범용 데이터 압축 효율 개선을 위한 SDD 알고리즘에 관한 연구

장승주*
*동의대학교 컴퓨터공학과

A Study of the SDD(Selective Data Distinction) Algorithm for Efficiency Improvement of the ASCII Data Compression

Seung Ju Jang*
*Dongeui Univ. Dept. of Computer Engineering

요 약

본 논문은 데이터 압축 효율 향상을 위하여 데이터에 대해서 무조건적인 압축을 시행하는 것이 아니라 SDD 알고리즘을 제시하고 범용데이타에 대한 압축을 수행한다. SDD 알고리즘은 범용 데이터 특성을 우선적으로 분석하여 압축 여부를 판단하게 된다. 이렇게 함으로써 압축 효율이 좋지 않은 경우에 대한 회피를 통해서 불필요한 압축을 하지 않을 수 있도록 한다. 불필요한 연산을 줄임으로써 압축 알고리즘의 성능 향상을 꾀할 수 있다. 특히, 이미 압축 알고리즘이 적용이 된 데이타의 경우에 압축 알고리즘을 재차 적용하더라도 효율적인 압축이 되지 않는 경우가 많다. 이러한 경우에도 불필요한 압축을 하지 않도록 한다. 본 논문에서 제안하는 기능에 대해 실제 구현하고, 구현된 내용에 대해서 실험을 수행하였다. 본 논문에서 제시한 내용에 대해서 실험한 결과 정상적인 동작이 됨을 확인할 수 있었다.

1. 서론

현대 사회에서 컴퓨터는 거의 모든 분야에서 사용하고 있으며, 사용 범위는 점차 넓어져가고 있다. 특히, 컴퓨터 시스템에 장애가 발생하거나 중요한 정보를 보호하기 위하여 안정성과 보안이 가장 중요시 되고 있다. 시스템 안정성에 문제가 생길 경우 파일 시스템 전체의 동작이 중단 되고 작업 중이었던 데이터가 손실 되거나 기존 파일 시스템의 데이터까지 복구가 불가능하게 되는 상황이 발생 할 수 있다.

파일 시스템이란, 파일의 내용과 그 파일과 관련된 데이터 즉, 메타데이터들을 유지하고 관리하는 체계이다. 리눅스는 표준 파일 시스템으로 EXT 파일 시스템, EXT2 파일 시스템으로 변천해왔다. EXT3 파일 시스템은 매우 안정적이고 치명적인 문제점도 없으며, EXT2 파일 시스템에서 EXT3 파일 시스템으로의 변경도 간단하여 일반적으로 많이 사용 되고 있는 저널링 파일 시스템이다.

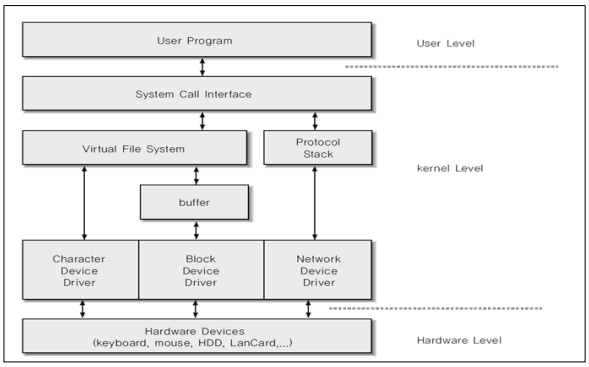
본 논문에서는 리눅스에서 사용 가능한 범용 파일 시스템을 사용한다. 본 논문에서는 SDD 알고리즘의 제시와 이 알고리즘의 사용으로 저장장치의 효율적인 활용을 위한 기능을 설계 및 구현한다.

본 논문에서는 SDD 알고리즘의 제시를 통해서 압축 저장 기법을 적용하여 효율적인 저장장치 공간 관리와 쓰기 속도를 향상시키는 파일 시스템의 개선을 제안한다.

2. 관련연구

리눅스 커널에서 디바이스 드라이버는 시스템

콜 인터페이스와 하드웨어 제어 사이에 위치하며 응용 프로그램이 디바이스 드라이버를 통해 각종 주변 하드웨어 장치에 접근할 수 있는 기능을 제공한다. 디바이스 드라이버에는 케릭터 디바이스 드라이버와 블록 디바이스 드라이버가 있으며 블록 디바이스 드라이버는 하드디스크처럼 내부에 파일 시스템을 가질 수 있는 디바이스 드라이버를 말하며, 임의 접근이 가능하고 블록 단위의 데이터 전송이 가능하다 [1,2,3].



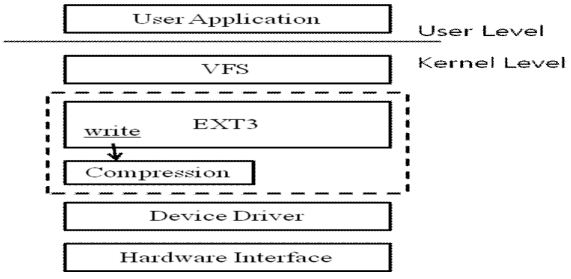
(그림 1) 리눅스 커널의 디바이스 드라이버

커널 모듈 프로그램이란 필요에 따라 커널에 load 하거나 unload 할 수 있는 특정한 기능을 수행하는 프로그램이다.

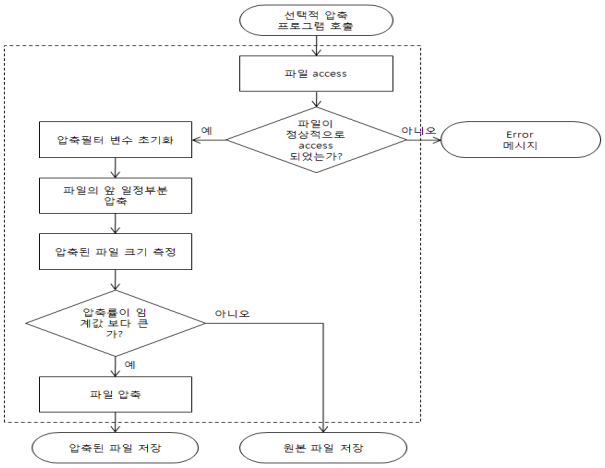
저널링 파일 시스템에서 저널은 파일 시스템에 반영될 수정 사항을 버퍼에 로그로 남겨두었다가 주기적으로 파일 시스템에 커밋된다. 비정상적 종료가 발생하면, 저장되지 않은 데이터를 복구하기 위한 검사 지점으로 사용되고 파일시스템의 메타 데이터의 손상을 막아준다.

3. SDD(Selective Data Distinction) 알고리즘 설계

EXT3 파일 시스템에 저장장치의 효율적인 공간 활용과 쓰기 속도를 향상시키기 위해 파일에 압축 저장 기법을 적용한다. 파일의 압축 시 사용자 수준의 압축 파일과 비압축 파일을 식별한 후 비압축 파일만 압축 저장 기법을 적용한다. (그림 2) 는 논문에서 선택적 압축 알고리즘을 설계한 시스템 구조이다.



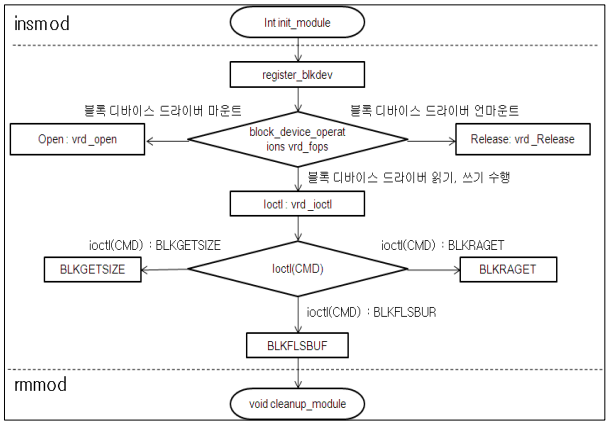
(그림 2) 시스템 구조



(그림 3) SDD 알고리즘 흐름도

사용자 수준의 압축 파일과 같이 압축률이 낮은 파일의 경우 압축 수행 후 원본 크기와 비슷한 크기의 압축 파일이 생성되어 압축 효율이 떨어질 뿐만 아니라 파일 중복 압축에 따른 불필요한 프로세서 낭비와 쓰기 수행시간까지 늘어나게 된다.

높은 압축률을 보이는 파일의 경우, 데이터 압축 시 소모되는 시간과 압축된 데이터를 디스크에 기록하는 시간이 원본 파일을 디스크에 기록하는 시간보다 수행시간이 단축 될 것이다. 압축률이 낮은 파일과 압축률이 높은 파일을 식별하여 선택적으로 압축하는 기법이 요구된다.



(그림 4) 커널 모듈 프로그램 구조

(그림 4)는 작성된 커널 모듈 프로그램의 구조이다. 리눅스 운영체제에서 블록 디바이스 드라이버는 ioctl()함수 사용하여 읽기, 쓰기 처리가 가능하다. ioctl()함수의 명령(cmd)은 통일이 되어있으며 아래의 <표 1>과 같다.

<표 1> 블록 디바이스 드라이버 명령어들

CMD	역할
BLKGETSIZE	섹터 수로 표현되는 블록 디바이스의 크기를 응용 프로그램에 전달
BLKFLSBUF	내부에 저장된 버퍼의 내용을 실제 블록 디바이스에 모두 써넣기를 요구
BLKRAGET	응용 프로그램에서 블록 디바이스에 설정되어있는 읽기값을 미리 얻어올 때 사용
BLKRASET	블록 디바이스의 미리 읽기 값을 설정
BLKRRPART	블록디바이스의 파티션 테이블을 다시 읽기를 요청
HDIO_GETGEO	블록 디바이스의 특성을 디스크 구조에 대한 형태로 응용 프로그램에 제공

ioctl()함수에서 BLKFLSBUF를 이용하여 내부에 저장된 버퍼의 내용을 블록디바이스 드라이버에 써넣으며 선택적 압축 알고리즘은 BLKFLSBUF에 구현한다.

```
int init_module(void)
{
    int lp;

    register_blkdev(MAJOR_NR, VRD_DEV_NAME, &vrd_fops);

    blk_queue_make_request(BLK_DEFAULT_QUEUE(MAJOR_NR), &vrd_make_request);

    read_ahead[MAJOR_NR]    = VRD_AHEAD;

    for( lp = 0; lp < VRD_MAX_DEVICES; lp++ )
        vrd_size[lp] = VRD_SIZE_KB;
    blk_size[MAJOR_NR] = vrd_size;

    vdisk[0] = vmalloc( VRD_SIZE );
    vdisk[1] = vmalloc( VRD_SIZE );

    for( lp = 0; lp < VRD_MAX_DEVICES; lp++ )
    {
        register_disk( NULL,
                       MKDEV(MAJOR_NR,lp),
                         1,
                       &vrd_fops,
                         VRD_SECTOR_TOTAL);
    }

    return 0;
}
```

(그림 5) init_module()

(그림 5)는 init_module() 프로그램이다. init_module()은 커널 콘솔에서 insmod 명령어를 이용하여 모듈 프로그램을 적재시킬 때 호출된다. blk_queue_make_request()함수를 사용 실제적인 입출력 통로를 등록하며, 실험 하였다 [1].

```
void cleanup_module(void)
{
    int lp;

    unregister_blkdev( MAJOR_NR, VRD_DEV_NAME );

    vfree( vdisk[0]);
    vfree( vdisk[1]);

    read_ahead   [MAJOR_NR] = 0;
    blksize_size[MAJOR_NR] = NULL;
    blk_size     [MAJOR_NR] = NULL;
}
```

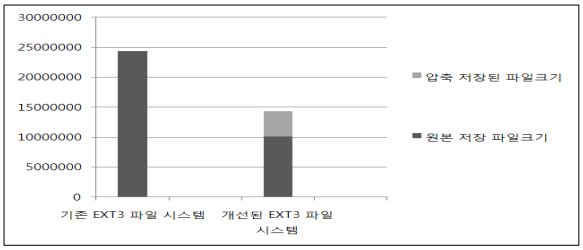
(그림 6) cleanup_module()

위 (그림 6)의 cleanup_module()은 리눅스의 콘솔에서 rmmod 명령어로 모듈 프로그램을 커널에서 제거할 때 호출 된다.

4. 실험 및 평가

본 장에서는 구현된 설계 내용을 바탕으로 실험한 내용의 성능 평가를 분석한다. 실험은 EXT3 파일 시스템의 기본적으로 사용하는 Oredered mode를 사용하였으며, zlib 1.2.5 버전을 사용하고 gcc를 이용하여 프로그래밍과 컴파일을 하였다. 실험은 기존의 파일 시스템과 SDD 압축 알고리즘이 적용된 파일 시스템의 성능 비교를 위해 멀티미디어 파일을 제외 후 무

작위로 선택된 50개의 파일을 임의로 적용하여 실험을 하였다. 성능 비교를 위해 기존의 파일 시스템과 선택적 압축 알고리즘이 포함된 파일 시스템의 전체 파일 크기를 비교 분석 하였다. 실험 파일의 앞부분 10240byte만큼 압축을 실행한 뒤 압축된 파일의 크기가 임계값으로 설정한 6800byte 보다 작을 경우 선택적 압축 저장기법을 통하여 쓰기 수행이 되도록 구현되어있다. 원본 파일크기의 약 2/3 이하로 압축률을 보일 경우이다. 기존 파일 시스템에서의 50개 파일의 총 크기는 24392704byte이며, 개선된 파일 시스템에서의 총 파일 크기는 14342756byte로 약 41%의 공간적 효율이 발생하였다. 전체 50개 파일 중 9개의 파일이 압축률이 낮아 원본저장이 되었고 41개의 파일이 선택적 압축 알고리즘을 통하여 압축 저장되었다. 압축 수행시간은 미비하여 공간적 효율에 비해 무시해도 될 정도이다. 아래의 (그림 7)은 실험한 결과를 그래프로 보여준다.



(그림 7) 파일 시스템별 총 파일 크기 비교

많은 실험 데이터의 결과를 바탕으로 일반 데이터 파일의 각 파일 포맷별 헤더 부분 압축 효율이 다르기 때문에 선택적 압축 알고리즘에서 앞부분 압축 시킬 크기와 임계값의 크기 조절로 보다 효율적인 임계값의 크기를 구할 수 있다.

5. 결론

본 논문에서는 SDD 알고리즘을 제아하여 파일 시스템 기능 및 성능 개선을 하였다. 특히, 압축을 수행하는 환경에 적합하도록 SDD 알고리즘을 제안하여 보다 효율적으로 압축 수행이 되도록 한다. 압축 효율이 좋지 않은 경우에 대한 회피를 통해서 불필요한 압축을 하지 않을 수 있도록 한다. 불필요한 연산을 줄임으로써 압축 알고리즘의 성능 향상을 꾀할 수 있다. 특히, 이미 압축 알고리즘이 적용이 된 데이타의 경우에 압축 알고리즘을 재차 적용하더라도 효율적인 압축이 되지 않는 경우가 많다. 이러한 경우에도 불필요한 압축을 하지 않도록 한다. 본 논문에서 제안하는 기능에 대해 실제 구현하고, 구현된 내용에 대해서 실험을 수행하였다.

실험은 기존의 파일 시스템과 SDD 압축 알고리즘이 적용된 파일 시스템의 성능 비교를 위해 멀티미디어 파일을 제외 후 무작위로 선택된 50개의 파일을 임의로 적용하여 실험을 하였다. 실험결과 대체적으로 본 논문에서 제안한 SDD 알고리즘의 성능이 우수함을 알 수 있었다.

참고문헌

[1] 이현철, “효율적인 메모리 사용을 위한 LZCode 기반의 압축 기법”, 아주대학교 석사학위논문, 2012.

[2] 전창규(Chang Kyu Jeon) 류경식(Kyeung Seek Lew) 김용득(Yong Deak Kim), “임베디드 시스템에서 실행 가능 압축 기법을 이용한 프로그램 초기 실행 속도 향상”, 電子工學會論文誌-CI, Vol.49 No.1, 2012.

[3] 이성헌(Seong-Heon Lee) 장승주(Seung-Ju Jang), “데이터 압축을 통한 효율적인 저장 공간 사용을 보장하는 저널링 파일 시스템 설계”, 한국정보과학회 학술발표논문집, Vol.38 No.2A, 2011.

[4] 조범석(Beom-Seok Joh) 김영로(Young-Ro Kim), “가변적 준무손실 압축 알고리즘 = Variable Near-Lossless Compression Algorithm”, 대한전자공학회 학술대회 논문집, Vol.2010 No.6, 2010.

[5] 조미남(Mi-Nam Cho) 지유강(Yoo-Kang Ji), "임베디드시스템을 위한 혼용텍스트 파일의 개선된 LZW 압축 알고리즘 구현", 한국콘텐츠학회논문지, Vol.10 No.12, 2010.