EXT3 파일 시스템의 효율적인 공간 활용을 위한 선택적 압축 알고리즘 설계

이성헌* · 장승주*

*동의대학교

Optional Compression Algorithm Design for Efficient Space Utilization of the EXT3 File System

Seong-Heon Lee* · Seung-Ju Jang*

*Dong-Eui University

E-mail : kkulee@deu.ac.kr, sjjang@deu.ac.kr

요 약

본 논문에서는 EXT3 파일 시스템의 Ordered mode 선택적 압축 알고리즘 기법 적용을 제안한다. 시스템이 비정상적으로 종료되거나 오류가 발생할 경우 수정 중이던 데이터가 손실되거나 기존 데이터의 복구가 불가능하게 될 수 있다. 이러한 문제점을 극복하기 위하여 저널링 파일 시스템이 사용된다. 저널링 파일 시스템은 저널 이라는 추가적인 공간을 사용하여 관리한다. EXT3 파일 시스템은 가장 널리 사용하는 저널링 파일 시스템이다. 본 논문에서는 기존 EXT3 파일 시스템의 파일 쓰기 수행 시 저장장치의 효율적인 공간 활용을 위하여 선택적 압축 알고리즘 기법을 적용을 제안한다.

ABSTRACT

In this paper, ordered mode of EXT3 file system offers to use an optional compression algorithm technique. If the system terminates abnormally or an error occurs, data which is being modified will be possibly damaged or a recovery of the existing data can be impossible. To overcome these problems, a journaling file system is used. Journaling file system manages by using an additional space called Journal. EXT3 file system is the most widely used journaling file system. In this paper, When performing a file writing of an existing EXT3 file system, it offers to use an optional compact algorithm technique for an efficient use of a space of storage device.

키워드

EXT3, Journaling File System, Selective Compression Algorithm

Ⅰ. 서 론

컴퓨터의 사용범위가 넓어져가고 임베디드 시스템이 발전하면서 리눅스 파일 시스템의 안정성이 중요시되고 있다. 파일 시스템이란, 파일의 내용과 메타데이터들을 유지하고 관리하는 체계이다. 파일 시스템에는 파일의 데이터와 메타데이터를 동시에 저장하지않는 비동기식 파일시스템과 메타데이터를 동시에 저장하는 동기식 파일시스템이 있다. EXT3 파일 시스템은 비동기식 파일 시스템으로 메타데이터를 파일의 내용을 저장할 때 저장하는 것이 아니라, 메모리에 두었다가 일정한 시간 간격을 두고 저장한다. 일정한 시간 간격을 두고 저장하기 때문에, 저장하기 전 시스템이 다운될 경우 많은 문제가 발생하게 된다.

기존 리눅스 파일시스템의 많은 문제점을 해결하기위해 다양한 저널링 파일 시스템이 개발되었으며, EXT3 파일 시스템은 가장 널리 사용하는 저널링 파일 시스템의 한 종류이다.

하지만 저널링 파일시스템의 단점으로 저널이라는 추가적인 디스크 공간이 필요하고, 저널링

기법을 수행을 위한 추가적인 작업이 수행된다.

본 논문에서는 EXT3 파일 시스템의 Ordered mode에서의 저장장치 공간을 효율적으로 절약하기 위한 선택적 압축 알고리즘 기법을 제안한다.

본 논문에서는 기존의 EXT3 파일 시스템의 Ordered mode 저널링 기법을 수행하기 때문에 시스템 안정성은 그대로 유지할 수 있다.

본 논문은 2장에서 EXT3 파일 시스템에 대하여 설명하고, 3장에서 선택적 압축 알고리즘 구현, 4장 실험 및 평가, 그리고 5장에서는 본 연구의 결론으로 구성한다.

Ⅱ. EXT3 저널링 파일 시스템

리눅스 운영체제 시스템에서 저널링 파일 시스템은 사용자가 데이터를 입력 또는 수정을 하면 메타 데이터를 로그에 기록한다. 여기서 메타 데이터는 파일의 위치, 크기, 소유자, 접근 권한 등의 파일과 관련된 데이터를 말한다. 만약 기록 중 오류가 발생하거나 비정상적인 종료로 인한 재부팅을 할 경우 로그에 기록되어있는 메타 데이터를 참고하여 파일시스템을 복구할 수 있다. 저널링 파일 시스템은 동기식 파일 시스템으로 수정되거나 새로 기록될 메타 데이터의 로그를 버퍼에 남겨두었다가 주기적으로 파일 시스템에 커밋한다. 오류가 발생하면 저널을 이용하여 데이터를 복구하고, 파일 시스템의 메타 데이터의 손상을 막는다.[1][2]

리눅스의 표준 파일 시스템으로 EXT 파일 시스템, EXT2 파일 시스템을 거쳐 저널링 기능이 포함된 EXT3 파일 시스템으로 발전해왔다. EXT3 파일 시스템은 가장 인기있는 저널링 파일 시스템이며, 기존의 EXT2 파일 시스템 기반으로 디스크 포맷이 동일하고 데이터의 손실 없이 간단하게 파일 시스템을 변경할 수 있다. EXT3 파일 시스템에서 저널링은 Writeback mode, Ordered mode, Journal mode 세 가지의 모드를 가지고 있으며 Default로 Ordered mode를 사용한다.[3]

Ordered mode는 메타데이터만 저널에 기록되며, 데이터는 저널에 기록되지 않지만 관련된 메타 데이터가 저널에 기록되면 데이터는 저장장치에 반드시 기록된다. 메타 데이터의 로그를 데이터 블록보다 먼저 디스크의 저널영역에 기록하고 로그의 기록이 끝나면 데이터 블록이 디스크에 기록된다.

파일을 쓰거나 수정 중 오류가 발생하면, 저널은 삭제 처리가 되며 이러한 방식으로 메타 데이터만 저널에 기록되더라도 EXT3 파일 시스템은 fsck를 사용하여 데이터와 메타데이터의 일관성을 유지할 수 있다.

fsck란, 예상치 못한 시스템 종료로 인해 일어나는 파일 시스템의 일관성을 체크하고, 복구하는 프로그램이다. 링크 수 나 데이터 블록의 값들을 사용하여 디스크의 슈퍼블록, 아이노드 등의 이상 유무를 점검하게된다. fsck는 파일 시스템의 크기가 커짐에 따라 수행시간이 길어지며, 메타데이터의 손실이 저널링 기능으로 복구 할 수 없는 경우 유용하게 사용할 수 있다.

EXT3 파일 시스템에서 저널링은 Journaling Block Device Layer(JBD)라고 하는 API를 사용하여 관리되고, JBD는 모든 종류의 블록 디바이스상의 저널을 구현하도록 설계되어있다.[4]

본 논문에서는 EXT3 저널링 파일 시스템의 Default mode인 Ordered mode를 기반으로 사용한다.

Ⅲ. 선택적 압축 알고리즘 구현

본 논문에서는 EXT3 파일 시스템에서 저장장치의 효율적인 공간 활용을 위하여 파일 쓰기 시 선택적 압축 알고리즘을 적용한다.

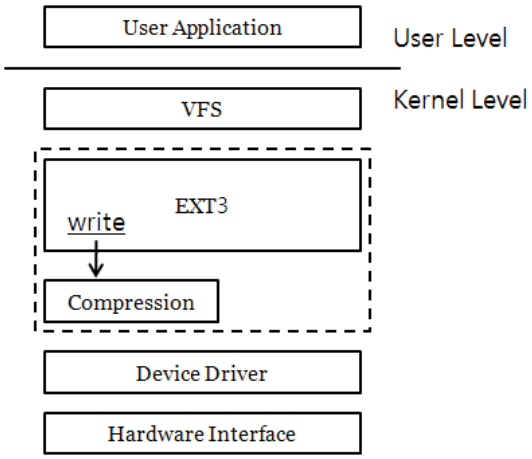

그림 1. 전체 시스템 구조

그림 1.은 본 논문에서 제시하는 전체 시스템 구조이다. 저널링 파일 시스템은 한 번의 쓰기를 수행하기 위해 실제로 두 번의 쓰기를 수행하며, 별도의 저널공간을 사용하여 추가적인 디스크 공간이 필요하다.[5]

개선된 EXT3 파일 시스템에서는 write() 시스템 콜이 호출되면 선택적 압축 알고리즘을 적용된다. 사용자 수준의 압축 파일과 같이 압축률이 낮은 파일의 경우 원본 크기와 비슷한 크기의 압축 파일이 생성되어 압축 효율이 떨어질 뿐만 아니라 파일 중복 압축에 따른 불필요한 프로세서 낭비와 쓰기 수행시간까지 늘어나게 된다. 여기서 사용자 수준의 압축 파일이란, jpeg, mpeg, mp3 등 멀티미디어 파일과 같이 압축 기법이 적용된 파일을 말한다. 따라서 압축률이 낮은 파일을 식별하기위하여 선택적 압축 기법을 적용한다. 쓰기 작업의 수행 전 파일의 앞부분을 일정크기 만큼 압축을 수행한 뒤 압축된 파일의 크기를 구한다.

압축된 파일의 크기와 미리 지정되어있는 임계값의 크기를 비교한 후 선택적 압축 알고리즘의 적용 유무를 판단하여 낮은 압축률을 보이는 파일을 필터링 시킬 수 있다.

압축 알고리즘은 gzip 파일 포맷의 zlib를 사용하여 구현하였다. zlib 압축 라이브러리는 C언어로 구현되어 있으며 LZW방식과 함께 널리 쓰이는 방식 중 하나로 거의 아무런 제한 없이 자유롭게 사용할 수 있는 라이선스이며 DEFLATE 압축 알고리즘을 이용한 다양한 압축 및 압축해제 함수들을 제공하여 간단하게 압축 프로그램을 만들 수 있다. DEFLATE 알고리즘은 내부적으로 LZ77 알고리즘을 통해 데이터를 압축한 뒤, 중복되는 내용에 대한 포인터를 허프만 부호화를 사용하여 한 번 더 압축한다.

압축 알고리즘은 zlib라이브러리의 gzwrite()함수를 사용하여 구현하였으며, gzopen(), gzclose(), gzerror() 함수 등을 사용하여 파일처리와 에러처리를 하였다.[6]

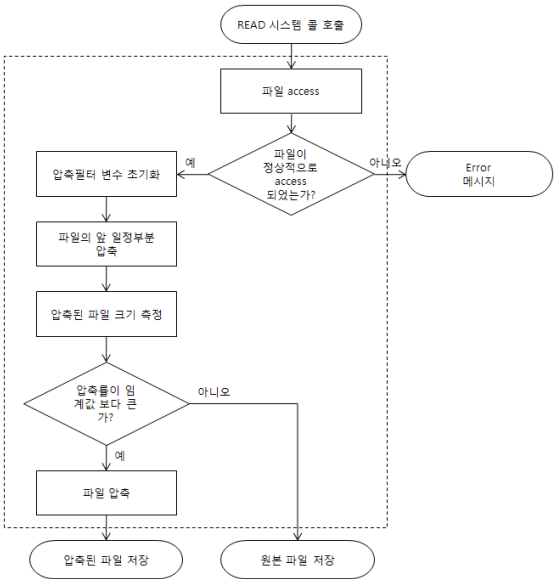

그림 2. 선택적 압축 알고리즘

그림 2.는 선택적 압축 알고리즘의 동작 흐름을 보여준다. user 프로그램에서 read() 시스템 콜이 호출 되면 파일을 access하게 되고 구현된 프로그램에 의하여 파일의 앞부분의 일정부분이 압축이 수행된다. 압축된 파일의 크기와 미리 지정해놓은 임계값과의 크기를 비교 한 후 압축 후 저장 또는 원본파일 저장을 판단하고 실질적인 쓰기작업을 수행한다.

본 논문에서는 페도라 core9 커널 2.6.33.6 버전을 사용 하였다. 커널의 /fs/ext3/inode.c파일의 ext3_journal_start()부분에 printk()함수를 적용하여 커널을 리빌딩 한 뒤 val/log의 메시지를 확인해 보면 ext3_write_begin()함수에서 저널링 기능이 시작됨을 할 수 있다.

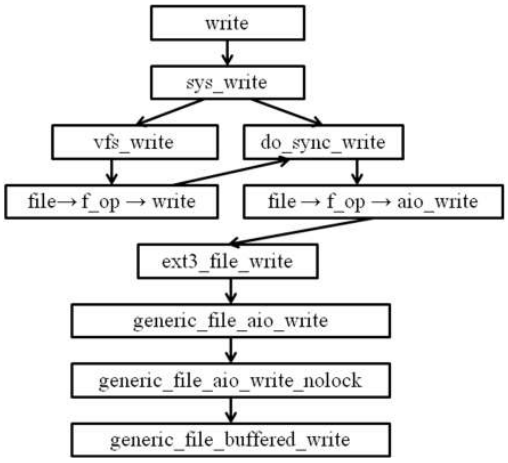

그림 3. write()시스템 콜 흐름도

그림3. 은 EXT3 파일 시스템에서 write() 시스템 콜을 호출 하였을 때 커널에서의 처리를 보여준다. 사용자 프로그램이 write() 시스템 콜을 호출하면 sys_write() 함수와 ext3_file_write()함수를 거쳐 generic_file_beffered_write() 함수에 의해 쓰기 연산의 수행이 시작된다. Zlib 라이브러리를 이용하여 generic_file_beffered_write() 함수에 선택적 압축 알고리즘을 적용한다.

Ⅳ. 실험 및 평가

본 장에서는 구현된 설계 내용을 바탕으로 실험한 내용의 성능 평가를 분석한다.

구현은 리눅스 커널 2.6.33.6버전을 사용하는 Fedora 9 core 운영체제를 사용하였고, Core 2 duo 2.13GHz CPU와 2GB의 메모리를 탑재하고 있다. 실험은 EXT3 파일 시스템의 기본적으로 사용하는 Oredered mode를 사용하였으며, zlib 1.2.5 버전을 사용하여 프로그래밍 하였다.

실험 구성은 다음과 같다. 기존의 EXT3 파일 시스템과 선택적 압축 알고리즘이 적용된 EXT3 파일 시스템의 성능 평가를 위해 한글, 엑셀, 파워포인트, 워드 포맷의 파일을 각각 20개를 무작위로 임의로 선정하여 실험을 하였다. 선택적 압축 알고리즘이 적용된 EXT3 파일 시스템의 성능 향상을 검증하기위해 대상 실험 파일의 쓰기 시간과 파일의 크기를 비교 분석 하였다.

	한글	엑셀	파워포인트	워드	전체
실험파일 수	20	20	20	20	80
선택적 압축알고리즘에 의해 압축된 파일 수	16	19	3	19	57
기존 파일크기 (byte)	4152832	5604352	32083963	15469420	57310572
선택적 압축 알고리즘 적용 후 파일 크기 (byte)	3795655	2775084	30801845	12915301	50287885

표 1. 각 파일별 실험 결과

표. 1은 선택적 압축 알고리즘이 적용된 EXT3 파일 시스템에서 실험한 결과이다. 각 파일의 앞부분 10240byte 크기만큼 압축을 실행한 뒤 미리 지정한 임계값과 비교하여 압축 저장 기법을 적용할지 판단하게 된다. 본 논문의 실험에서는 임계값의 크기를 5120byte로 설정하고 임의의 압축파일이 50% 이상의 압축률을 보이면 압축 저장 기법을 적용한 뒤 쓰기 수행을 하도록 구현되어 있다. 전체 80개의 파일 중 57개의 파일이 압축저장 기법이 적용되어 쓰기 되었다는 것을 알 수 있다. 기존의 EXT3 파일 시스템에서 80개의 실험 대상 파일의 총 크기가 57310572byte인 반면 선택적 압축 저장기법이 적용된 EXT3 파일 시스템에서는 50287885byte로 약 10%이상의 공간적 효율이 발생하였다는 것을 확인할 수 있다.

압축 수행 시간은 전체적으로 낮게 측정 되어 선택적 압축 알고리즘의 공간적 효율에 비해 무시해도 될 정도로 보인다.

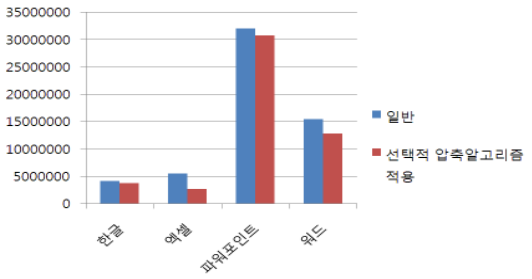

그림 4. 각 파일 포맷별 공간적 효율

그림 4.는 각 파일포맷별 공간적 효율을 나타낸다. 엑셀 파일 포맷의 경우 선택적 압축알고리즘의 적용으로 월등한 성능을 나타 내었고, 한글 파일 포맷의 경우 선택적 압축 알고리즘에 의해 압축 저장기법이 많이 적용되었음에도 불구하고 공간적 효율은 낮게 측정된 것을 볼 수 있다.

각 파일 포맷의 헤더 부분 압축 효율이 다르기 때문에 임의로 압축 시킬 파일의 크기와 임계값의 크기 조절로 더 효율적인 결과를 기대할 수 있다.

V. 결 론

본 논문에서는 EXT3 파일 시스템에 선택적 압축 알고리즘 적용을 제안하였다. 파일의 쓰기 수행 시 선택적 압축 알고리즘을 통해 압축률이 높은 파일의 경우 저장장치의 효율적인 공간 활용을 가능하게 해준다. 그리고 기존의 EXT3 파일 시스템의 Ordered mode 저널링을 수행하기 때문에 안정성을 그대로 유지할 수 있다.

선택적 압축 알고리즘은 C언어 기반의 zlib압축 라이브러리를 이용하여 구현하였다. zlib 압축 라이브러리를 사용하여 아무런 제한 없이 자유롭게 사용할 수 있다.

본 논문에서 제안한 선택적 압축 알고리즘의 적용으로 압축 수행시간이 발생하여 성능상의 단점이 발생하지만 저장장치를 효율적으로 사용할 수 있는 공간적인 효율을 확인하였다. 현재 사용자 수준의 압축파일과 같은 압축률이 낮은 파일도 선택적 압축 알고리즘에 의해 압축 작업이 수행되도록 설계되어있다. 향후 jpeg, mpeg등과 같이 사용자 수준에서 압축이 되어있는 파일을 보다 효율적으로 필터링 할 수 있게 설계 내용을 추가 할 계획이다.

참고문헌

[1] The Linux Cross Reference http://lxr.linux.no/

[2] Korean Linux Documentation Project http://kldp.org/

[3] Linux kernel source code http://www.kernel.org/

[4] RedHat, “Red Hat’s New Journaling File System : ext3”, 2001

[5] TAKAHASI HIROKAZU, “리눅스 커널 2.6 구조와 원리”, 한빛미디어, 2007

[6] http://www.zlib.net/manual.htm