는지 파일시스템만이 알 수 있기 때문이다. 따라서 SOAR는 2가지의 기능을 파일시스템으로부터 가져와서 활용하였다. 첫 번째는 물리적인 오류를 발생하고자 하는 파일의 파일 디스크립터를 얻기 위해 캐릭터 디바이스의 INPUT/OUTPUT을 조정할 수 있는 IOCTL 기능을 활용하였다. 두 번째는 파일시스템의 IOCTL기능으로 얻어온 파일 디스크립터 구조체의 정보와 원하는 블록 인덱스 번호를 가지고 파일시스템의 RMAP 기능을 통해 실제 파일시스템 상에서의 위치를 알아낼 수 있는 기능을 활용하였다. 이 두 가지의 파일시스템 기능

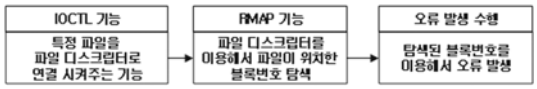

그림 5 FAULT FILE 방식을 이용한 오류 발생 흐름

을 활용해서 파일의 경로와 블록 인덱스 번호를 통해 물리적인 오류를 발생시킨다.

지금까지 오류발생 위치를 입력 받아 저장장치의 물리적인 오류를 발생 시키는 일반적인 인터페이스와 SOAR만의 특화된 기능 3가지를 살펴보았다. 이와 같은 기능을 이용하면 보다 SOAR의 물리적인 오류 발생 기능을 효과적으로 사용할 수 있다.

3.3 특정 파일시스템에 특화된 기능

물리적인 오류가 발생하면 디지털 정보가 손상 된다. 이러한 디지털 정보를 관리 하고 있는 시스템이 파일시스템이기 때문에 물리적인 오류에 가장 밀접한 관계가 있다. 따라서 SOAR를 이용해서 파일시스템의 물리적인 오류에 대한 강인성을 보다 쉽게 검증할 수 있도록 기능을 구성하였다.

현재 SOAR의 버전은 리눅스 기반이기에 가장 보편적으로 사용하는 EXT3 파일시스템[11]에 특화된 기능을 구성하였다. 하지만 다른 파일시스템에서도 이와 같은 역할을 할 수 있는 특화된 기능을 추가적으로 구성할 수 있다.

EXT3 파일시스템은 2가지의 중요한 영역으로 나눌 수 있다. 첫 번째 영역은 EXT3 파일시스템뿐만 아니라 다른 파일시스템에서도 중요시 여기는 메타 데이타 영역이다. 메타 데이타란 데이타의 구조화된 데이타로서 파일의 위치, 크기, 소유자, 허가권 등을 저장하고 있는 중요한 영역이다. 이 영역에 물리적인 오류가 발생한다면 다른 영역에 물리적인 오류의 발생으로 인해 발생되는 손상 그 이상이 될 것이다. 따라서 파일시스템 메타데이타 영역 각각의 세부 영역에 물리적인 오류를 발생시켜 파일시스템의 강인성과 신뢰성을 반드시 확인해야 한다.

EXT3 파일시스템은 메타데이타가 블록그룹 단위로 여러 개가 존재한다. 또한 포맷 옵션에 따라 각각의 블록그룹 내에 이와 같은 모든 세부 메타데이타들이 존재하지 않을 수 있다. 이와 같은 동적으로 변하는 메타데이타의 특성을 SOAR는 고려하고 있다. 따라서 SOAR

표 5 EXT3 파일시스템에 특화된 기능

	영역	설 명
EXT3 파일시스템	메타데이타 영역	세부 메타데이타 영역에 대한 오류 발생
	저널 영역	세부 저널 영역에 대한 오류 발생

사용자는 파일시스템의 포맷 옵션을 알 필요 없이 저장장치의 파티션 정보와 몇 번째 블록 그룹인지와 세부 메타데이타의 선택만으로 원하는 영역에 물리적인 오류를 발생 시킬 수 있도록 구성 하였다. 어떤 파일시스템이든 메타데이타는 매우 중요한 영역이다. 그래서 이와 같은 파일시스템의 중요한 영역을 SOAR에서 적극적으로 검증할 수 있도록 구성한 것이다.

두 번째로 중요한 EXT3 파일시스템 영역은 저널 영역이다. 저널이란 파일을 실제 수정하기 전 로그에 그 수정된 내용을 저장시켜 일관성이 깨진 상태가 되었을 때 로그만을 검사해서 깨진 일관성을 복구 시켜주는 기술이다. 이때 로그를 저장하고 있는 영역이 바로 저널 영역이다. EXT3 파일시스템의 저널 영역은 저널 슈퍼 블록, 저널 리복 블록, 저널 디스크립터 블록, 저널 커밋 블록으로 구분 하고 있다. 4가지의 상세 저널 영역 중 저널 슈퍼 블록을 제외한 블록들은 메타데이타처럼 고정된 위치가 없으며 동적으로 위치가 변한다. 따라서 SOAR는 실시간으로 어떠한 저널 블록이 저장장치의 어떠한 위치에 저장되는지를 자료구조 형태로 가지고 있기 때문에 SOAR를 통해 동적으로 변하는 저널 블록일지라도 원하는 저널 영역에 물리적인 오류를 발생시킬 수 있다.

이와 같은 기능들을 이용한다면 신뢰성과 강인성을 장점으로 하는 EXT3 파일시스템이 과연 물리적인 오류에 대해 어느 정도 수준까지 신뢰성과 강인성을 보장하는지를 간단한 테스트 프레임 워크를 통해 확인 할 수 있다.

4. SOAR의 사용자 인터페이스

지금까지 SOAR의 오류 발생 기능에 대해서 상세히 살펴 보았다. 하지만 효과적인 기능이 있음에도 사용자는 그 기능들을 효과적으로 사용하지 못하는 경우가 많이 있다. 왜냐하면 대부분의 사용자가 복잡한 매뉴얼을 보지 않고 직관적으로 사용하며 개발자의 주관이 개입된 인터페이스를 구성하고 있기 때문이다.

SOAR는 최대한 직관적인 사용자 중심의 2가지 인터페이스를 가지고 있다. 첫째, 윈도우 기반의 그래픽 환경에서 사용하기 편한 GUI 방식과 둘째, 임베디드 보드나 텍스트 기반의 환경에서 사용하기 편한 TUI방식으로 구성되어 있다.

4.1 그래픽 기반의 사용자 인터페이스

현대의 많은 프로그램들은 그래픽 기반의 사용자 인터페이스를 선호하고 편리성과 미적인 기능을 중시하기 시작했다[12]. 그래서 SOAR는 모든 기능을 사용자 직관적이고, 사용하기 가장 편리하도록 그래픽 기반의 인터페이스를 제공하고 있다.

리눅스 기반의 X윈도우 상에서 GTK 2.0 버전을 이용해서 개발하였다. SOAR의 특징과 사용자의 편의성을 고려한 화면 구성과 인터페이스를 갖추었다. 따라서 특별한 사용자 지침서가 필요 없이 사용자 직관적으로 사용하면 된다.

4.2 텍스트 기반의 사용자 인터페이스

임베디드 시스템은 PC와는 다르게 그래픽 유저 인터페이스를 갖추기가 힘들며 환경이 제한적이다. 따라서 SOAR는 텍스트 기반의 사용자 인터페이스를 제공해서 임베디드 시스템을 지원하였다. 사용자에게 가장 직관적인 인터페이스를 제공하기 위해서 명령어 입력이 아닌 사용자에게 선택할 수 있는 질의응답 형식의 인터페이스를 갖추었다. 이것은 사용자의 입력 실수를 최대한 고

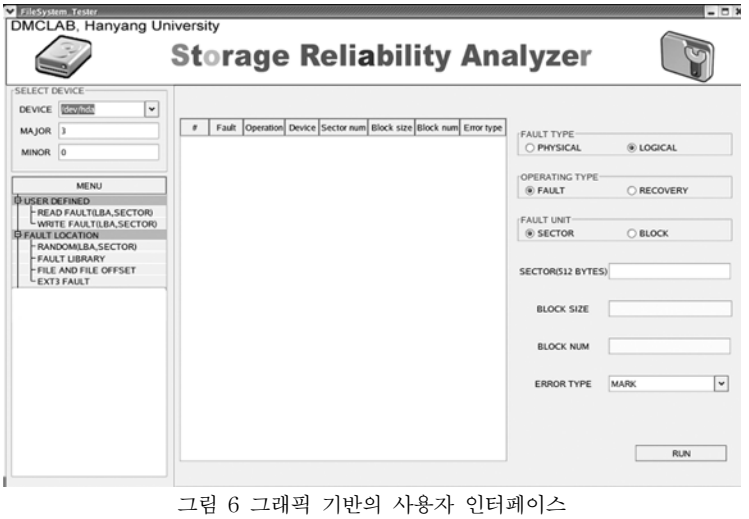

그림 6 그래픽 기반의 사용자 인터페이스

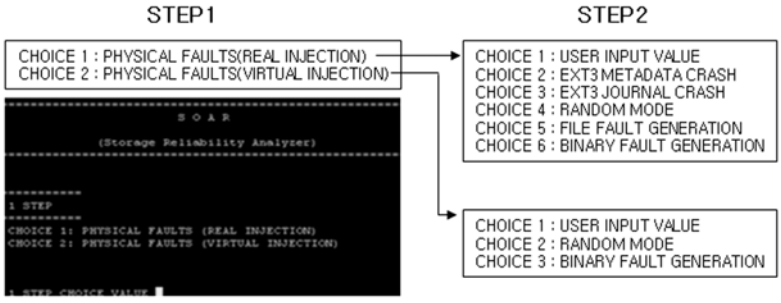

그림 7 텍스트 기반의 사용자 인터페이스

려한 것이다.

SOAR의 텍스트 기반 인터페이스가 사용자 입장에서 사용하기 더 편리할 수 있다. 이를 이용해서 텍스트 기반의 인터페이스가 필요한 임베디드 보드에서 보다 유용하게 사용될 것이다.

5. SOAR을 이용한 실험

SOAR의 물리적인 오류 발생 기법을 이용해서 어플리케이션과 커널 영역의 물리적인 오류에 대한 대처 수준을 확인하였다. 첫째, 물리적인 오류에 대해서 리눅스용 멀티미디어 플레이어와 MP3 플레이어가 어떠한 대처수준을 가지며 문제점은 무엇이 있는지 확인하였다. 둘째, 파일시스템을 벤치마킹하는 도구가 물리적인 오류로 오염된 영역을 테스트 영역이 되었을 경우 검증 결과와 문제점을 확인하였다. 마지막으로 SOAR의 파일시스템에 특화된 기능을 이용해서 파일시스템의 핵심 자료구조에 물리적인 오류를 발생시키고 워크로드를 실행했을 경우 어떠한 현상이 발생되며 문제점은 무엇인지 확인하였다. 이와 같은 실험을 통해서 지금까지 살펴본 SOAR의 기능과 활용 방법을 살펴볼 수 있었다.

5.1 리눅스용 멀티 미디어플레이어 및 MP3 플레이어

멀티미디어 서비스를 제공하는 대표적인 어플리케이션 타입은 멀티 미디어 플레이어와 MP3 플레이어이며 이를 이용하는 멀티미디어 파일은 작은 크기의 파일에서부터 상당히 큰 파일에 이르기 까지 그 범위는 다양하다. 다양한 크기의 멀티미디어 파일에 빈번히 발생되는 베드섹터와 같은 읽기 실패 섹터는 충분히 존재할 수 있다. 하지만 몇 개의 베드섹터는 사용자가 재생하여 플레이를 보는데 큰 문제를 주지 않는데도 소중한 멀티미디어 파일 전부를 재생하지 못한다면 개인에게 큰 실망과 곤란한 상황이 발생 될 것이다. 그래서 본 실험은 일반 리눅스 사용자들이 많이 사용하고 있는 어플리케이션을 중심으로 읽기실패에 대한 다양한 관점의 실험을 진행하였다.

베드섹터 즉, 읽기실패 부분을 플레이어가 읽으려고 했을 경우 여러 가지 반응을 보일 것이다. 읽기실패 시 화면 수준표와 에러 메시지 수준표를 표 6과 표 7에서와 같이 알파벳으로 구성하였으며 가장 좋은 반응 순서로 정의 하였다.

SOAR의 ECC-CREATOR 기능과 FAULT FILE 위치선정 기법을 이용해서 멀티미디어 파일의 위치를 선정한다. 파일의 블록 인덱스 번호가 0일 경우는 멀티미디어 파일의 헤더부분이고, 그 밖의 위치는 멀티미디어의 디지털 정보이다. 검증의 목적에 따라 위치를 선정

표 6 읽기실패 시 화면 수준표

기 호	상 황
A	재시도
B	읽기실패 부분 점프
C	정지화면
D	정상종료와 같은 상황
E	프로그램 자동 종료
F	프로그램 크래쉬

표 7 에러 메시지 수준표

기 호	상 황
a	정확한 에러메시지 출력
b	부정확한 에러메시지 출력
c	에러메시지 출력 없음

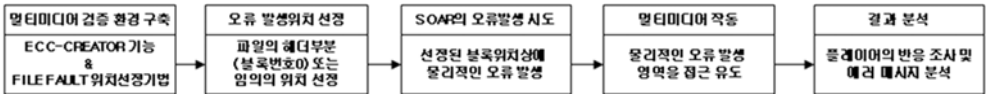

그림 8 멀티미디어 플레이어 테스트 프레임 워크

표 8 멀티미디어 플레이어의 읽기실패에 관한 결과표

멀티미디어 플레이어	화면 수준 결과	에러 메시지 정확도 결과	헤더 파일의 읽기 실패 시 에러메시지
Avifile	C	c	a
MTV	C	c	b
MPlayer-0.90	A/B	a	a

표 9 MP3 플레이어의 읽기실패에 관한 결과표

멀티미디어 플레이어	화면 수준 결과	에러 메시지 정확도 결과	헤더 파일의 읽기 실패 시 에러메시지
RealPlayer 7.0	E	b	b
MPlayer-0.90	D	c	a
Freeamp (ZINF)	D	c	b

한 후 물리적인 읽기 실패 오류를 발생시킨다. 그리고 플레이어를 작동시켜 읽기 실패가 발생되었을 경우 어플리케이션의 상황과 에러 메시지를 확인하였다.

첫 번째 실험은 멀티미디어 파일의 임의의 위치에 물리적인 오류가 발생한 경우 적절한 행동과 정확한 에러 메시지를 확인할 수 있는지 살펴보았다. Avifile과 MTV는 마지막 읽은 프레임에서 정지하게 된다. 하지만 MPlayer-0.9 같은 경우 Avifile 과 MTV와는 다르게 재시도를 수 차례 진행 후 읽기실패 영역을 넘어 다음 영역의 읽기를 수행하여 정상적인 화면을 계속 보여주었다. 에러메시지 역시 Avifile과 MTV는 어떠한 에러 메시지도 확인할 수 없었으나 MPlayer-0.9는 상세한 에러메시지를 확인할 수 있었다.

멀티미디어 파일의 정보를 담고 있는 헤더 정보 부분에 읽기 실패가 발생했을 경우 Avifile은 멀티미디어 파일의 헤더 포멧 정보에 문제가 발생하여 재생하지 못한다는 정확한 메시지를 보여주고 있으며 MPlayer-0.9 역시 파일을 읽는 모듈에서 헤더 정보를 읽지 못하였기 때문에 파일을 열 수 없음을 정확히 알려주고 있다. 하지만 MTV의 경우는 일반적인 경고 메시지만을 확인할 수 있었을 뿐 헤더 정보나 포멧에 문제가 발생했음을 나타내고 있지 않았다. 이와 같은 결과는 SOAR를 이용해서 멀티미디어 동영상 플레이어가 물리적인 오류에 어떠한 대처 수준을 가지고 있는지 명확히 확인 할 수 있었다.

이번 실험은 동영상 멀티미디어 플레이어와 같은 실험 방식으로 리눅스 사용자가 가장 많이 사용하고 있는 MP3 플레이어에 대한 실험을 진행하였다. 동영상 멀티미디어 플레이어와 같은 실험 방식이기 때문에 SOAR의 ECC-CREATOR 기능을 사용하고 여러 가지 위치 선정 기법 중 FAULT FILE을 사용해서 실제 물리적인 오류를 발생시키고 MP3플레이어를 실행시켜 보았다.

RealPlayer는 읽기실패 섹터를 읽으려고 할 때 프로그램이 종료되는 현상을 발견했다. 이와 같은 현상은 물리적인 오류에 매우 약함을 알 수 있으며 이때 발생되는 에러 메시지 또한 정확하지 않았다. 메시지는 “Bus error” 였고 헤더 파일의 읽기 실패가 발생되었을 경우 또한 같은 메시지를 보였다. 플레이 중에 읽기실패가 이루어졌을 경우 읽기실패가 이루어진 블록을 알려주고 재시도를 수행한 후 해당 영역을 점프 하는 것이 가장 이상적인 현상이지만 프로그램이 종료되고 에러 메시지 또한 추상적인 표현을 사용하고 있어 물리적인 오류에 매우 약함을 확인할 수 있었다. 그리고 헤더 파일의 읽기 실패 역시 정확하게 포맷에 문제가 있음을 알려줘야 하지만 일관되게 “Bus error” 메시지만을 보여주고 있는 것으로 봐선 에러 처리 루틴이 매우 협소하게 되어 있음을 짐작할 수 있다.

MPlayer와 Freeamp는 읽기 실패가 이루어지면 음악이 정상적으로 종료 된듯한 화면을 보여주고 있으며 메시지 역시 성공을 나타내고 있다. 이 현상으로 알 수 있는 사실은 읽기실패가 이루어졌을 경우 파일의 끝에 도달했다고 어플리케이션이 단정짓기 때문에 나타나는 현상이다. 헤더 파일의 읽기 실패가 되었을 경우도 두 어플리케이션은 비슷한 메시지를 보여주고 있다. MPlayer 경우는 해당 파일의 정확한 경로와 해당 파일의 헤더 부분 분석 시 읽기 실패 되었음을 정확히 말해주고 있다. Freeamp 경우는 팝업 창이 떠서 해당 파일을 읽으려고 하였으나 파일이 부패되었음을 알려주고 있다. 하지만 헤더 파일의 읽기 실패일 경우는 파일 형식에 문제가 있다는 메시지나 파일 문법적인 문제가 있음을 알려주지 않아 부정확한 메시지라고 표현하였다.

SOAR를 이용한 멀티미디어 관련 어플리케이션이 물리적인 오류에 대한 강인성을 검증하는 절차와 검증 결과를 살펴보았다. 검증 결과 아직 많은 어플리케이션은 물리적인 오류에 대한 대처에 소홀하고 있음을 명확히 결과표를 통해서 확인할 수 있었다. 이는 SOAR가 가지고 있는 기능을 명확한 결과물로 확인할 수 있는 계기였으며 더불어 개인의 디지털 정보가 점점 더 중요시되어가는 상황 속에서 어플리케이션도 물리적인 오류에 대한 대책이 반드시 필요하다는 것을 다시 한번 느낄 수 있는 결과이다.

5.2 파일시스템 벤치마킹 도구

파일시스템의 성능을 측정하는 벤치 마킹 도구는 테스트를 진행하게 되는 파일의 읽기 실패와 쓰기 실패에 대해서 보다 강인하고 정밀한 특성을 가지고 있어야 한다. 만약 특정 파일 시스템을 비교하기 위해서 벤치 마킹 도구를 통해 데이타를 얻었는데 한쪽 파일시스템에서 테스트를 진행하던 파일에 물리적인 오류 즉, 읽기

실패 오류나 쓰기 실패가 발생되어 결과값에 영향을 주게 된다면 벤치마킹하는 의미가 없어지는 것뿐만 아니라 잘못된 판단을 내릴 수 있게 된다. 그래서 파일시스템 성능을 테스트하기 위해 가장 많이 사용하고 있는 벤치마킹 도구를 선정하였으며, 벤치 마킹 도구의 읽기 성능을 측정하는 과정에서 읽기 실패가 일어나는 경우와 쓰기 성능 테스트를 하는 과정에서 쓰기 실패가 일어나는 경우를 만들어서 어떠한 결과가 나오고 있는지 확인하였다.

베드블록과 같은 읽기 실패가 LMBENCH[13]의 읽기 테스트 중에 발생하였을 경우, 어떠한 결과와 에러 메시지가 출력되고 있는지를 살펴보기 위해서 그림 9에서와 같이 몇 가지 단계를 거쳐야 한다. 우선 읽기 실패 테스트를 위해서 벤치마킹도구의 쓰기 테스트를 진행한다. 왜냐하면 읽기 성능 테스트의 특성상 쓰기 성능 테스트를 선행해야 가능하기 때문이다. 그래서 읽기 성능 테스트를 위한 파일이 생성된 후 SOAR의 읽기 실패 발생 도구인 ECC-CREATOR 기능과 위치 선정 방법 중 FAULT FILE 기법을 사용해서 쓰기 성능 테스트 이후에 생성된 파일에 베드섹터를 발생시킨다. 지금까지가 읽기 실패 테스트를 위한 환경 구축 단계였다.

쓰기 실패 테스트는 테스트 특성상 테스트 파일이 존재하는 것이 아니라 테스트 과정에서 파일이 생성되는 것이다. 따라서 일단 벤치마킹도구의 쓰기 성능을 측정한다. 측정 후 발생되는 파일이 저장장치의 어느 부분에 위치하고 있는지를 확인해야 하는데 이때 SOAR의 읽기실패발생 방법과 같이 FAULT FILE 기법을 이용해서 원하는 부분의 LBA주소를 알아 낸다. 그 후 테스트를 진행했던 저장장치 또는 파티션을 포맷해서 저장장치의 물리적인 오류가 모두 없도록 만든다. 그리고 SOAR의 쓰기 실패 발생 기능을 가지고 있는 ERROR-CREATOR 기능과 사용자 위치 지정 기법을 이용해서 곧 쓰기파일이 쓰여질 부분에 쓰기실패 오류를 적용시킨다.

표 10은 LMBENCH[13]의 읽기 실패 발생 유무에 따른 비교 분석표로써 읽기 실패가 이루어지게 되면 실패 직전까지의 내용을 분석해서 결과치를 보여준다. 하지만 읽기 실패가 이루어지기 직전까지의 시간을 이용해서 계산하게 된다면 지금보다 정확한 결과값이 나오겠지만, 읽기실패로 인해 지연되는 시간까지 모두 포함되기 때문에 부정확한 메시지를 사용자에게 알려주는 문제점이 발견되었다. 차라리 정확한 결과값을 보여주지 못한다면 결과메시지 자체를 보여주지 않는 편이 더 올바르다고 생각한다.

쓰기 실패의 경우는 어떠한 에러 메시지 없이 쓰기가 정상적으로 완료되었음을 알려주고 있으며, 그 결과값 또한 쓰기 오류가 없는 테스트의 경우와 비교했을 때 매우 근소한 차이를 보이고 있음을 그림 10에서 알 수 있다.

테스트한 환경은 읽기 성능을 측정했던 환경과 동일하며, 2GByte의 파일을 쓰는 동작이다. 정상적인 쓰기 작업이 이루어졌을 경우 56.20MB/sec의 속도와 쓰기 실패가 있는 경우 54.30MB/sec의 속도를 보이고 있다. 근소한 차이로 쓰기 실패가 있는 쓰기 작업이 약간 느리다. 어플리케이션에서는 어떠한 에러 메시지 또한 발생하지 않음을 확인하였다. 이와 같은 결과가 나타나는

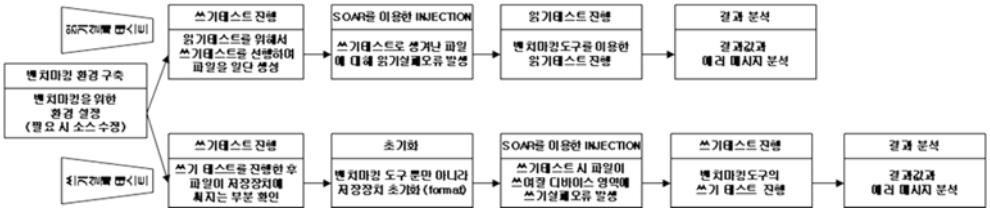

그림 9 벤치마킹 도구의 읽기/쓰기 실패에 따른 테스트 프레임 워크

표 10 읽기실패 발생 유무에 따른 LMBENCH의 결과분석표

	읽기오류가 없는 경우	4Mbyte 지점에 읽기오류
파일전부 읽기시도 성공유무	성공	실패 (4Mbyte까지만 읽기 성공)
읽기성능결과값	2097.15MB in 41.45secs 50.59MB/sec	4.0632MB in 21.0160secs 0.1933MB/sec
에러메시지	없음	Read: Input/output error
TEST 환경	CPU : Pentium4 2.4GHz, 메인 메모리 : 512MB HDD : Western Digital 80G, OS : Linux 2.4.30, EXT3	
TEST 방법	1Mbyte씩 2GByte 파일을 순차적으로 읽기 작업 수행	

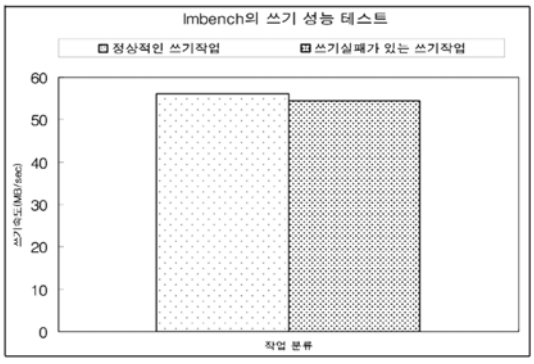

그림 10 LMBENCH의 쓰기성능 측정 결과

이유는 파일시스템 영역에서 쓰기 실패에 대한 어떠한 에러 값을 리턴 하지 않기 때문이며, 단순히 단방향으로 저장장치에 쓰기 작업만을 수행시키기 때문에 실제 저장장치의 물리적인 오류로 인해 쓰여지지 않더라도 어떠한 대처도 하지 않음을 확인하였다. 하지만 쓰기 실패가 있는 쓰기 작업이 약간 느린 이유는 디바이스 드라이버 영역에서는 물리적인 오류가 발생했음을 인지하여 에러 루틴을 거쳐 커널 메시지 형태로 알려주는 워크로드가 발생했기 때문이다. 만약 콘솔 모드의 테스트가 아닌 X윈도우 상태에서 이와 같은 테스트를 진행한다면 커널 메시지를 확인할 수 없을 것이고, 어플리케이션 조차 오류가 있음을 감지하지 못하기 때문에 잘못된 수치 값이 정확한 수치 값이라고 생각할 수 있는 문제점이 있다.

두 번째 실험을 진행할 벤치마킹 도구는 IOZONE[14]이다. LMBENCH와 실험 순서는 비슷하지만 IOZONE 특성상 읽기/쓰기 성능 테스트를 동시에 진행하는 워크로드를 가지고 있어 IOZONE의 소스를 수정할 필요가 있다. LMBENCH와 같이 IOZONE도 쓰기 성능 테스트를 먼저 하고, 읽기 성능 테스트를 하기 때문에 테스트가 전환되는 시점에서 break 포인트를 주고, 성능 테스트를 분리 시켰다. 그래서 LMBENCH에서 보여줬던 실험 순서와 동일하게 실험을 진행하였다.

IOZONE은 LMBENCH와는 다르게 읽기 실패가 발생했을 경우 결과값을 0으로 표현하고, 에러메시지를 보여주고 있다. LMBENCH에서 보았듯이 정확하지 않은 데이타를 보여주는 것 보다는 결과값을 0으로 표현하여 읽기 실패로 인해 읽기 성능평가가 이루어지지 않았음을 명확히 표현하였다. 그리고 에러메시지 또한 LMBENCH와 비교했을 때 단순히 읽기 실패만을 표현한 것이 아닌 읽기 실패가 이루어진 블록의 위치와 테스트를 진행하던 파일의 경로, 파일의 디스크립터 번호까지 정확히 나타내고 있다. 하지만 에러 메시지 내에 포함되지 않았어야 하는 표현이 섞여 있는 단점을 가지고 있으나, 전체적인 관점에서 보았을 때 물리적인 오류로 인한 읽기 실패가 발생하였을 경우 IOZONE은 LMBENCH보다 정확한 결과값을 표현하였으며, 에러메시지 또한 정확히 표현되고 있음을 확인할 수 있었다. 이는 저장장치의 베드섹터와 같은 물리적인 오류에 보다 강인하다는 의미이다. 쓰기 실패의 경우는 LMBENCH와의 결과와 상당수 비슷함을 확인할 수 있었다.

테스트 환경은 LMBENCH의 읽기 성능을 측정했던 환경과 동일하며 2GByte 파일을 씀으로 쓰기 동작을 수행하였다. IOZONE의 경우 정상적인 쓰기 작업이 이루어졌을 경우 57.39MB/sec의 속도였으며, 쓰기 실패가 포함되어 있는 경우는 54.32MB/sec의 속도를 보이고

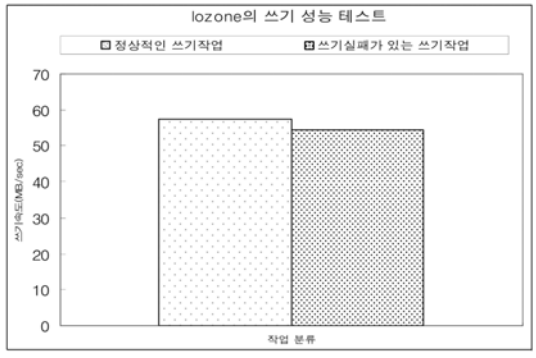

그림 11 IOZONE벤치마킹 도구의 쓰기성능 측정 결과

표 11 읽기실패 발생 유무에 따른 IOZONE의 결과분석표

	읽기오류 없는 경우	4Mbyte 지점에 읽기오류
파일전부 읽기 시도 성공유무	성공	실패 (4Mbyte까지만 읽기 성공)
읽기성능결과값	“Initial read” 50412.61 Kbytes/sec	“Initial read” 0
에러메시지	없음	Error reading block 31, fd=3 Read: Success /mnt/hdd/test.DUMMY.0 : no such file or directory
TEST 환경	CPU : Pentium4 2.4GHz, 메인 메모리 : 512MB HDD : Western Digital 80G, OS : Linux 2.4.30, EXT3	
TEST 방법	1024 Kbyte씩 2기가 파일을 순차적으로 읽기 작업 수행	

있어 매우 근소한 차이로 쓰기 실패가 있는 쓰기 작업이 약간 느리다는 결과값을 얻었다. 또한 어떠한 에러 메시지 또한 발생되지 않음을 확인하였다. 이는 LMBE-NCH에서 결과를 분석했듯이 쓰기 실패에 대한 어떠한 처리도 하지 않기 때문이다.

파일시스템 벤치마킹 도구의 차이점은 지금까지 벤치마킹 결과값과 간단한 내부구조뿐 이였다. 하지만 SOAR를 벤치마킹 도구에 적용함으로써 벤치마킹의 결과물뿐만 아니라 보다 정확한 결과값을 보여줄 수 있는지를 확인한 것이다. 이와 같은 발견은 일반 어플리케이션 보다 물리적인 오류에 더욱 민감해야 하는 벤치마킹 도구의 강인성을 SOAR를 통해서 증명한 것이다.

5.3 파일시스템

파일시스템은 디지털 정보의 저장을 위해서 저장장치와 유기적인 관계 속에서 중요한 역할을 담당하고 있다. 따라서 저장장치의 오류에 대해서 민감하게 반응 할 필요성이 있으며 디지털 정보의 보호까지 이루어져야 마땅하다. 이를 위해 저장장치에 물리적인 오류로 인한 읽기 실패와 쓰기 실패로 구분이 필요하며 파일시스템의 중요한 자료구조를 바탕으로 디지털 정보의 훼손상태를 조사하였다. 검증 대상으로는 리눅스에서 신뢰성과 견고성을 자랑하는 EXT3파일시스템과 JFS파일시스템을 선정하였고 각각의 결과표를 통해 분석을 진행하였다.

그림 12에서와 같이 간단하게 파일시스템의 강인성을 검증할 수 있는 테스트 프레임 워크를 제안하였다. 물리적인 오류로 인한 읽기/쓰기 실패 검증을 위해서는 무결한 하드디스크에 검증을 원하는 파일시스템의 포맷과 마운트를 수행한다.

읽기실패의 경우는 파일이나 디렉토리와 같은 디지털 정보를 읽지 못하는 경우이다. 그래서 검증을 위해 간단한 어플리케이션을 이용해서 디렉토리와 파일을 임의개 생성하는 워크로드를 수행한다. 그리고 SOAR를 이용해서 파일시스템의 중요 자료구조에 읽기실패를 발생시킨다.

쓰기 실패의 경우는 반대로 SOAR를 이용해서 파일시스템의 중요 자료구조에 쓰기 실패를 발생 시킨 후 워크로드를 수행해서 물리적인 오류의 특성에 맞는 파일시스템 접근을 유도한다. 그 이후에 파일이나 디렉토리와 같은 객체가 사라졌는지, 데이타가 손실되었는지, 데이타가 손상되었는지, 파일시스템의 언마운트와 마운트가 정상적으로 동작하는지를 확인해 봄으로써 파일시스템이 물리적인 오류로 인한 강인성을 간단하고 명확하게 검증하였다. 표 12, 표 13은 제안한 파일시스템 강인성 테스트 프레임 워크를 통해 신뢰성과 견고성을 자랑하는 EXT3 파일시스템과 IBM에서 개발한 JFS에 대해서 검증을 수행한 결과표이다.

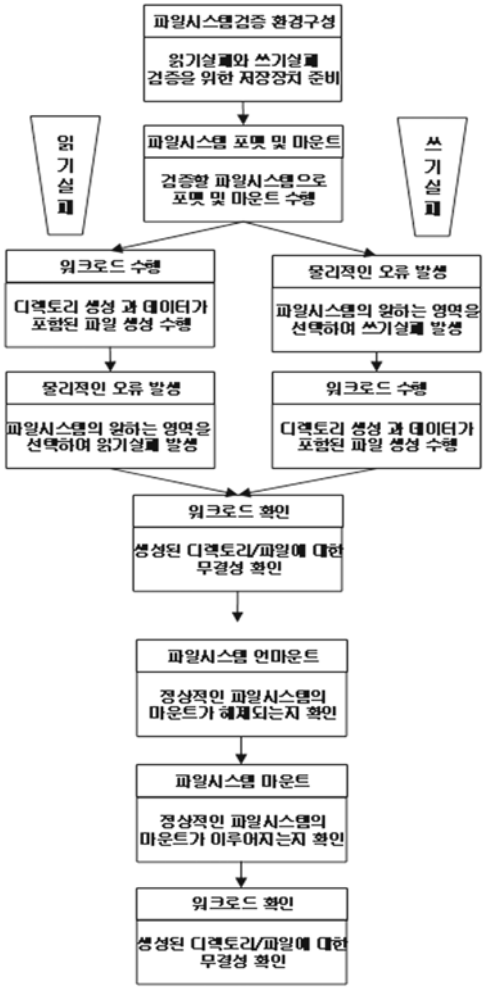

그림 12 파일시스템의 강인성 테스트 프레임 워크 진행도

EXT3 파일시스템은 물리적인 오류로 인한 슈퍼블록, 그룹 디스크립터, 아이노드 테이블에 읽기실패가 이루어지면 파일시스템의 마운트가 실패되는 특징과 그 밖의 자료구조에서는 마운트는 되지만 디지털 정보의 부패가 발생함을 확인할 수 있었다. 그리고 저널 영역의 경우는 파일시스템의 마운트가 안되는 심각한 현상이 발견되었다. 이 모든 현상은 베드섹터와 같은 물리적인 오류를 EXT3 파일시스템이 적절하게 대처하지 않고 있음을 확인하는 것이다. 아울러 슈퍼블록과 그룹 디스크립터와 같은 자료구조는 복사본을 여럿 두고 있지만 이를 적절하게 활용하지 않고 있음을 쉽게 발견할 수 있었다.

물리적인 오류로 인해 쓰기가 실패될 경우 데이타의 부패는 발생하고 있으나 파일시스템의 마운트와 언마운트는 이상 없이 진행됨을 확인하였다. 이는 파일시스템

표 12 EXT3 파일시스템의 읽기실패와 쓰기 실패에 대한 결과표

EXT3의 읽기실패 검증 결과표

파일시스템	자료구조	파일시스템 상태					
		데이터 손상	데이터 손실	객체 손실	마운트 실패	언마운트 실패	시스템 크래쉬
EXT3 (META)	superblock	x/-	x/-	x/-	o	-	-
	group descriptor	x/-	x/-	x/-	o	-	-
	indirect	o	o	-	-	-	-
	inode table	-	-	-	o	o	-
	inode bitmap	x	x	o	-	-	-
	data bitmap	o	o	-	-	o	-
EXT3 (JOURNAL)	Journal revoke block	x	x	x	o	x	-
	Journal commit block	x	x	x	o	x	-
	Journal super block	x	x	x	o	x	-
	Journal data block	x	x	x	o	x	-
	Journal checkpoint blo	x	x	x	o	x	-

EXT3의 쓰기실패 검증 결과표

파일시스템	자료구조	파일시스템 상태					
		데이터 손상	데이터 손실	객체 손실	마운트 실패	언마운트 실패	시스템 크래쉬
EXT3 (META)	superblock	-	-	-	-	o	-
	group descriptor	-	-	-	-	o	-
	indirect	-	o	-	-	-	-
	inode table	x	x	o	-	-	-
	inode bitmap	-	-	o	-	-	-
	data bitmap	o	o	-	-	-	-
EXT3 (JOURNAL)	Journal revoke block	o	o	o	-	-	-
	Journal commit block	o	o	o	-	-	-
	Journal super block	o	o	o	-	-	-
	Journal data block	o	o	o	-	o	-
	Journal checkpoint blo	o	o	o	-	-	-
기호 표현		o : 발생가능 , - : 발생하지않음, x : 검증할수없음					

표 13 JFS 파일시스템의 읽기실패와 쓰기실패에 대한 결과표

JFS의 읽기실패 검증 결과표

파일시스템	자료구조	파일시스템 상태					
		데이터 손상	데이터 손실	객체 손실	마운트 실패	언마운트 실패	시스템 크래쉬
JFS (META)	superblock	-	-	-	-	-	o
	control page	-	-	-	-	-	o
	Inode allocation map	-	-	-	-	-	o
	aggregate inode table	-	-	-	-	-	o
	block allocation map	-	-	-	-	-	o
JFS (JOURNAL)	Journal super block	x	x	x	x	x	o
	Journal data block	x	x	x	x	x	o

JFS의 쓰기실패 검증 결과표

파일시스템	자료구조	파일시스템 상태					
		데이터 손상	데이터 손실	객체 손실	마운트 실패	언마운트 실패	시스템 크래쉬
JFS (META)	superblock	-	-	-	o	-	o
	control page	-	-	-	o	-	-
	Inode allocation map	-	o	o	o	-	-
	aggregate inode table	-	o	o	o	-	-
	block allocation map	o	o	o	o	-	-
JFS (JOURNAL)	Journal super block	x	x	x	x	o	o
	Journal data block	o	o	o	o	o	-
기호 표현		o : 발생가능 , - : 발생하지않음, x : 검증할수없음					

에서 쓰기 실패에 대한 오류처리를 하지 않고 있음을 한눈에 파악하는 것이다. 그리고 이와 같은 검증을 진행하면서 EXT3 파일시스템의 시스템 파손은 발견하지 못했다. 이것으로써 읽기실패와 쓰기 실패에 대한 고려를 대부분 하지 않기 때문에 시스템이 멈추는 빈도는 극히 드물다는 예상을 할 수 있었다.

JFS 파일시스템의 물리적인 오류로 인한 읽기 실패 시 EXT3와 대체로 상반되는 결과를 보였다. 파일시스템의 마운트를 예로 든다면 특정 자료구조에 물리적인 오류로 인한 읽기 실패 시 EXT3와는 다르게 복사해놓은 복사본 자료구조를 원본 영역에 복사함으로써 읽기 실패에 대한 복구 동작을 충실히 하고 있어 실험 초반에는 검증의 신뢰성까지 의심해볼 정도였다. 하지만 자료구조의 원본과 복사본 모두에서 읽기 오류가 발생한다면 시스템이 파손됨을 확인하였다. 더불어 원본과 복사본에서의 읽기 실패가 동시에 발생하는 경우는 고려하지 않음을 확인할 수 있었다.

JFS의 쓰기 실패에 대한 검증 결과는 EXT3와는 다르게 파일시스템의 마운트가 실패됨을 확인할 수 있었다. 이것의 의미는 쓰기 실패에 대한 대처는 아닐지라도 기본적인 검사는 수행함을 알 수 있었다. 하지만 저널관련 자료구조에 쓰기 실패가 발생하면 시스템 파손을 제외한 모든 현상을 고루 확인하였다. 이는 JFS의 다른 자료구조에 비해서 저널 영역에 대한 쓰기 실패를 고려하고 있지 않음을 확인할 수 있었다.

지금까지 SOAR를 이용한 파일시스템 강인성 테스트 프레임 워크를 통해 신뢰성과 견고성을 자랑하는 EXT3 파일시스템과 JFS 파일시스템에 대해서 검증하고 결과

를 분석하였다. 가장 심각한 쓰기 실패에 대한 실험 결과에서처럼 디바이스 드라이버 영역에서는 쓰기 실패가 발생했음을 인지했음에도 불구하고 파일시스템 영역 뿐만 아니라 사용자 영역까지 실패 정보를 전달하지 않는 문제를 확인하였다. 이는 디바이스 드라이버 영역, 파일시스템 영역, 사용자 영역이 명확하게 계층 구조를 이루고 있으며 계층간의 원활한 정보교환이 이루어지지 않기 때문이다.

파일시스템의 물리적인 오류에 대한 강인성 여부를 확인 할 명확한 방법은 지금까지 존재하지 않았다. SOAR의 개발로 인해 복잡한 구조를 가지고 있는 파일시스템도 간단하고 명료하게 물리적인 오류에 대한 파일시스템의 분석을 진행할 수 있었다. 향후 기존에 개발된 파일시스템보다 견고하고, 논리적/물리적인 오류에 강인한 파일시스템이 개발된다면 SOAR를 통해 강인성 검증을 진행할 수 있을 것이다.

6. 결론 및 향후 연구방향

본 연구 개발을 통해서 저장장치에서 발생되는 물리적인 오류에 대한 심각성을 살펴 볼 수 있었다. 기존에 존재하는 저장장치의 무결성 검증 도구는 대부분 시스템의 논리적인 오류와 견고성만을 검증하고 있다. 그래서 저장장치에 발생되는 물리적인 오류에 대한 검증은 할 수 없으며 기존 관련 연구 또한 검증의 신뢰성을 보장할 수 없는 수준 이였다. 따라서 기존 도구와는 다른 방식의 물리적인 오류 발생 도구를 제안하였다. 그리고 이 도구를 이용해서 어플리케이션에서부터 파일시스템까지 물리적인 오류에 대한 검증과 검증을 위한 테스트 프레임 워크를 제안하였다.

도구의 기능과 제안을 뒷받침하기 위해 SOAR의 물리적인 오류를 발생시키는 2가지 기능과 저장장치의 원하는 영역에 접근하는 다양한 기법을 통해 멀티미디어 플레이어, MP3플레이어, IOZONE벤치마킹, LMBENCH 벤치마킹, EXT3 파일시스템, JFS파일시스템의 물리적인 오류에 대한 강인성을 검증하고 결점을 확인하였다. 이것은 보다 강인한 시스템으로 발전시킬 수 있는 방향을 제시한 것이다.

디스크 제조업체나 시스템의 강인성을 확인하고 싶은 그룹이 있다면 SOAR는 매우 편리하고 유용하게 사용될 수 있을 것이다. 지금까지 물리적인 오류에 대해서 심각성은 인정하였지만 실제 시스템 내에서는 적극적인 관심을 갖고 있지 않았다. 향후 SOAR를 이용해서 물리적인 오류에 강인한 시스템을 개발하고 SOAR를 이용해서 시스템의 강인성을 증명할 생각이다. 또한 SOAR의 현실적이면서 다각적인 물리적인 오류 발생 기법을 이용해서 기존에 출시된 시스템의 강인성을 확인할 수 있는 보다 체계적인 테스트 프레임 워크를 개발할 생각이다.

참 고 문 헌

[1] Jake Adriaens, Dan Gibson, "A Software Layer for IDE Disk Fault Injection," System Lacking Originality Workshop 2005.

[2] Daniel.P.Bpbet.and.Marco.Cesati, *Understanding the Linux Kernel 2nd chapter 13*: O'rilly, 2002.

[3] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Model-Based Failure Analysis of Journaling File Systems," presented at Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05) Brighton, United Kingdom, 2005.

[4] P. Vijayan, N. B. Lakshmi, A. Nitin, S. G. Haryadi, C. A.-D. Andrea, and H. A.-D. Remzi, "IRON file systems," presented at Proceedings of the twentieth ACM symposium on Operating systems principles (Brighton, United Kingdom, October 23 - 26, 2005), Brighton, United Kingdom, 2005.

[5] Hard disk register & ide, "http://hem.passagen.se/communication/ide.html".

[6] Silberschatz., Galvin, and Gagne, OPERATING SYSTEM CONCEPTS chapter 2 WILEY, 2003.

[7] The Guide to ATA/ATAPI documentation. "http://www.stanford.edu/~csapuntz/ide.html".

[8] S.M.A.R.T., "http://www.die.net/doc/linux/man/man8/smartd.8.html".

[9] Sector remapping, "http://www.storagereview.com/guide2000/ref/hdd/perf/qual/featuresRemap.html".

[10] Harddisk register value & physical fault define "/usr/src/linux/include/linux/hdreg.h".

[11] "EXT3fs Home Page, "http://sourceforge.net/projects/ext3sj".

[12] A. Joakim, V. Jonny, F. Peter, and K. Johan, "GOOFI: Generic Object-Oriented Fault Injection Tool," in Proceedings of the 2001 International Conference on Dependable Systems and Networks: IEEE Computer Society, 2001, pp. 83-88.

[13] "lmbench home page, http://www.bitmover.com/lmbench/."

[14] "IOzone Filesystem Benchmark, http://www.iozone.org."

김 영 진

2005년 평택대학교 컴퓨터과학과 학사 2005년 8월~2007년 8월 한양대학교 공과대학 전자컴퓨터통신공학부 석사. 2007년 9월~현재 NHN Corp. 관심분야는 파일시스템, 임베디드 시스템, 멀티미디어 시스템, 운영체제

원 유 집

1990년 서울대학교 자연과학대학 계산통계학과 학사. 1992년 서울대학교 자연과학대학 계산통계학과 석사. 1997년 University of Minnesota 전산학 박사. 1997년~1999년 Performance Analyst, Intel Corp. 1999년 3월~현재 한양대학교 공과대학 전자컴퓨터통신 공학부 부교수. 관심분야는 멀티미디어 시스템, 운영체제, 컴퓨터 네트워크

김 락 기

2006년 부경대학교 전자통신컴퓨터공학과 학사. 2006년~2008년 한양대학교 공과대학 전자컴퓨터통신공학부 석사. 2008년~현재 삼성전자. 관심분야는 운영체제, 파일시스템, 임베디드 시스템