FMAX(Flexible MetadatA eXtention) Filesystem: 리눅스 커널 환경에서 메타데이터 파일시스템의 설계와 구현

FMAX(Flexible MetadatA eXtention) Filesystem: Design and Implementation of Linux Kernel-Level Metadata Filesystem

박치현(Chi-Hyun Park)*, 김우철(Woo-Cheol Kim)**, 노홍찬(Hong-Chan Roh)**, 박상현(Sang-Hyun Park)***,

요 약

운영체제에서 파일시스템은 파일에 대한 접근, 관리 등을 총괄하는 중요한 부분이다. 최근 대용량 정보 저장 장치의 발달과 함께 저장하고 관리해야 할 파일의 양이 증가하면서, 많은 데이터 혹은 파일들을 효과적으로 다룰 수 있는 파일시스템의 필요성이 증가하고 있으며 이에 대한 연구도 이루어지고 있다. 이런 파일시스템은 주로 메타데이터(metadata)를 활용하는데, 커널 환경(kernel-level)에서 이러한 파일시스템에 대한 연구는 사용자 환경(user-level)에서의 연구보다 상대적으로 미흡하다. 본 논문에서는 리눅스(Linux) 커널 환경에서 메타데이터를 통합적으로 관리함으로써 효율적인 파일 탐색이 가능한 파일시스템을 제안한다. 또한 사용자가 스크립트 언어를 통하여 직접 파일시스템의 메타데이터 관리 기능을 제어할 수 있도록 한다. 특히 파일시스템과 데이터베이스 관리 시스템을 융합할 수 있는 기본적인 환경을 제공하여 기존의 파일시스템이 제공하지 못한 다양한 기능이 개발될 수 있도록 한다. 본 논문은 커널 환경으로 이러한 파일시스템을 설계하고 구현하기 때문에 가상 파일시스템(Virtual File System)과의 연결을 위한 별도의 인터페이스가 필요 없으며 적재(mount) 명령을 통하여 쉽게 파일시스템을 사용할 수 있다. 커널의 많은 제약조건을 해결하면서 설계되고 구현된 본 논문의 파일시스템은 파일 연산에 대해서 기존 연구들이 제안하였던 파일시스템보다 전반적으로 향상된 성능을 보여준다.

주제어 : 파일시스템, 커널 환경, 리눅스, 메타데이터, 스크립트언어, 멀티디렉토리

Abstract

A filesystem which deals with file accesses and management is an essential part of an operating system. As the mass storage devices has been developed and the amount of files has increased recently, the necessity of filesystem that can effectively deal with huge data or files

* 연세대학교 컴퓨터과학과 석사과정
** 연세대학교 컴퓨터과학과 박사과정
*** 연세대학교 컴퓨터과학과 교수 (교신저자)

has been increasing and the research regarding this have been carried out. This filesystem generally utilizes metadata, but the research regarding this filesystem with kernel-level implementation fashion is less than the research with user-level implementation fashion. In this paper, we propose the filesystem which is able to search files efficiently by managing the metadata with kernel-level implementation fashion. Additionally, in this proposed filesystem, users can directly control the function of metadata management by script language proposed in this paper. Especially, our filesystem provides a basic environment that merges a filesystem with database management system in order to support new functions which have not been implemented by the traditional filesystems. Using our filesystem, there is no need to have an interface to connect virtual file system with a filesystem. Thus user can easily use the filesystem only with a mount command. Compared with the filesystems suggested in previous research, our filesystem implemented under many kernel restrictions demonstrated enhanced overall performance.

Key Words : Filesystem, Kernel-level, Linux, Metadata, Script Language, Multi Directory

1. 서론

파일시스템의 가장 중요한 기능은 신뢰성 높은 파일 저장 기능과 안정적이면서 빠른 파일 접근 또는 탐색 기능이다. 예를 들어 주로 리눅스에서 사용되는 EXT3 파일시스템은 EXT2 파일시스템보다 파일 저장의 신뢰성을 높인 파일시스템이며 FAT 파일시스템의 파일 저장의 신뢰성을 강화한 파일시스템이 NTFS이다[11,12,13,14]. 이러한 새로운 파일시스템들을 통해서 파일 저장에 대한 신뢰성은 높아졌지만 이러한 파일시스템들은 트리 구조(tree structure)에 기반한 단순한 탐색 방식만을 제공하고 있다.

최근 컴퓨터의 사용이 늘어남에 따라서 저장해야 할 파일의 종류가 다양해지고, 그 양 또한 급격하게 증가하고 있다. 예를 들어 디지털 카메라나, MP3 플레이어와 같은 많은 정보 기기들의 발달로 인하여 개인용 컴퓨터가 처리하고 관리해야 할 파일들이 증가하고 있고, 컴퓨터의 하드웨어 성능이 크게 발달함으로써 처리할 수 있는 파일의 양 또한 증가하고 있다. 이러한 환경에서 전통적인 파일시스템의 방법인 단순하게 계층적인 트리 구조를 이용해 파일을 탐색하는 것은 비효율적이다. 즉 현재의 파일시스템의 경우에는 파일을 탐색하려면 명확하게 경로명을 알아야 하는데 저장된 파일들이 많아질수록 사용자가 찾는 데이터의 저장 경로명을 기억할 수 없어 탐색하기 어렵기 때문이다.

따라서 최근에는 파일시스템에 효율적인 파일 탐색 기능을 제공하기 위한 연구들이 많이 진행되고 있다. 특히 강력한 데이터 탐색 기능을 제공하는 데이터베이스 시스템의 관련 기술을 이용해서 데이터베이스 시스템과 파일시스템의 통합을 목표로 각각의 장점을 결합하는 연구들이 많다. 하지만 데이터베이스 시스템과 파일시스템은 서로 다른 목적으로 개발된 데이터 저장 방식이기 때문에 데이터베이스 관련 기술을 직접적으로 바로 파일시스템에 적용하기 힘들다. 두 시스템간의 대표적인 차이점은 다음과 같다.

첫 번째, 데이터베이스 시스템과 파일시스템은 데이터의 탐색 방식이 다르다. 현재 사용되는 대부분의 파일시스템은 디렉토리를 사용하여 계층적인 트리 구조를 만들 수 있고, 디렉토리 안에 데이터, 즉 파일들을 저장

할 수 있다. 따라서 파일들을 탐색할 때 디렉토리로 이루어진 계층적인 트리 구조를 탐색해야 한다. 실제 리눅스 파일시스템은 디렉토리 엔트리(entry)를 모델화한 dentry 구조체를 사용하여 계층적으로 저장 경로를 탐색한다. 그러나 이와 다르게 데이터베이스 시스템은 일반적으로 테이블 구조에 데이터를 저장하고 관리한다. 테이블 구조를 사용해서 데이터를 저장한다면 계층적으로 데이터를 탐색할 필요가 없을 뿐만 아니라, 질의(query) 기반으로 조건을 만족하는 데이터를 테이블 구조에서 쉽게 찾을 수 있기 때문에 효율적인 데이터 탐색이 가능하다.

두 번째, 두 시스템은 데이터 탐색을 위해 필요한 메타데이터의 양이 다르다. 파일시스템은 위에 언급했듯이 dentry 구조체를 사용하여 트리 구조에서 파일을 탐색하기 때문에 탐색에 필요한 메타데이터의 양이 적다. 하지만 데이터베이스 시스템은 질의 기반으로 탐색을 하고, 테이블 구조에서 속성에 해당하는 값은 모두 탐색에서 질의 조건으로 사용될 수 있기 때문에 그와 관련된 메타데이터가 많을수록 탐색이 용이해진다. 그렇기 때문에 두 시스템을 통합하여 파일시스템이 좀 더 효율적으로 질의 기반으로 파일을 탐색을 할 수 있도록 하기 위해서는 메타데이터를 추출할 수 있는 모듈이 필요하다. 또한 추출한 메타데이터를 효율적으로 저장, 관리할 수 있는 구조도 필요하다.

마지막으로 두 시스템은 구현 환경이 다르다. 파일시스템은 운영체제의 한 부분으로서 커널 환경에서 모두 구현되어 있으며, 운영체제와 독립적으로 동작할 수 없기 때문에 복잡한 구조를 가지고 있으며, 디바이스 드라이버, 메모리 관리 부분, 프로세스 관리 부분 등과 같은 운영체제의 다른 부분과 밀접한 관련이 있다. 그러나 데이터베이스 시스템은 대부분 사용자 환경에서 독립적으로 실행되는 일종의 응용프로그램의 성격을 갖는다고 볼 수 있다. 따라서 파일시스템에 데이터베이스 시스템의 기능들을 융합시킨다는 것은 커널 환경에서는 쉽지 않은 일이다. 그래서 기존 연구들은 대부분 사용자 환경에서 파일시스템을 개발하는 방법을 사용해왔다. 그러나 파일시스템은 근본적으로 커널 환경에서 모듈화 되어 운영체제의 한 부분이 되어야 하기 때문에 본 논문에서는 커널 환경의 많은 제약 조건하에서 파일시스템과 데이터베이스 시스템의 장점을 융합할 수 있는 방법을 제시한다. 본 논문에서 제시하는 파일시스템은 메타데이터를 유연하게 활용하면서 데이터베이스의 장점을 융합하기 때문에 FMAX(Flexible MetadatA eXtention) 파일시스템이라고 한다.

FMAX 파일시스템은 응용프로그램이나 사용자 단계에서 부가적으로 메타데이터를 관리하는 부하를 줄여줄 수 있을 뿐만 아니라, 기존 파일시스템이나 데이터베이스 시스템과 같이 통합적으로 파일들을 관리할 수 있는 기능을 갖는 것이 큰 특징이다. 또한 메타데이터를 관리하여 효율적인 파일 탐색이 가능하도록 하며 스크립트 언어를 통하여 사용자와 응용프로그램 단계에서 파일시스템의 메타데이터 관리 기능을 조정할 수 있는 기능을 제공하는 것도 FMAX 파일시스템의 특징이다. 본 논문에서는 프로토타입(protype) 형태로 FMAX 파일시스템을 실제 커널 환경에서 설계하고 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 논문들이 제안한 파일시스템들의 특징들과 연구의 흐름에 대해서 서술하고, 3장에서는 본 논문이 제시하는 메타데이터 기반의

데이터베이스가 융합된 파일시스템의 특징에 대해서 살펴본다. 4장에서는 이러한 파일시스템의 설계 및 구현 방법에 대해 서술한다. 5장에서는 FMAX 파일시스템과 기존 파일시스템과의 성능을 실험으로 비교하고, 마지막 6장에서는 결론과 향후 연구방향을 제시한다. 본 논문의 구성에서 본문을 3장과 4장으로 나눈 이유는 제안하고자 하는 파일시스템의 논리적인 설계 부분과, 실제 구현에 관련된 부분을 나누어 설명함으로써 각각의 부분을 좀 더 명확히 제시할 수 있기 때문이다.

2. 메타데이터 기반의 파일시스템

메타데이터의 활용을 기반으로 해서 파일의 탐색 기능을 강화하고, 파일시스템과 데이터베이스 시스템의 기능을 융합하여 새로운 패러다임을 갖는 파일시스템에 대한 연구들의 대표적인 예는 다음 세 논문이다.

파일시스템에 유연한 파일 탐색 기능과 일부 데이터베이스의 기능을 접목시켜 하나의 파일시스템을 제안한 가장 첫 번째 연구는 Semantic File System[1]이다. [1]은 전통적인 파일시스템이 단지 하나의 물리적인 파일 저장 경로만을 탐색할 때 사용하는 것과 다르게, 파일과 디렉토리의 관계를 파일들이 가지는 속성과 그에 대응하는 값에 따라 변경하여 사용자가 탐색할 때 이용할 수 있도록 하였다. 즉 파일에 대한 메타데이터를 사용하여서 파일을 탐색할 때 사용자가 질의를 주면 질의를 만족하는 파일들을 그 결과로 보여주고 이를 통해 파일을 탐색할 수 있도록 하는 구조이다. 이렇듯 [1]은 기존의 트리 구조 접근 방식의 전통적인 파일시스템을 내용기반 접근(contents based access)으로 변환한 것이 가장 큰 특징이다. 또한 [1]은 기존에도 연구되고 있던 이러한 파일 혹은 데이터 탐색 기법을 처음으로 파일시스템 영역으로 확장시켰다. 때문에 지금까지 많은 논문들이 [1]을 참조하고 있다. 또한 사용자가 파일들의 어떠한 속성들을 추출할 것인지 결정할 수 있도록, 즉 어떠한 메타데이터를 추출할 것인가를 나타내는 트랜스듀서(transducer)라는 것을 제안하였다. 하지만 트랜스듀서는 구현 내용이 추상적으로만 나와있으며 전체 파일시스템이 커널 환경에서의 구현이 아니라 사용자 환경에서 구현되었다는 한계가 있다.

Logic File System[2]은 [1]의 방법과는 다르게 파일의 탐색 기능을 발전시켰다. [1]은 단지 파일들에 대한 속성과 그에 해당하는 값에 따라서 질의가 들어왔을 때 그에 만족하는 파일들을 탐색할 수 있도록 해주었는데, [2]는 [1]을 보다 더 확장하여 부울구조(boolean organization)에 바탕을 두어서 파일을 탐색할 수 있도록 하였다. [2]의 특징은 사용자 환경의 파일시스템이고 메타데이터를 저장할 때 버클리 데이터베이스 (Berkeley Database)를 사용한다는 점이다. 즉 파일시스템 자체에서 메타데이터를 저장할 수 있는 장소를 가진 것이 아니라 데이터베이스를 사용하여 이러한 일을 한다.

Richer File System[3]은 사용자가 정의한 파일들 사이의 관계를 링크 개념을 사용해서 나타내었다. 또한 데이터베이스의 한 특징인 트리거(trigger)의 개념을 파일시스템에 적용하여 특정 파일에 대해서 트리거를 정의할 수 있도록 하였다. 파일 트리거란 특정 파일에 대해 사용자가 미리 정의한 조건을 걸어놓고 그 조건을 만족하는 연산이 파일에 행해질 시에 사용자에 의해 미리 정의된 동작을 수행하게 하는 것이다.

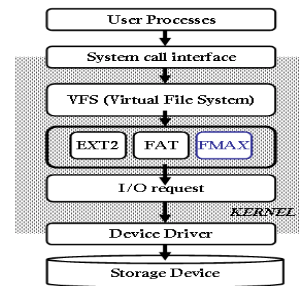
위에 언급한 세 논문은 가장 대표적으로

알려진 논문들이고, 모두 공통적으로 파일시스템의 탐색 구조를 변경할 수 있는 기능을 제공해주며, 파일시스템이 데이터베이스의 기능들을 부분적으로 수행할 수 있도록 그 기능을 확장하고 있다. 이는 본 논문에서도 가장 큰 목표로 하고 있는 부분이다.

이 밖에 위 논문들이 제안하고 있는 디렉토리를 사용하는 유연한 탐색 기법에 대해 연구한 [4]와 같은 것이 있다. 또 메타데이터를 활용하는 WinFS[16]라는 차세대 파일시스템을 윈도우 운영체제에서 제안했지만 아직 활발히 연구되고 있지는 않다. 또한 본 논문이 제안하는 파일시스템을 설계하기 위해 메타데이터를 활용하는 방법이 필요한데 이에 대한 연구들도 이루어 졌다. [6]에서는 메타데이터를 이용한 파일시스템에 대한 연구가 진행되었다. 이 논문은 인터넷 응용프로그램에서 현재 많이 사용되고 있는 태깅(tagging)을 이용하여 그래프 형태로 메타데이터 정보를 제공하는 TagFS(Tag File System)를 제안하였다. [7]에서는 메타데이터를 분류하는 연구가 진행되었고, [8]에서는 데이터베이스 관리 시스템에 기반하여 메타데이터를 관리하는 방법들에 대한 연구가 진행되었다. 또 [9]에서는 웹 기반의 메타데이터 파일시스템에 대한 연구가, [10]에서는 전통적인 파일시스템상에서 메타데이터를 관리하는데 생기는 문제점에 관한 연구가 진행되었다.

3. 메타데이터를 활용하는 FMAX 파일시스템의 특징과 구조

본 논문이 제안하는 FMAX 파일시스템은 커널 환경에서 설계되었기 때문에 EXT2나 FAT 계열 파일시스템처럼 커널 모듈화(kernel module)가 되어서 하나의 독립적인 파일시스템으로 사용될 수 있다. 즉 가상 파일시스템과 완전하게 인터페이스가 호환되므로 사용자는 기존 파일시스템을 사용하듯이 적재(mount) 과정을 거치기만 하면 쉽게 본 논문이 제안한 파일시스템을 사용할 수 있도록 설계되었다. 전체적으로 FMAX 파일시스템의 커널 상에서의 위치는 그림 1에 나타나 있다. 3장에서는 실제 파일시스템의 구현에 관련되어 언급하기 전에 FMAX 파일시스템의 논리적인 설계 구조와 특징에 대하여 언급하겠다.



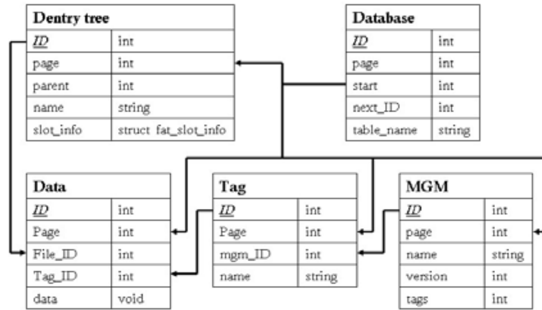
<그림 1> 커널 환경에서의 파일시스템 구조

3.1 FMAX 파일시스템에서 메타데이터 저장 공간과 관리 형태

FMAX 파일시스템이 기존 파일시스템들과 구분될 수 있는 가장 큰 점은 메타데이터를 직접 수집하고 관리하는 파일시스템이라는 것이다. 로컬(local) 컴퓨터 환경에서 사용되고 있는 파일시스템들은 대표적으로 NTFS나 FAT, EXT2 등이 있는데 이들 파일시스템은 자체적으로 메타데이터를 추출하거나 관리하는 기능이 없다. 물론 단순하게 파일의 이름이나, 갱신 시각 같은 정보들은 inode 구조체와 dentry 구조체, 파일 구조체에 들어 있다. 하지만 이러한 정보는 메타데이터들 중 일부

분에 불과하다. 예를 들어 음악 파일의 경우 재생 시간은 물론, 작곡자, 가수, 인코딩 형식 같이 사용자나 응용프로그램이 메타데이터로 정의할 수 있는 데이터들이 파일에 많이 포함되어 있다. 따라서 각 파일 종류 별로 어떠한 메타데이터를 추출할 것인지 결정할 수 있고 실제 추출 할 수 있는 기능을 파일시스템이 지원할 수 있다면, 응용프로그램이나 사용자가 별도로 파일들에서 메타데이터를 추출해야 하는 부하를 줄일 수 있을 뿐만 아니라 여기서 추출된 메타데이터를 통합적으로 관리할 수 있기 때문에 이를 활용하여 기존 파일시스템이 지원하지 못했던 기능들을 확장하는데 적용할 수 있다.

본 논문에서는 독립적인 스키마를 갖는 관계형 데이터베이스의 형태로 메타데이터를 저장, 관리 한다. 메타데이터의 관리에 테이블 구조를 적용한 이유는 기본적으로 테이블과 테이블 사이의 조인(join) 연산을 통해서 어떠한 파일에서 어떠한 메타데이터를 추출하였으며 실제 그 값은 얼마인지 모두 알 수 있는 구조이면서 최대한 중복을 줄이면서 분산해서 정보들을 저장할 수 있기 때문에 효율적인 탐색을 가능하게 해주기 때문이다. 또한 모든 파일을 계층적인 구조로만 관리하는 것이 아니라 관계형 구조로 변환시켜 관리하면 데이터베이스의 기능을 확장하여 적용시킬 수도 있기 때문이다. 따라서 이러한 목표를 최대한 달성할 수 있는 논리적인 스키마를 생각했고, 표 1을 보면 알 수 있듯이 5개의 테이블로 분산하여 저장하면 가장 효율적으로 메타데이터를 관리할 수 있다.



<표 1> 메타데이터 관리, 저장 구조의 관계 스키마

표 1이 나타내는 스키마 구조를 갖고, FMAX 파일시스템이 적재된 상태에서 예를 들어 'yesterday.mp3' 라는 음악 파일을 저장한다고 하자. 그리고 이때 파일의 읽기 횟수(read_count), 쓰기 횟수(write_count), 읽는 크기(read_size), 쓰는 크기(write_size)가 정의된 'mgm_fs' 라는 MGM이 등록된 상태라고 한다면 표 1과 같은 스키마를 갖는 데이터베이스 테이블들이 생성된다. 현재 파일시스템이 적재된 디렉토리를 루트로 한 채 그 아래 저장되는 디렉토리 혹은 파일들은 각각 고유의 ID를 가지고 계층 관계를 Parent 속성을 통해서 유추할 수 있는 구조로 테이블 Dentry tree에 저장 된다. 또한 'yesterday.mp3' 라는 파일에 대해서 추출해야 하는 메타데이터들인 위에 언급한 네 가지 속성들은 Tag로 표현되며 'yesterday.mp3' 파일의 Tag들은 테이블 Tag를 보면 알 수 있다. 테이블 Data에는 'yesterday.mp3' 파일에 대한 읽기 크기 값, 쓰기 크기 값 등 추출해야 하는 4가지 tag에 대한 값이 data라는 속성에 저장된다. 위에서 언급했듯이 분산되어 저장된 테이블들 간의 조인 연산을 통하여 특정 메타데이터에 관련된 파일들을 찾을 수 있으며, 파일들 간의 종속 관계뿐만 아니라 연관 관계도 메타데이터를 통하여 파악할 수 있다.

3.2 MGM(Metadata Generation Module)의 구조와 원리

FMAX 파일시스템에서 메타데이터를 추출하는 기능은 MGM이 담당한다. MGM은 메타데이터를 추출, 생성하는 모듈로서 여러 조건 혹은 상황에 따라서 파일시스템이 가변적으로 메타데이터를 추출, 생성하기 위해 만들어졌다. MGM은 파일 종류별로 만들 수 있고, 같은 종류의 파일이라도 추출해야 하는 메타데이터의 종류가 다르다면 다른 MGM을 가질 수 있다.

MGM은 하나의 모듈 단위로 이루어져 있기 때문에 여러 개의 MGM들이 연결되어 사용될 수 있다. FMAX 파일시스템은 파일 입출력, 파일 생성, 파일 삭제 같은 연산에 의하여 메타데이터가 변경될 수 있을 경우에 이벤트를 발생시킨다. 그리고 이때 발생한 이벤트에 따라 MGM을 통하여 메타데이터를 추출 혹은 생성 하여 메타데이터 저장 테이블에 필요한 값을 저장하거나, 필요한 경우 값을 삭제하는 일을 하게 한다.

MGM의 가장 큰 특징은 사용자가 직접 그 기능을 정의할 수 있다는 점이다. 이러한 특징은 Semantic File System[1]에서 제안한 것으로 사용자가 트랜스듀서를 만들어서 파일 종류 별로 다르게 메타데이터를 추출할 수 있도록 한다는 것에서 아이디어를 얻었다. 이 아이디어를 바탕으로 FMAX 파일시스템은 실제 커널 영역에서 처리할 수 있는 메타데이터 추출 모듈을 개발한다는 것이 Semantic File System[1]과의 차이점이다. 또한 MGM들을 리스트 형태로 관리되는데 그 이유는 하나의 파일이라도 추출해야 하는 메타데이터에 대한 정의가 꼭 하나의 MGM에 국한되지 않아도 되기 때문이다.

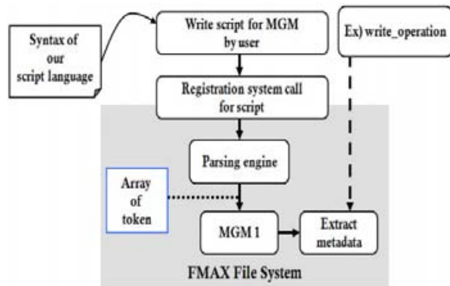
3.3 MGM을 위한 스크립트 언어

사용자가 직접 파일시스템의 특정 기능을 조종할 수 있다면 메타데이터의 추출, 수집 과정을 통제할 수 있을 뿐만 아니라, 개인의 컴퓨팅 환경에 따라서 최적화된 상태로 메타데이터를 관리할 수 있는 장점이 있다. 따라서 기존의 연구들을 보면 공통적으로 파일시스템의 이러한 기능을 조종할 수 있는 방법들이 연구되었다. 그러나 파일시스템이 커널 환경으로 구현된다면 사용자 환경에서 쉽게 정의하는 트랜스듀서처럼 파일시스템의 특정 기능을 조종할 수 없다. 따라서 이를 가능하게 할 수 있는 구조를 제공해 주어야 한다. 본 논문에서는 이런 기능을 관리할 수 있는 자체 스크립트 언어를 정의하여 이를 활용하여 사용자가 직접 추출할 메타데이터에 관한 정의와 이벤트 발생 시점 등을 MGM에 등록할 수 있게 한다.

자체 스크립트 언어는 C언어의 구문을 활용하였고 크게 TAG에 대한 부분과 이벤트의 처리에 대한 부분으로 나누어져 있다. 파일의 읽기, 쓰기 등 파일에 대한 접근이 발생할 경우 FMAX 파일시스템은 이벤트가 발생한다고 정의 하는데, 스크립트 언어에서는 이러한 다양한 이벤트의 발생에 대하여 각각 하나의 함수로 만들고, 인자도 줄 수 있다. 또 각 이벤트 함수 안에서는 그 이벤트 함수가 불렸을 경우 tag 값의 변경에 대한 정의가 있다. 예를 들어 MP3 파일의 경우 WRITE 라는 이벤트 함수가 불릴 경우에는 헤더의 어느 부분을 읽어서 title이라는 tag에 제목을 넣도록 정의되어 있다. 스크립트 언어에는 C언어의 구조체와 같은 모습으로 tag들에 대한 정의도 있다. 즉 tag들의 변수명은 무엇인지, 어떠한 변수 타입인지가 TAG에 정의 되어있

다.

FMAX 파일시스템에는 사용자가 직접 작성한 스크립트 언어를 처리해줄 수 있는 스크립트 언어 분석 엔진이 있으며 사용자 환경에서 이 스크립트 언어를 등록하는 시스템 콜을 호출하면 분석 엔진은 스크립트 언어를 해석해서 MGM에 분석 내용을 등록해준다.



<그림 2> 스크립트 언어를 등록하는 과정

그림 2를 보면 스크립트 언어 분석 엔진은 미리 정의한 문법에 의거해서 사용자가 작성한 스크립트 코드를 해부하여 의미 단위로 나눈다. 의미 단위로 나뉘어진 토큰들 중 어떠한 메타데이터를 추출해야 하는지 정의할 수 있는 토큰들은 2차원 배열형태로 저장된다. 이후 메타데이터를 추출해야 하는 쓰기 연산 같은 것이 호출되면 토큰들이 저장된 MGM을 사용하여 정의한 메타데이터를 추출한다. 사용자 수준에서 MGM을 정의하는 실제 스크립트 언어의 예를 보면 그림 3과 같다. 그림 3은 MP3 파일들에 대해서 적용 가능한 MGM의 예이다. MP3 파일의 경우는 분석된 스크립트 코드에 의거하여 추출해야 하는 메타데이터, 즉 MP3 파일의 제목 혹은 재생 시간과 같은 tag 들을 각 파일의 헤더 부분을 분석하여 얻는다. 본 논문의 파일시스템은 일정 포맷을 갖는 파일의 헤더를 분석하는 부분이 있다. 그러나 아직 FMAX 파일시스템은 프로토타입 단계로 실제 MP3파일의 헤더를 분석하지는 않으며, 실제 MP3 파일의 헤더 부분의 앞부분과 유사한 구조를 임의의 파일에 만든 후 그 파일의 헤더를 분석한다. 그러나 FMAX 파일시스템은 이러한 작업을 가능하게 하는 기본 구조는 모두 설계된 상태이기 때문에 실제 MP3 파일의 헤더 부분 분석과 메타데이터의 추출은 향후 연구에서 확장할 예정이다.

그림 3의 MGM_MP3는 앞서 설명하였듯이 TAG와 이벤트 처리 부분으로 나뉘고 READ와 WRITE 이벤트에 대하여 어떠한 메타데이터를 추출할 것인지 나와있다. 이 예제에서는 MP3파일이 등록될 경우, 즉 파일이 새롭게 생긴 경우에 메타데이터를 추출하며 파일을 읽을 경우 메타데이터를 추출하게 된다. 이때 추출되는 메타데이터는 제목(title), 가수이름(singer), 재생시간(playtime) 이렇게 3가지 이다.

```
TAG
{
 STRING title;
 STRING singer;
 INT playtime;
}

EVENT CREATE ()
{
 title = "";
 singer = "";
 playtime = 0;
}

EVENT READ (INT pos, BYTE data, INT data_len)
{
}

EVENT WRITE (INT pos, BYTE data, INT data_len)
{
 IF ( data[0] != 42 )
 {
  title = ""; singer = ""; playtime = 0;
 }
 ELSE
 {
  INT id; id = 0;
  INT[3] str_start;
  INT[3] str_len;
  INT start; start = 2;
  INT current; current = 2;
  WHILE( current < data_len )
  {
   IF( data[current] == 32 )
   {
    str_start[id] = start;
    str_len[id] = current - start;
    start = current + 1;
    current = current + 1;
    id = id + 1;
   }
   ELSE
   {
    current = current + 1;
   }
  }
  title = to_str(data, str_start[0], str_len[0]);
  singer = to_str(data, str_start[1], str_len[1]);
  playtime = to_int(data, str_start[2], str_len[2]);
 }
}
```

<그림 3> 사용자 수준의 스크립트 언어의 예제 (MGM_MP3)

3.4 멀티디렉토리를 사용한 유연한 파일 탐색

파일시스템에서 추출된 메타데이터를 활용하는 방법 중 대표적인 것은 기존 논문에서도 언급했던 파일 탐색이다. 본 논문에서는 유연한 파일 탐색 또는 접근을 할 수 있도록 임의의 디렉토리를 생성해 원하는 파일만 접근하도록 해주는 방법을 이용하였는데 이를 본 논문에서는 '멀티디렉토리' 라고 한다. 파일 탐색 기능의 개선은 파일시스템이 가지고 있는 근본적인 구조적 문제, 즉 트리 형태의 저장 구조의 한계를 극복하려는 시도이다. 사용자나 응용프로그램은 파일시스템이 트리구조로 되어있기 때문에 파일을 탐색할 때 원하는 파일을 찾을 때까지 원하지 않는 디렉토리들까지 탐색해야만 한다. 따라서 효율적인 탐색이 어렵기도 하고, 파일이 저장된 경로명을 정확하게 모를 때에는 파일 탐색이 어려울 수 있다. 따라서 이러한 파일 탐색의 단점을 질의 기반으로 쉽게 탐색을 하는 데이터베이스의 기능을 적용하여 해결할 수 있는 방법이 멀티디렉토리이다.

멀티디렉토리는 계층적 구조로 저장되어있는 파일을 메타데이터를 활용하여 질의 기반으로 효율적으로 탐색할 수 있도록 해준다. 즉 사용자 질의에 의해 선택된 파일만 빠르고 쉽게 접근할 수 있도록 해준다. 멀티디렉토리를 생성하게 되면 질의에 해당하는 파일들이 물리적인 경로가 아닌 임의로 만들어진 디렉토리 구조, 즉 질의의 결과로 묶이는 파일들끼리 하나의 디렉토리에 포함이 되고, 질의들 사이에 종속관계가 있다면 하나의 멀티디렉토리 안에 하위 멀티디렉토리가 생성되는 구조가 만들어 진다. 이렇게 되면 사용자나 응용프로그램들은 원하는 파일의 물리적인 경로를 알고 있지 못해도 찾으려고 하는 파일을 접근할 수 있는 질의만 준다면 쉽게 파일에 접근할 수 있다. 따라서 효율적인 탐색이 가능해 진다.

4. FMAX 파일시스템의 구현 및 설계 접근 방법

파일시스템을 설계 및 구현하는 대표적인 방법 중 하나는 사용자 환경에서의 개발하는 것이고, 다른 하나는 커널 환경에서의 개발하는 것이다. FMAX 파일시스템은 이 두 가지 접근 방법 중 후자를 선택하였다. 커널 환경에서의 개발과 사용자 환경에서의 개발은 각자 장단점이 있다. 4장에서는 FMAX 파일시스템의 실제 구현에 관련된 부분을 제시한다.

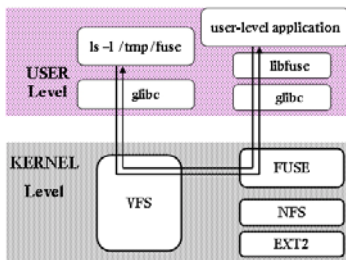
4.1 사용자 환경에서의 파일시스템 설계 및 구현

사용자 환경의 파일시스템은 FUSE(File System in User Space)[17]나 PerlFS[18]와 같은 사용자 환경 파일시스템 인터페이스를 사용하여서 사용자 수준에서 커널 프로그래밍의 제약사항[1]에 얽매이지 않고 쉽게 원하는 파일시스템을 설계한다. 또한 이러한 사용자 환경 파일시스템 인터페이스에 버클리 데이터베이스 혹은 포스트그레스 데이터베이스(Postgres Database) 같은 데이터베이스를 사용해서 메타데이터를 저장하고 관리할 수 있게 해준다. 그림 4는 FUSE를 사용하는 사용자 환경의 파일시스템을 간단하게 나타낸다. FUSE는 EXT2나 NFS[5]와 같이 하나의

[1] 커널 코드는 전체가 하나의 커다란 프로그램과도 같으므로 새로운 파일시스템을 설계하기 위해 기존의 스케줄링 커널 함수들이나 여타의 기존 함수들 또는 데이터구조와의 호환성을 우선하여 고려해야 한다.

커널 모듈로서 커널에 적재 된다. FUSE는 파일시스템이 내부적으로 처리해주어야 하는 작업들을 사용자 환경에서 일반 응용프로그램을 구현하듯이 사용자가 손쉽게 구현할 수 있도록 하여 파일시스템의 구현 편의성, 유지보수성을 높여준다. 이러한 사용자 환경에서 동작하는 파일시스템의 모듈이 이미 등록된 FUSE의 커널 모듈과 연계되어 동작하므로 응용프로그램이나 사용자 측에서는 일반 파일시스템을 사용하는 것과 동일한 인터페이스로 접근할 수 있게 된다.

FUSE를 사용해서 파일시스템을 설계하고 구현하는 연구는 많이 이루어 지고 있지만, 이 방법은 파일시스템의 사용자 환경 모듈이 커널 모듈과 연계하여 동작하는 과정에서 부하가 발생하여 궁극적으로 EXT2나 FAT 같은 완전히 커널 안에서 모듈화 된 파일시스템보다는 성능이 떨어질 수 있다. 이는 Logic File System[2]의 실험 결과에서도 나타났다. 따라서 FUSE를 사용하는 파일시스템의 설계는 연구의 측면에서는 많이 활용되고 있지만 기존 파일시스템처럼 실제 상용화 가능한 파일시스템의 형태로는 개발되지 못하고 있다.



<그림 4> FUSE를 사용한 사용자 환경 파일시스템 구조

4.2 커널 환경에서의 파일시스템 설계 및 구현: FMAX 파일시스템

3장에서 언급한 내용을 바탕으로 FMAX 파일시스템 설계의 특징을 다음과 같이 정리할 수 있다.

■ 커널 환경에서 파일시스템을 설계하여 완전히 모듈화된 형태를 갖는다. 따라서 사용자나 응용프로그램들이 적재 명령을 통하여 쉽게 사용할 수 있다.

■ 전통적인 파일시스템에 새로운 패러다임을 적용한다. 단순히 계층적인 트리구조로 파일을 저장하는 것뿐만 아니라 관계형 구조로도 파일을 저장한다.

■ 파일시스템이 직접 메타데이터를 추출, 저장, 관리할 수 있도록 한다. 이는 사용자나 응용프로그램이 직접 메타데이터를 관리하는 부담을 줄일 수 있다.

■ MGM의 핵심 부분을 사용자가 직접 정의할 수 있도록 스크립트 언어를 제공하여 메타데이터의 추출을 사용자가 정의할 수 있도록 한다.

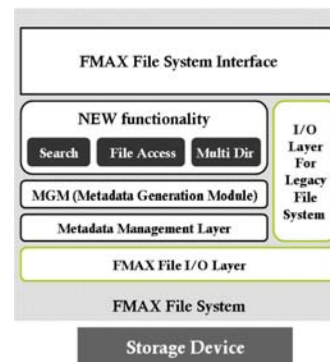
■ 멀티디렉토리를 사용하여 효율적인 파일 탐색이 가능하도록 한다. 이 과정에서 사용자의 질의를 처리하여 멀티디렉토리를 만든다.

FMAX 파일시스템은 위에 언급한 특징을 잘 나타낼 수 있도록 설계되었다. 위에 언급한 특징들의 공통점을 추려본다면 파일시스템과 데이터베이스의 통합 패러다임을 적용했다는 것을 알 수 있다. 그렇지만 커널 환경에서 이러한 기능을 하는 파일시스템을 설계하고 구현하는 일은 매우 복잡하다. 이는 커널의 특징인 모놀리틱(monolithic) 때문이다. 쉽게 얘기해서 커널은 하나의 커다란 프로그램으로 내부의 작은 부분의 수정이 다른 부분에도 영향을 미칠 수 있다. 즉, 커널은 시스템 운영에 필요한 많은 서비스 루틴

(routine)을 포함하지만 위에 언급한 특징들을 아무런 제약 없이 제공할 수 있는 환경을 제공해 주지 않는다. 커널의 많은 서비스 루틴들은 복잡하게 얽혀 있으며, 따라서 커널의 작은 부분을 수정하는 것도 커널 코드에 대한 높은 분석력을 요구한다. 이러한 이유 때문에 기존 연구들은 커널 환경에서 파일시스템을 설계하는 것보다 사용자 환경에서 설계하는 것이 효율적이라고 판단하였다. 그러나 커널 환경에서 파일시스템을 설계한다면 사용자 환경에서의 설계 보다 성능이 좋을 수 있을 뿐만 아니라, 완벽히 모듈화된 형태로 파일시스템을 사용할 수 있는 장점이 있다.

FMAX 파일시스템은 기본적으로 FAT 계열 파일시스템을 바탕으로 설계되었다. FMAX 파일시스템은 FAT 계열 파일시스템에서 활용할 수 있는 루틴들을 최대한 활용하면서 FMAX 파일시스템만의 특징을 나타낼 수 있는 기능들을 융합하였다. 이렇게 설계한 이유는 FAT 계열 파일시스템을 활용하는 것이 최대한 시간과 비용을 줄이면서 개발할 수 있기 때문이다. 또한 FAT 계열 파일시스템의 최적화된 루틴들을 잘 활용하면 안정적인 파일시스템을 구축할 수 있기 때문이다. 그러나 FMAX 파일시스템의 가장 핵심적인 부분인 데이터베이스의 특징들을 융합시키는 영역은 FAT 계열 파일시스템의 루틴을 활용할 수 없는 부분이 많고, 기존에 사용되는 파일시스템 중 참고할 수 있는 것이 없다. 따라서 새로운 기능을 구현하기 위해 커널 내부 코드에 대한 최대한의 정확한 이해를 바탕으로 설계가 이루어 졌다. 또한 이러한 이유 때문에 추후에 시스템의 성능 개선에 관한 연구와 설계 방법 변경에 대한 연구가 수행될 수 있을 것이다.

FMAX 파일시스템의 내부 구조는 그림 5와 같다. 그림 5와 같이 FMAX 파일시스템은 FAT 계열의 파일시스템이 제공하는 기본적인 루틴을 활용하면서 FMAX 파일시스템이 고유하게 정의한 루틴을 사용한다. FMAX 파일시스템은 자체적으로 모든 루틴을 정의하고 있지 않기 때문에 FAT과 VFAT 파일시스템이 FMAX 파일시스템에게 이러한 루틴을 제공해줄 수 있도록 수정이 필요하다. 따라서 FMAX 파일시스템을 사용하기 위해서는 VFAT와 FAT 파일시스템이 모듈로 등록된 상태에서 FMAX 파일시스템 모듈을 등록해야 한다.



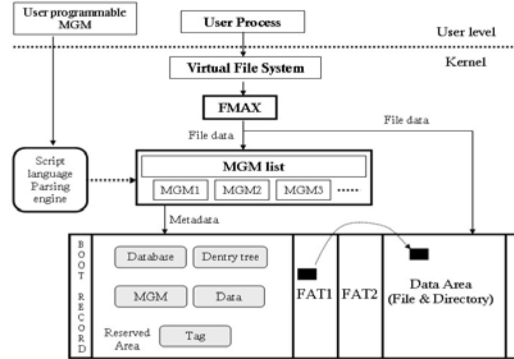
<그림 5> FMAX 파일시스템 전체 구조

4.2.1 메타데이터 저장, 관리 공간의 구현 및 설계

3.1에서 언급한 논리적인 구조를 갖는 메타데이터를 저장하고 관리할 수 있는 구조는 필연적으로 기존 파일시스템보다 성능 저하를 발생할 수 있게 하는 요인이 된다. 직관적으로 보면, 하나의 파일 연산에 대해서 추가로 메타데이터 관련 연산들도 수행되어야 하기 때문이다. 따라서 메타데이터를 단순히 하나의 파일 형태에 저장하고 일반적인 파일 입출력 연산(I/O operation)을 사용하는 것은 비효율적이다. 또한 메타데이터에 대한 입출

력 연산은 시스템 내부적으로도 빈번하게 발생하기 때문에 연산의 부하를 줄일 수 있는 효율적인 저장 구조와 저장 공간이 필요하다. 본 논문에서 제안하는 파일시스템은 FAT 파일시스템의 구조에서 사용하지 않는 예약된 영역(reserved area)에 메타데이터를 관리하는 방법을 적용하였다. FAT 파일시스템에서 예약된 영역에 대한 접근은 별도의 읽기(read), 쓰기(write)함수를 사용하지 않고 페이지(page) 단위로 입출력 연산을 통하여 이루어 진다. 페이지 단위의 입출력 연산의 장점은 일반적인 읽기, 쓰기 연산이 복잡한 루틴을 거쳐서 이루어지는 반면, 좀 더 간결한 루틴을 사용하는, 즉 좀 더 하위 단계의 입출력 연산을 바로 호출할 수 있다는 것이다. 따라서 이 방법은 좀 더 효율성을 높일 수 있는 입출력 방법이다. 그렇지만 이 방법은 페이지 단위로 입출력 연산을 해주어야 하기 때문에 일반 파일들을 저장하는 데는 좋은 구조는 아니다. 예약된 영역은 파일시스템을 적재하기 전에 그 크기를 조절할 수 있기 때문에 공간의 부족 문제는 대비할 수 있다. 예약된 영역의 공간 부족해질 만큼 메타데이터가 생성된다면, 일반 파일을 저장하는 영역에 테이블 구조로 메타데이터를 저장할 수 있다. 다만 빈번한 메타데이터 관련 입출력 연산으로 인하여 효율성은 떨어질 수 있다. 또한 오랫동안 사용되지 않은 파일에 대한 메타데이터들은 삭제한 후 그 파일이 다시 사용될 때 다시 수집되게 하는 방법을 적용한다면 공간을 효율적으로 사용할 수 있다. 아직 이 문제에 대해서는 최적의 해결 방안을 FMAX 파일시스템에 적용하지는 못하였지만, 향후 연구에서 중점적으로 다룰 예정이다.

본 논문에서는 예약된 영역에 그림 6과 같은 구조를 통하여 3.1에서 언급한 표 1에 나와있는 것과 같은 5개의 독립적인 논리적 스키마를 갖는 데이터베이스로 메타데이터를 저장, 관리하도록 구현하였다.



<그림 6> 메타데이터 관리, 저장 구조

4.2.2 MGM의 구현 및 설계

```
struct meta_mgm_slot{
        struct meta_slot_data     slot_data;
        char*                     name;
        int                       signature;
        bool                      status;
        char*                     script;
        int                       tag_size;
        struct meta_tag_slot**    tags;
        struct mgm_var_table      tag_var_table;
        struct mgm_func_table     func_table;
};
```

<그림 7> MGM 구조체

3.2에서 언급한 MGM은 그림 7이 나타내는 구조체와 같이 정의된다. 그림 7에 나와있듯이 MGM은 각각 고유의 이름을 가지고 있으며 고유의 ID인 signature를 가지고 있다. 또한 사용자가 직접 스크립트 언어를 통해서 MGM의 기능을 정의할 수 있기 때문에 script에 대한 포인터 변수도 가지고 있다. 또 MGM에서 추출해야 할 메타데이터의 종류를 나타내는 태그에 대한 변수도 있으며, 각 MGM은 meta_slot_data구조체를 통해서 리스트로 연결되어 있다.

실제 MGM이 어떠한 단계를 통해서 파일에서 메타데이터를 추출할 수 있는지는 그림

8이 나타내는 MGM 내부 알고리즘을 보면 알 수 있다. 알고리즘 1은 읽기 연산을 예로 들어서 나타낸 것이다. 읽기 연산과 마찬가지로 쓰기 연산이나 파일 생성 연산 또한 같은 과정을 거치게 된다.

FMAX 파일시스템의 읽기, 쓰기 연산이 호출되면 실제 읽기, 쓰기 연산을 할 수 있는 부분이 호출되고, 메타데이터를 변경 혹은 생성, 추출해야 하는 이벤트를 발생시킨다. 실제로 추출 또는 변경해야 할 메타데이터가 어떤 것인지 알기 위해서는 현재 등록된 MGM에 대한 접근을 할 수 있어야 한다. 여기서는 runtime 이라는 객체를 통해서 이런 접근이 가능하게 해준다. 이렇게 리스트 형태로 되어있는 MGM에 대한 접근을 하면 사용자가 스크립트 언어를 통하여 직접 정의한 내용에 의하여 어떠한 메타데이터를 추출해야 하고, 변경해야 하는지 알 수 있다. 이러한 내용을 바탕으로 실제 메타데이터를 추출 혹은 생성, 변경해주는 일을 수행하고, 마지막으로 결과를 메타데이터 저장 공간에 넣어준다.

Algorithm 1. 읽기 연산의 수행 시 MGM 내부 알고리즘
Input: fmax_read 함수의 인자
Output: 새롭게 생성, 혹은 추출된 메타데이터

```
1.  /* Phase 1:  읽기 연산의 호출 */
2.  fmax_read (file structure, buffer, read size, file position)
3.     do_sync_read(file structure, buffer, read size, file position);
4.     /* MGM이 등록된 경우 */
5.        If (MGM 등록 == true)    /* Phase 2 수행 */
6.          mgm_read_event(superblock, dentry_id, buffer, file position, read size);
7.        else
8.          return read_length;
9.
10. /* Phase 2: read event 처리와 메타데이터 테이블 갱신 */
11. /* 등록된 MGM 리스트를 모두 확인 */
12. while (! END_of_MGM_list )
13.    /* MGM 리스트에서 하나 선택하여 mgm_slot 을 구성 */
14.       mgm_slot = select one MGM entry;
15.    /* mgm_slot에 등록된 MGM에 관한 정보를 runtime 객체에 할당 */
16.       runtime = mgm_alloc_runtime(suberblock, mgm_slot);
17.
18.       /* runtime 객체가 가지고 있는 정보를 분석하여 메타데이터 처리 함수 수행 */
19.          run_read_stmt(runtime, start position, buffer, read size);
20.
21.          /* 스크립트 언어를 사용하여 사용자가 등록한 MGM에 관한 내용을 분석 */
22.          /* 분석 내용을 저장하고 있음 */
23.             for( i=0; i<# of TAG; i++  )
24.             /* 메타데이터를 수정, 추출하는 함수 호출하고 MGM table 갱신 */
25.                run_function( i_th TAG);
```

<그림 8> MGM 내부 알고리즘의 pseudo code

4.2.3 멀티디렉토리의 구현 및 설계

3.4에서 언급한 멀티디렉토리의 구현 및 설계는 다음과 같다. FMAX 파일시스템은 멀티디렉토리를 만들기 위한 새로운 시스템 콜(system call)을 정의하지 않고, 디렉토리를 생성하는 쉘 명령인 mkdir을 사용해서 멀티디렉토리를 만든다. FMAX 파일시스템만을 위해서 새로운 시스템 콜을 만든다면 모듈화된 파일시스템의 의미가 약해질 수 있기 때문에 기존에 사용하던 mkdir 명령을 사용한다. 이와 같은 방법은 Semantic File System[1]에서도 사용되었다. 기존의 명령을 사용한다는 것은 커널 내부적으로는 기존에 사용중인 시스템 콜을 그대로 사용하는 것이기 때문에 멀티디렉토리를 위한 루틴을 적절한 위치에 추가해 주어야 한다. 가상 파일시스템 영역을 거친 시스템 콜은 결국에는 FMAX 파일시스템의 inode 연산인 mkdir 함수를 호출할 것이기 때문에, 이 부분에 멀티디렉토리를 위한 루틴을 추가한다면 기존의 mkdir이 하는 기능인 새로운 디렉토리를 생성해 줄 수 있을 뿐만 아니라, 멀티디렉토리도 생성해 줄 수 있게 된다. 또한 멀티디렉토리를 만들기 위한 질의를 입력하는 것도 mkdir 명령어 다음에 입력할 수 있고, 이를 FMAX의 mkdir 함수가 처리해 줄 수 있기 때문에 다양한 질의 처리를 할 수 있다. 이렇게 하기 위해서는 새로운 디렉토리를 생성하는 mkdir 명령과 멀티디렉토리를 생성하는 mkdir 명령을 구분할 수 있어야 한다. FMAX 파일시스템에서는 ‘@md’ 구분자를 사용하여 두 명령을 구분한다. 질의에 대한 문법은 문

맥 자유 문법(Context Free Grammar)으로 BNF(Backus Naur Form)를 사용해서 그림 9와 같이 정의하였다.

Multi Directory syntax

```
<MD syntax>        ::= <MD aware mark> <del_f> <MGM name> <del_r>  <tag>
<tag>              ::= <del_f> < tag_name> <del_in> <data_condition> <del_in> <value>
                        <del_r> | <del_f> <tag_name> <del_r> | <tag>
<tag_name>         ::= string
<data_condition>   ::= desc | asc
<value>            ::= string
<MD aware mark>    ::= @md
<del_f>            ::= {
<del_r>            ::= }
<del_in>           ::= +
```

<그림 9> 멀티디렉토리 생성 질의 문법

그림 9가 정의한 문법을 가지고 실제 멀티디렉토리를 생성하는 명령을 예를 들어 만들어 보면 다음과 같이 된다.

```
$mkdir @md{mgm_fs}{read_count+desc+3}{write_size+asc+2}
```
(1)

(1)을 분석해 보면 '@md' 표식이 있기 때문에 멀티디렉토리를 만들겠다는 것을 뜻한다. 그리고 {mgm_fs}는 멀티디렉토리를 만들기 위해 이용하는 메타데이터가 정의된 MGM의 이름이다. 그리고 '{' 와 '}' 구분자로 나누어진 질의를 만족하는 종속적인 관계를 갖는 멀티디렉토리를 만들겠고, 질의 중 첫 번째는 읽기 횟수가 가장 많은 파일부터 3개만 해당 파일 집합에서부터 가지고 와서 멀티디렉토리를 만들라는 요청이고, 두 번째 질의는 그러한 파일들을 다시 쓰기 크기가 가장 적은 파일 2개를 하위 디렉토리에 나타나게 하라는 요청이다. 이런 식으로 파일들을 어떠한 조건에 맞는 것만 선택적으로 보여줄 수 있게 하기 때문에 물리적인 저장 경로 보다 낮은 깊이만을 탐색해서 파일 접근이 가능하게 된다.

멀티디렉토리를 처리하는 내부 구조는 크게 기존 새로운 디렉토리를 생성하는 명령과 멀티디렉토리를 생성하는 명령을 구분할 수 있는 부분과, 질의를 받아서 분석하는 부분, 실제 멀티디렉토리를 만들고 마지막으로 만들어진 멀티디렉토리에 해당 파일을 내부에서 링크시켜주는 부분으로 나뉘어 진다. 멀티디렉토리의 개념을 활용하면 단순히 파일의 탐색뿐만 아니라 파일을 접근하는 사용자나 프로세스에 따라서 꼭 필요한 파일들과 경로들만 제공해 주는 기능까지 확장하여서 파일시스템 자체에서 1차적인 시스템 보안이 가능하도록 할 수 있다.

5. 실험 및 결과 분석

실험에서는 본 논문에서 제안한 FMAX 파일시스템의 파일 입출력 성능 실험과 함께 [2]에서 제안한 Logic File System과 비교 실험도 하여 서로 다른 환경에서 구현된 파일시스템의 성능을 비교 분석하도록 한다.

5.1 실험 환경

본 논문은 제안한 파일시스템의 성능을 측정하기 위해서 Iozone[15] 이라는 파일시스템 벤치마크(benchmark)를 사용한다. Iozone은 파일시스템의 입출력 연산 성능을 측정하는데 사용되는 대표적인 벤치마크로서, 특히 읽기와 쓰기를 비롯한 다양한 파일 입출력 연산에 대한 성능을 측정한다. 파일시스템의 실험에는 다양한 환경을 설정하고 다양한 실험이 필요하지만, 본 논문에서는 커널 기반으로 파일시스템이 설계되었기 때문에 안정적인 파일 입출력을 측정하고, 그 때의 성능을 측정하는 것을 목표로 한다. 따라서 빈번하게

파일이 생성되고 읽기, 쓰기 연산이 발생할 때의 성능을 측정하게 된다.

파일시스템 내에서 파일 입출력 연산은 많은 함수들을 호출 한다. 일단 VFS 계층의 함수를 호출하며, 현재 마운트된 파일시스템 내부의 함수 호출을 거친다. 물론 대부분 파일 입출력 연산은 파일시스템에 상관없이 공통된 부분이 많지만, 그런 공통된 부분들도 많은 함수 호출이 수행되게 한다. 따라서 커널 기반으로 파일시스템을 설계한 경우 가장 중요한 부분이 안정적인 파일 입출력 연산이다. 특히 입출력 연산에 관계해서 메타데이터를 추출하고, 기존 파일시스템과 다른 입출력 루틴을 갖도록 커널 내부를 수정한 본 논문의 파일시스템의 경우는 안정적인 입출력 연산을 보장하는 것이 일차적인 목표이다. Iozone은 이미 많은 파일시스템의 성능 측정에 사용되고 있으며, 파일 크기를 다양하게 변화시키면서, 입출력 레코드의 크기도 변화 시키면서 단순한 읽기, 쓰기 연산뿐만 아니라, 임의 쓰기, 읽기, 이미 존재하는 파일에 대한 읽기, 쓰기를 비롯하여 fwrite(), fread() 같은 연산까지 테스트하기 때문에 본 파일시스템의 입출력 연산을 광범위하게 측정하는 것엔 가장 적당한 파일시스템이라고 생각했다. 실험에 사용된 하드웨어의 사양은 아래 표 2와 같다.

<표 2> 하드웨어 사양

	Hardware spec
Processor	Intel(R) Pentium(R) 4 CPU 3.00GHz
Memory	1 GB
Disk	60 GB
OS	Linux version 2.6.20 (gcc version 4.1.1 20070105 (Red Hat 4.1.1-51))

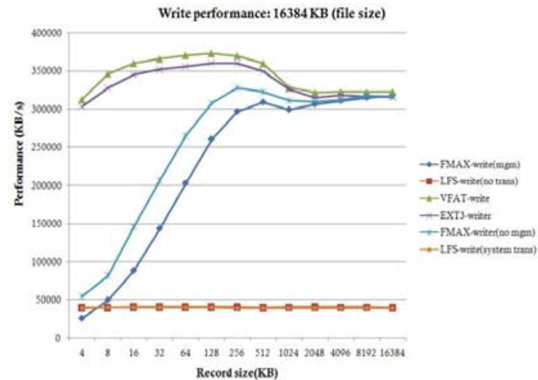
5.2 실험 설계와 결과 분석

실험 설계는 크게 두 부분으로 나뉜다. 첫째는 파일시스템의 입출력 연산을 측정하는 부분이고, 둘째는 멀티디렉토리의 성능을 측정하는 부분이다. 파일시스템의 입출력 연산은 Iozone을 사용하여 측정하고 비교대상으로 기존 커널 기반 파일시스템인 VFAT 파일시스템, EXT3 파일시스템 그리고 사용자 기반 파일시스템인 Logic File System[2]을 사용한다. Logic File System은 [19]에서 다운받을 수 있다. [1], [3]의 파일시스템은 공개되어 있지 않기 때문에 비교 실험은 [2]를 대상으로 하였다. Iozone은 다양한 파일 연산의 실험 측정 결과를 제공하는데, 그 중 기본적으로 read, write 연산에 대한 결과를 비교한다. 멀티디렉토리의 성능 측정 실험은 자체적으로 성능 측정 프로그램을 만들어서 테스트 하였다. 멀티디렉토리의 성능을 측정하는 프로그램은 FMAX 파일시스템을 마운트하고 MGM을 등록한 상태에서 멀티디렉토리를 생성하는 명령을 수행하도록 되어있다. 프로그램 실행 단계에서는 멀티디렉토리에 대한 테스트를 하기 위해 테스트 파일을 만들어주고 다양한 질의에 대해서 만들어진 파일 개수에 따라 멀티디렉토리의 생성을 완료 하는데 걸리는 clock tick을 측정한다. 이런 루틴을 테스트 파일의 개수를 다양하게 변경시키면서 수행한다.

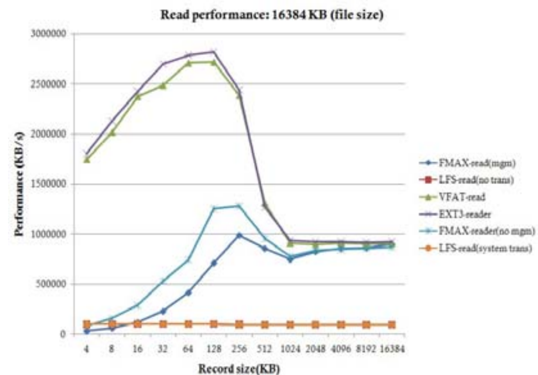
그림 10과 11은 파일의 크기를 16384 KB로 고정하였을 때 레코드의 크기를 변경시킴에 따라 파일시스템의 읽기, 쓰기 성능을 측정한 것으로 결과값은 초당 처리량을 나타낸다. 쓰기 결과를 분석해보면 일단 VFAT파일시스템과 EXT3 파일시스템은 레코드 크기가 변경됨에 따라서 서로 비슷한 결과가 나온다는 것을 알 수 있다. 그리고 FMAX 파일시스템은 레코드 크기가 작을 때 VFAT과 EXT3

에 비하여 좋지 못한 성능을 내는 것을 볼 수 있다. 이유는 FMAX 파일시스템은 일단 MGM을 등록하게 되면 메타데이터를 관리하는 모듈이 동작해서 파일 쓰기가 수행될 때 항상 메타데이터를 새롭게 추출하고 저장공간에 변경된 값을 넣어주기 때문이다. 그리고 이러한 일은 한번의 쓰기 연산에 대해서 수행되므로 레코드 크기가 작을 때는 여러 번의 쓰기연산 수행으로 인하여 기존 파일시스템과 비교했을 때 성능이 낮아지게 된다. 그러나 레코드의 크기가 커지면서 성능의 차이는 작아지게 된다. 그리고 MGM을 등록하지 않은 FMAX 파일시스템도 자체적으로 저장되는 모든 파일에 대한 dentry 구조를 새롭게 정의하여 가지고 있기 때문에 이에 따른 추가적인 부담이 생기면서 기존 파일시스템보다는 낮은 성능을 나타낸다. 두 그래프에서 보면 레코드의 크기가 256KB정도 일 때 FMAX 파일시스템은 최고의 성능을 나타냄을 알 수 있다. 또한 기존 파일시스템도 256KB 정도에서 최고의 성능을 나타낸다. 읽기 결과를 비교하면 전체적으로 쓰기 성능처럼 레코드의 크기가 작을 때는 성능 차이가 크다가 레코드의 크기가 커지면서 비슷한 성능을 나타낸다. 이유는 위에 언급 했던 쓰기 성능 차이의 원인과 같다. 두 실험을 통해서 메타데이터에 대한 루틴은 파일시스템의 성능을 저하시키는 원인이 된다는 것을 알 수 있었다. 읽기 성능의 측정에서도 FMAX 파일시스템은 256KB정도의 레코드 크기에서 최고의 성능을 나타낸다.

Logic File system의 성능을 보면 쓰기와 읽기 연산에서 모두 동일하게 다른 파일시스템들 보다 낮은 성능을 보인다는 것을 알 수 있다. 이는 [2]에서도 밝혀진 결과이다. Logic File System은 FMAX 파일시스템 보다 상대적으로 복잡한 루틴으로 파일 입출력 연산이 수행된다. Logic File System 또한 FMAX 파일시스템의 MGM에 해당하는 트랜스듀서를 등록하거나, 등록하지 않고 사용될 수 있다.



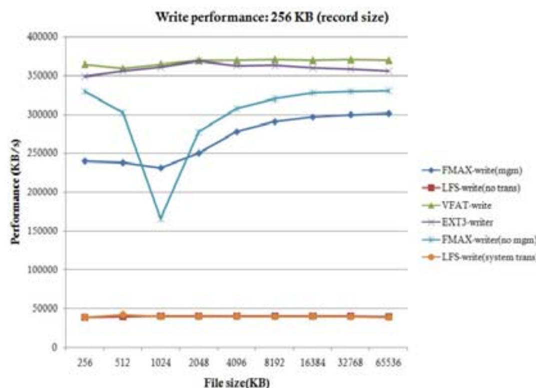
<그림 10> 파일 크기 고정 시 쓰기 성능



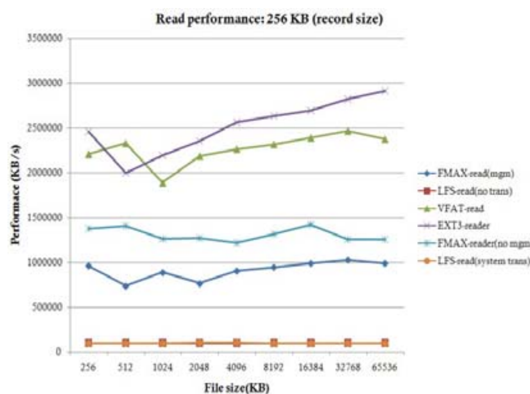
<그림 11> 파일 크기 고정 시 읽기 성능

위 그림 10과 11에서는 트랜스듀서의 등록에 따른 성능 차이를 알아보기 어렵다. 이는 성능의 차이가 작기 때문인데 이 대한 자세한 비교는 그림 14에 나타나 있다. 실험에서 사용된 트랜스듀서는 4가지 속성 혹은 메타데이터를 뽑아낼 수 있는 'system transducer' 이다. 이 트랜스듀서에서 추출하는 메타데이터는 'ext', 'data', 'name', 'size'

이다. FMAX 파일시스템도 일반적인 파일에 대하여 4가지의 메타데이터를 추출하는 MGM인 MGM_FS를 등록하고 실험을 하였다. MGM_FS의 4가지 메타데이터는 read_size, read_count, write_size, write_count 이다. Logic File System의 특징은 레코드의 크기를 변화시킴에 따라서 성능의 차이가 없다는 것이다. 이에 대한 정확한 이유는 Logic File System 에서도 언급되지 않았기 때문에 알 수 없지만 사용자 환경에서 동작하는 파일시스템이기 때문에 고정된 파일크기에 대해서 여러 번의 쓰기 연산이나 한번의 쓰기 연산이나 커널에서의 부하 정도는 같을 것이라고 추측할 수 있다.



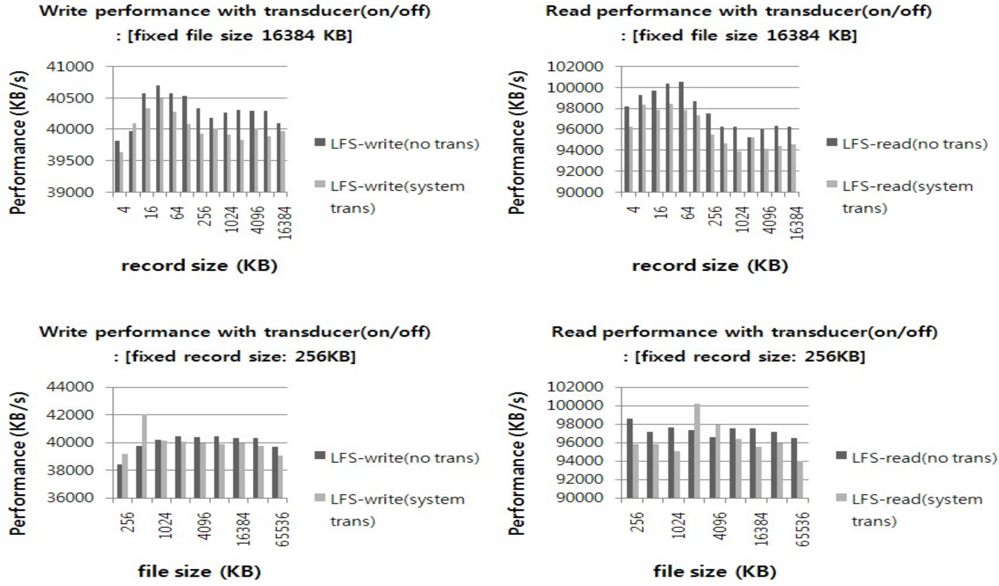
<그림 12> 레코드 크기 고정 시 쓰기 성능



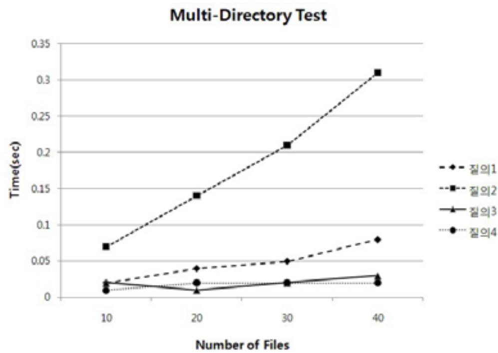
<그림 13> 레코드 크기 고정 시 읽기 성능

그림 12와 13은 레코드의 크기가 256KB로 고정된 상태에서 파일의 크기를 변경시키면서 성능을 측정한 것이다. 그림 12와 13에서 보면 알 수 있듯이 레코드 크기가 고정일 때는 쓰기 성능의 경우 평균적으로 FMAX 파일시스템은 다른 파일시스템보다 평균적으로 1.3배 좋지 않다는 것을 알 수 있고 읽기 성능의 경우 평균적으로 2.3 배 좋지 않다는 것을 알 수 있다. 즉 256KB의 고정 크기일 경우에는 전통 파일시스템과의 성능 차이가 Logic File System[2]에서 제안하였던 사용자 환경의 파일시스템보다 좋다고 할 수 있다. FMAX 파일시스템과 Logic File System는 쓰기 성능의 경우 약 5~6배 차이가 나며, 읽기성능의 경우도 4~5배 정도 차이가 나는 것을 알 수 있다.

그림 10, 11, 12, 13은 Logic File System에 대하여 트랜스듀서를 등록했을 경우와 그렇지 않은 경우에 대한 차이를 그래프 상에서 구분하기 어렵다. 따라서 그림 14에서 성능의 차이를 다시 나타내었다. 전반적으로 트랜스듀서를 등록했을 경우와 그렇지 않은 경우 사이에 성능의 차이가 발생하며, 트랜스듀서를 등록한 경우의 성능이 그렇지 않은 경우보다 좋지 않다. 하지만 트랜스듀서의 등록 여부에 따른 성능의 차이 정도는 FMAX 파일시스템 보다 작다고 할 수 있다. 이유는 Logic File System의 경우 트랜스듀서에서 메타데이터를 추출하고 처리하는 구조가 FMAX 파일시스템의 MGM보다 최적화되고 효과적으로 설계되었기 때문이라고 할 수 있다.



<그림 14> Logic File System에서 트랜스듀서의 등록에 따른 성능 비교



<그림 15> 멀티디렉토리 생성에 대한 부하 측정

멀티디렉토리의 성능은 다음 그림 15와 같다. 실험에 사용된 멀티디렉토리 생성 명령은 4.2.3에서 언급한 명령(1)에 3개의 다른 멀티디렉토리 생성 명령을 추가하였다. 네 가지의 서로 다른 명령은 멀티디렉토리를 만들기 위해 사용되는 TAG의 수, 질의에 주는 조건의 유무를 가지고 구분 된다. 물론 이 밖에 다양한 질의가 있을 수 있지만 대표적인 질의 구조의 예를 가진 다음 질의 4개를 실험에 사용하였다.

■ 질의1:
@md{mgm_fs}{read_count+desc+3}{write_size+asc+2}

■ 질의2:
@md{mgm_fs}{write_size}{read_count}

■ 질의3:
@md{mgm_fs}{read_count+asc+4}

■ 질의4:
@md{mgm_fs}{read_count}

그림 15는 실험 결과를 나타낸다. 결과를 분석하면 질의 2가 가장 많은 부하를 요구한다. 두 번째로 많은 부하를 요구하는 질의는 1번 이다. 마지막 질의 3과 4는 비슷한 부하를 요구한다. 질의의 구조를 분석해 보면 하나의 TAG보다 두 개의 TAG를 가지고 멀티디렉토리 명령을 주었을 때가 전체적으로 많은 부하를 요구한다. 이는 멀티디렉토리를 처리하는 내부 구조에서 많은 수의 TAG가 명

령으로 들어온 만큼 계층적으로 만들어야 할 하위 디렉토리가 많을 뿐만 아니라, 그 만큼 파일들을 찾아서 멀티디렉토리에 링크를 시켜주어야 하기 때문이다. 2개의 TAG를 사용한 명령의 경우 질의에 조건을 주지 않은 질의 2번이 더 많은 부하를 요구하는데 이 또한 내부에서 만들어야 할 하위 디렉토리와 링크해야 할 파일의 수가 조건을 주지 않은 명령보다는 많기 때문에, 이에 따라 더 많은 부하를 발생시킨다. 그러나 이 부하는 멀티디렉토리를 생성하는 경우에만 파일시스템에 생기는 것으로 멀티디렉토리를 생성한 후 그 디렉토리나 내부 파일들에 대한 입출력 연산에는 영향을 미치지 않는다. 하지만 멀티디렉토리를 구성하는 알고리즘 상에서 이러한 부하를 줄이기 위하여 좀 더 효율적으로 디렉토리 구조를 만드는 방법을 생각해 볼 수 있다. 한 예로 한 종류의 메타데이터에서 일정 영역 안의 값을 갖는 파일들은 하나의 동일한 멀티디렉토리 안에 존재하도록, 멀티디렉토리의 수를 줄여서 나타낼 수 있다. 즉 비슷한 멀티디렉토리들은 하나의 멀티디렉토리로 클러스터링 하는 방법이다. 아직 이러한 개선 방법은 실제 구현되지는 않았지만 향후 멀티디렉토리 생성의 부하를 어느 정도 해결할 수 있는 방법이다.

다음으로 질의 3과 4는 부하의 정도 차이가 거의 없는 것을 볼 수 있다. 질의 3과 4는 모두 하나의 TAG만을 멀티디렉토리의 명령으로 주는 경우 인데, 이의 경우 2개의 TAG를 사용하는 것 보다 내부에서 처리해야 할 작업이 줄어들기 때문에 당연히 더 적은 부하를 나타낸다. 실험 환경에서 질의 1과 2는 write_size와 read_count를 각 파일 마다 그 값을 변경해 주어서 많은 파일들이 조건에 따라서 분류가 되었지만, 질의 3과 4를 실험할 때는 동일한 read_count를 주었다. 이렇게 실험을 설계한 이유는 질의에 주는 조건이 실제로 파일들을 분류하는 정도에 따라 부하의 차이를 알고 싶었기 때문이다. 결과적으로 질의 3과 4는 파일 수의 변화에 따른 부하 정도가 커지지 않은 반면, 질의 1과 2는 부하 정도가 증가하였다. 이는 내부적으로 질의 조건에 만족하는 파일들을 정렬하고 멀티디렉토리와 그 하위 멀티디렉토리에 파일을 분류화하는 작업량이 증가했기 때문이라고 할 수 있다.

6. 결론

본 논문에서는 리눅스 환경에서 메타데이터를 관리할 수 있는 새로운 파일시스템을 제안하였다. 기존 파일시스템처럼 FMAX 파일시스템은 커널 환경에서 모듈화되었기 때문에 적재명령을 통하여 쉽게 사용자나 응용프로그램이 새로운 파일시스템을 사용할 수 있다. 또한 기존 FAT계열 파일시스템을 바탕으로 설계되었기 때문에 커널 환경에서 안정적으로 작업을 수행할 수 있는 구조를 가지고 있다.

파일시스템 내부에서 MGM으로부터 추출된 메타데이터들은 관계형 구조로 파일시스템이 사용하지 않는 저장 장치 상의 예약된 공간에 저장 된다. MGM이 어떠한 메타데이터를 추출할 것인지는 사용자 수준의 스크립트 언어를 제공해 주어서 사용자나 응용프로그램이 MGM을 정의할 수 있게 하였다. 이는 파일시스템 내부에 스크립트 언어를 분석할 수 있는 분석 엔진을 만들어서 사용자가 원하는 작업을 할 수 있도록 파일시스템을 조작할 수 있다는 것을 의미한다. 또한 추출된 메타데이터를 사용하여 질의 기반으로 파일

의 탐색을 좀 더 효과적으로 할 수 있는 멀티디렉토리라는 기능을 제공한다.

기존 논문들은 사용자 환경에서 위에 언급한 기능들을 부분적으로 가지는 파일시스템을 제안하였지만, 통합적으로 모든 기능을 제공하지는 못하였다. 또한 커널 환경에서의 설계와 구현이 갖는 많은 시간적 환경적 제약조건 때문에 모두 커널 환경이 아닌 FUSE 등을 사용하여 사용자 환경에서 파일시스템을 설계하고 구현하였다. 본 논문이 제안하는 FMAX 파일시스템은 커널 환경의 많은 제약조건들을 분석하여 설계되었고 안정적으로 동작하는 것을 실험을 통하여 확인하였다. 또한 파일 입출력 연산의 성능은 전통적인 파일시스템보다는 평균적으로 2.5배 정도 좋지 않은 결과를 보여준다. 이는 직관적으로 기존 파일시스템들이 제공하지 못했던 메타데이터 관리 기능을 제공하기 때문에 수반되는 한계이다. 하지만 기존 논문들에서 제안한 사용자 환경에서 구현된 메타데이터 관련 파일시스템들은, 대표적으로 Logic File System 같은 경우에 전통적인 파일시스템보다 좋지 않은 성능을 보여주면서 또한 본 논문이 제안하는 파일시스템 보다 전반적으로 4~6배 정도 좋지 않은 성능을 나타냈다. 이는 사용자 환경에서 설계된 파일시스템은 다양한 기능을 보다 쉽게 제공할 수는 있지만 성능상의 한계로 인하여 실제 실용화되어 사용될 수 있는 파일시스템으로서는 한계가 있다는 것을 의미한다. 그렇지만 트랜스듀서의 등록 여부에 따른 성능의 차이는 본 논문의 파일시스템보다 크지 않은 것으로부터 향후 연구에서 본 논문의 파일시스템도 MGM의 구조를 최적화할 필요가 있다고 생각된다.

본 논문이 제안한 FMAX 파일시스템에서는 커널 환경에서 메타데이터를 관리하는 파일시스템의 프로토타입을 개발한다는 목적으로 개별 알고리즘의 효율성보다는 전체 시스템의 구현에 초점을 맞추었다. 따라서 몇 가지 부분들에 대하여 알고리즘의 효율성을 개선하거나 발생할 수 있는 문제점의 다양한 해결 방법에 대한 연구의 여지를 남겨 놓았다. 이러한 부분들은 향후 연구에서 다룰 예정이다.

참고 문헌

[1] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, Jr. Semantic file system. In Proceedings of the 13th ACM Symposium on Operating Systems Principles(SOSP '91), pages 16-25. ACM, Oct. 1991.

[2] Y. Padioleau and O. Ridoux. A logic file system. In Proceedings of the 2003 USENIX Annual Technical Conference, pages 99-112, Stanford, Nov, 1998.

[3] A Ames, C Maltzahn, N Bobb, EL Miller, SA Brandt, A Neeman, A Hiatt, D Tuteja. Richer file system metadata using links and attributes. In the Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005), pages 49-60, Monterey, CA. April. 2005.

[4] Prashanth Mohan, Raghuraman, Venkateswaran S, Arul Siromoney. Semantic file retrieval in file systems using virtual directory. The 13th IEEE International Conference on High Performance Computing(HiPC 2006), Bangalore, India, December 18-21, 2006.

[5] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck.

NFS version 4 protocol. RFC 3010, Network Working Group, December 2000.

[6] Simon Schenk, Olaf G¨orlitz, Steffen Staab, “TagFS: Bringing Semantic Metadata to the Filesystem”, Demos and Posters of the 3rd European Semantic Web Conference(ESWC 2006), Budva, Montenegro, 11th 14th June 2006.

[7] Ganesan Shankaranarayanan, Adir Even, “The metadata enigma”, Communications of the ACM, Volume 49 Issue 2, pp.88~94 February 2006.

[8] Divesh Srivastava, Yannis Velegrakis, “Storage engine and access methods: Intensional associations between data and metadata”, Proceedings of the 2007 ACM SIGMOD international conference on Management of data SIGMOD '07, pp.131~136, June 2007.

[9] Bernhard Schandl, Ross King, “The SemDAV Project: Metadata Management for Unstructured Content”, Proceedings of the 1st International Workshop on Contextualized Attention Metadata: Collecting, Managing and Exploiting of Rich Usage Information CAMA '06, pp.27~31, November 2006.

[10] Gregory R. Ganger, Marshall Kirk McKusick, Craig A. N. Soules, Yale N. Patt, “Soft updates: a solution to the metadata update problem in file systems”, ACM Transactions on Computer Systems (TOCS), Volume 18, Issue 2, pp.127~153, May 2000.

[11] File System Performance: The Solaris™ OS, UFS, Linux ext3, and ReiserFS. A Technical White Paper August 2004. Sun Microsystems. http://www.sun.com/software/whitepapers/solaris10/fs_performance.pdf

[12] EXT3, Journaling Filesystem, Dr. Stephen Tweedie(2000), http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html

[13] Andrew Pitt. The FAT32 Resource page. http://www.project9.com/FAT32

[14] Werner Vogels. File System Usage in Windows NT 4.0. in Proceedings of 17th ACM Symposium on Operating Systems Principles. pp.93~109, 1999.

[15] Don Capps and William Norcott. Iozone File System Benchmark, 1998. http://www.iozone.org

[16] Microsoft Corporation. Microsoft MSDN WinFS Documentation. Http://longhorn.msdn.microsoft.com/lhsdk/winfs/daovrwelcometowinfs.aspx, 2003.

[17] M. Szeredi. File System in User Space README. http://www.stillhq.com/extracted/fuse/README, 2003.

[18] Calvelli, Claudio, “The Perl Filesystem”, http://dd-sh.assurdo.com/perlfs, April 2001.

[19] http://aryx.kicks-ass.org/~pad/wiki/wiki-LFS/doku.php

박 치 현
2007.2 : 홍익대학교 컴퓨터공학과 (공학사)
2007.3~ : 연세대학교 컴퓨터과학과 석사과정
관심 분야 : 파일시스템, 바이오인포매틱스, 데이터 마이닝
Email : tianell@cs.yonsei.ac.kr

김 우 철
2003.2: 연세대학교 컴퓨터과학과 (공학사)
2006.2: 연세대학교 컴퓨터과학과 (공학석사)
2006.3~: 연세대학교 컴퓨터과학과 (박사과정)

관심 분야 : 바이오인포매틱스, 데이터베이스시스템, 데이터 마이닝
Email : twelvepp@cs.yonsei.ac.kr

노 홍 찬
2006.2: 연세대학교 컴퓨터과학과 (공학사)
2008.2: 연세대학교 컴퓨터과학과 (공학석사)
2008.3~: 연세대학교 컴퓨터과학과 (박사과정)
관심 분야: 플래쉬메모리 인덱스, SSD, 데이터 마이닝
Email : fallsmal@cs.yonsei.ac.kr

박 상 현
1989.2 : 서울대학교 컴퓨터공학과 (공학사)
1991.2 : 서울대학교 컴퓨터공학과 (공학석사)
2001.2 : UCLA 대학교 전산학과 (공학박사)
2001.2 ~ 2002.6 : IBM T. J Watson Research Center Post-Doctoral Fellow.
2002.8 ~ 2003.8 : 포항공과대학교 컴퓨터공학과 조교수
2003.9 ~ 2006.8 : 연세대학교 컴퓨터과학과 조교수
2006.9 ~ 현재 : 연세대학교 컴퓨터과학과 부교수
관심분야 : 데이터베이스 보안, 데이터 마이닝, 바이오인포매틱스, XML
E-mail : sanghyun@cs.yonsei.ac.kr