가변 블록을 이용한 사용자 파일 시스템설계

유영준*, 김병관*, 고영웅*
*한림대학교 컴퓨터공학과
e-mail:{willow72, kwani, yuko}@hallym.ac.kr

Design of user-level file system using variable-length blocks

Young-Jun Yoo*, Byung-Kwan Kim*, Young-Woong Ko*
*Dept of Computer Engineering, Dae-sung University

요 약

본 논문은 기존의 파일 시스템에서 파일의 일부를 수정했을 경우 수정된 위치 이후의 모든 블록이 수정되는 문제점을 개선하고자 가변 길이 블록 파일 시스템을 제안한다. 가변 길이 블록 파일 시스템은 파일 수정 시 수정이 발생한 데이터 블록만 새로 저장하고 나머지 블록의 상태는 유지함으로써 파일의 쓰기 연산을 최소화시킨다. 제안된 시스템을 사용할 경우 파일 수정 시 기존 파일 시스템보다 빠른 속도로 파일의 내용을 변경 시킬 수 있으며, 특히 대용량 파일에서 우수한 성능을 보인다.

1. 서론

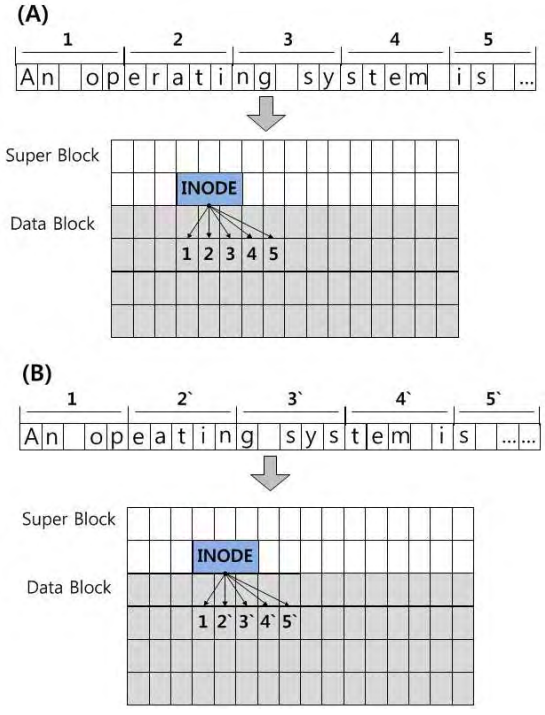
파일 시스템은 운영체제에서 문서, 이미지 그리고 다양한 실행 파일 등을 저장, 관리하는 역할을 하고 있다. 유저 영역에서 파일은 바이트 스트림(byte stream)으로 간주되며 파일 핸들러를 통하여 파일에 대한 연산을 수행한다. 유저 영역에서 파일에 대한 읽기 및 쓰기 연산은 커널 영역의 파일 시스템으로 전달되며 최종적으로 블록 디바이스에 저장된 데이터 블록에 대한 읽기 및 쓰기연산으로 변환된다. 이와 같은 파일 연산에서 데이터 스트림과 블록의 변환과정은 다양한 문제점을 발생시킨다. 예를 들어 데이터가 추가되거나 삭제되는 작업에서는 수정된 이후 블록의 내용은 모두 다시 써지는 문제점[1][2][3][4][5]이 존재한다. 특히 최근 많이 사용되는 플래시 기반의 저장 장치들은 데이터 수정에서 많은 오버헤드가 발생[6][7][8]하며, SSD에서 쓰기 연산은 장치의 가용기간에 영향을 미치는 중요한 요인이 된다.

본 논문에서는 파일에서 삽입/삭제 같은 수정이 발생할 경우 입출력 연산을 최소화하는 가변 길이 블록 파일 시스템을 제안한다. 가변 길이 블록 파일 시스템은 파일의 내용이 변경이 있을 경우, 해당 블록만 수정하고 나머지 블록은 유지한다. 이와 같이 쓰기 연산을 수행 할 경우 하나의 데이터 블록만 수정되므로 빠르게 파일 수정을 완료할 수 있다.

본 논문의 2장에서는 앞서 언급한 기존 파일 시스템에서 발생하는 파일 수정의 문제점을 설명하고, 3장에서 가변 길이 블록의 개념과 시뮬레이션 프로그램 구현을 설명한다. 그리고 4장에서 실험 및 평가로 타당성을 증명하며 5장 결론과 향후 연구로 맺는다.

1) 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2014R1A2A1A11054160)

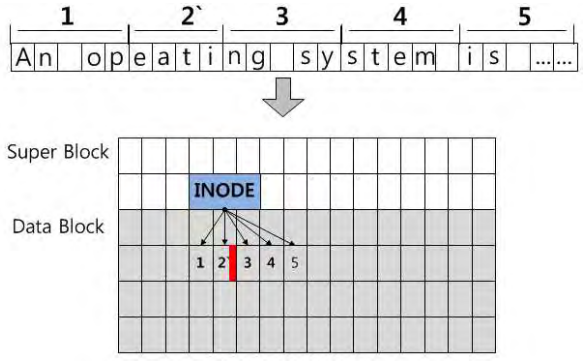
2. 기존 파일 시스템의 문제점



(그림 1) 기존 파일 시스템의 문제점. A는 파일의 수정 전, B는 파일에서 한 글자를 삭제한 후 각 블록의 저장된 모습을 보인다.

그림 1은 기존의 파일 시스템에서 파일의 수정을 보이고 있다. 만약 사용자가 파일의 내용에서 알파벳 'r'을 삭제했다고 가정하면, 수정 위치 이후의 블록 내용은 (B)와 같이 모두 수정되며, 결과적으로 디스크의 지속적인 입출력 연산이 발생하는것을 의미한다.

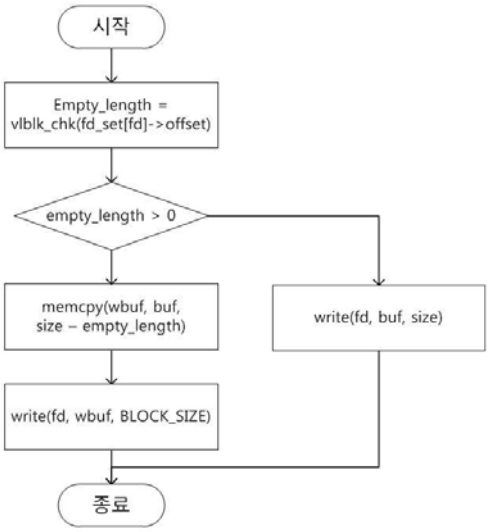
3. 시스템 설계 및 구현



(그림 2) 가변 길이 블록의 개념. 파일 내용 일부 삭제 시 블록의 크기보다 작은 데이터 조각이 저장된다.

그림 2는 가변 길이 블록의 개념을 보여준다. 그림 1의 (B)와 같은 기존의 파일 시스템에서 데이터 삭제 시 수정이 발생한 블록 이후에 존재하는 모든 블록을 수정하는 것이 아니라 수정이 발생한 블록의 데이터만 수정한다. 이 때 수정된 데이터의 크기는 블록의 크기보다 작게 저장되며, 변경된 블록의 정보와 해당 블록의 데이터 크기는 별도로 만들어진 *vb* 구조체에 저장된다.

본 논문에서는 가변 길이 블록 파일 시스템의 성능 개선 결과를 증명하기 위해 간단한 시뮬레이션 프로그램을 구현하였으며, 해당 프로그램은 단순히 읽기 쓰기 연산을 수행한다. 데이터 블록의 저장을 위해 수정된 위치를 알아내는 작업과 블록에 데이터를 저장하는 두 단계로 나뉜다. 현재 시스템에서 수정 위치는 파일의 앞부분에서 1바이트가 삭제됐다고 가정하고 있으며, 이 정보를 알아내는 작업은 현재 시스템에서 다루지 않는다. 수정된 블록에 대한 정보는 미리 *vb* 구조체에 저장되며, 쓰기 작업 시 이 구조체의 정보를 이용하여 가변 길이 블록을 적용한다. 블록에 내용을 쓰기 전 구조체를 확인하여 해당 블록에 가변 길이 블록을 적용해야 하는 경우 별도의 쓰기 함수를 제공한다. 이 때 블록의 크기보다 데이터의 크기가 작을 경우 나머지 부분은 빈 공간을 유지하며, 파일의 offset은 블록의 크기를 그대로 증가시킨다. 만약 해당 블록이 가변 길이 블록이 아닐 경우 쓰기 함수를 빠져 나오며 이후 블록의 정보는 수정하지 않는다. 파일에 대해 읽기 연산을 수행 할 때는 한 블록을 모두 읽은 후 해당 블록이 가변 길이 블록일 경우 *vb*구조체 내에 저장된 블록의 길이만 유저 프로그램에 출력한다.



(그림 3) vlfs_write 함수 순서도

그림 3은 앞서 설명한 가변 길이 블록을 적용하는 vlfs_write의 순서도이다. fd는 파일 디스크립터로 쓰여질 파일, buf는 저장을 하려는 데이터 버퍼, size는 버퍼의 크기를 나타낸다. vlblk_chk함수는 미리 저장된 가변 길이 블록의 정보를 찾고 만약 가변 길이 블록일 경우 빈 공간의 크기를 계산해 돌려준다. 그 후 빈 공간을 제외한 저장될 데이터만 실제 블록에 저장한다.

4. 실험 및 평가

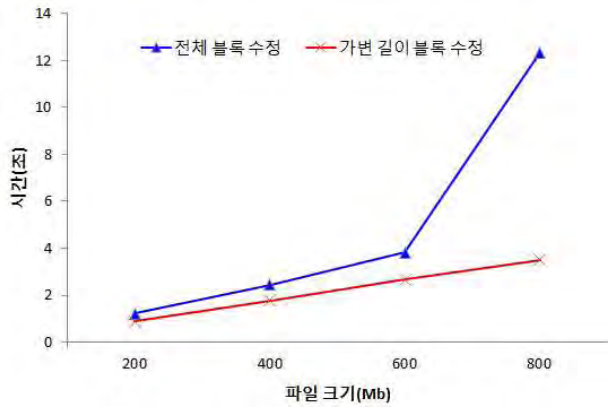
<표 1> 실험 환경

OS	centos 6.4 linux kernel 2.6.32
CPU	Intel® Xeon® Processor E5520 @ 2.27GHz
Memory	6G

표 1은 실험에서 사용된 환경을 보인다.

<표 2> 전체 블록 수정과 가변 길이 블록 수정 실험 결과

파일 크기 (단위M)	전체 블록 수정	가변 길이 블록 수정
200	1.229682	0.883019
400	2.440112	1.776746
600	3.8099916	2.660655
800	12.347353	3.517854



(그림 4) 전체 블록 수정과 가변 길이 블록 수정의 실험 결과

본 논문에서는 파일이 삭제 된 후 가변 길이 블록을 적용해 해당 블록의 데이터만 수정하는 방법을 제안하고 있으며, 제안한 방법은 쓰기 연산에서 특히 빠른 수행속도를 보인다. 이 실험을 위해 저장된 파일의 앞부분에서 1바이트만 수정해 다시 저장도록 구현하였으며, 그림 4와 표 2은은 쓰기 연산에 대한 수행속도를 나타낸다. 파일의 크기가 200M일 경우 각각 약 1.23초와 0.88초로 제안한 방법이 0.35초 빠른 결과를 나타내고, 특히 파일의 크기가 800M인 경우 각각 약 12.35초와 3.51초로 약 8.84초 빠른 결과를 나타낸다. 일반적으로 파일내용을 수정하고 저장할 경우 쓰기 연산을 반복적으로 호출하며 모든 블록을 다시 쓰며, 특히 파일의 크기가 커질수록 파일의 읽기 쓰기 연산을 훨씬 느려진다. 하지만 가변 길이 블록을 적용 할 경우 쓰기 연산에서 데이터가 수정된 블록만 새로 쓴다. 이후 블록의 데이터는 변경 여부만 확인하고 변경이 없으면 쓰기 함수를 종료하기 때문에 실제 파일 쓰기 연산 횟수는 변경된 블록 수에 따르며, 변경 블록 수가 적을 경우 훨씬 빠른 속도를 보인다. 또한, 그림 4를 통해 파일의 크기가 커짐에 따라 제안한 방법의 우수성을 알 수 있다. 이는 전체 블록의 내용을 수정하는 것이 아니라 데이터의 수정이 발생한 블록에서만 쓰기 연산을 수행하기 때문이다.

```
[willow72@localhost MissionFileSys]$ md5sum vlb_file rewrite
31ba230d37203f0655e774ca7f331b0d  vlb_file
31ba230d37203f0655e774ca7f331b0d  rewrite
[willow72@localhost MissionFileSys]$
```

(그림 5) 파일 내용 비교(800M)

그림5는 리눅스의 *md5sum*명령어를 이용해 수정된 파일의 내용을 비교하고 있다. 속도가 빠름에도 두 파일의 내용에는 차이가 없음을 확인할 수 있다.

5. 결론 및 향후연구

본 논문에서는 가변 길이 파일 시스템의 우수성을 증명하기 위해 시뮬레이션 프로그램을 작성하였다. 실험 결과 파일의 한 부분을 수정 했을 때, 모든 블록을 수정하는 것보다 데이터의 수정이 발생한 블록만 수정한 경우 수행속도가 빨랐으며 특히 800M 이상의 파일에서 큰 차이를 보이며 우수한 성능을 보였다.

향후 연구에서는 리눅스에서 많이 사용되는 파일 시스템은 EXT계열의 파일 시스템을 모방한 프로토타입을 구현할 예정이며, 차후 FUSE파일 시스템과 실제 커널 파일 시스템에도 적용할 예정이다.

참고문헌

[1] Brin, Sergey, James Davis, and Hector Garcia-Molina. "Copy detection mechanisms for digital documents." ACM SIGMOD Record. Vol. 24. No. 2. ACM, 1995.

[2] Joglekar, Usha A., Bhushan M. Jagtap, and Koninika B. Patil. "Deploying De-Duplication on Ext4 File System." International Journal of Engineering Research and Technology. Vol. 3. No. 1 (January-2014). ESRSA Publications, 2014.

[3] Mathur, Avantika, et al. "The new ext4 filesystem: current status and future plans." Proceedings of the Linux Symposium. Vol. 2. 2007.

[4] Cao, Mingming, Suparna Bhattacharya, and Ted Tso. "Ext4: The next generation of ext2/3 filesystem." 2007 Linux Storage & Filesystem Workshop. 2007.

[5] Djordjevic, Borislav, and Valentina Timcenko. "Ext4 file system in Linux Environment: Features and Performance Analysis." International Journal of Computers 1.6 (2012): 2012.

[6] Kawaguchi, Atsuo, Shingo Nishioka, and Hiroshi Motoda. "A Flash-Memory Based File System." USENIX. 1995.

[7] Lim, Seung-Ho. "Efficient IO and Layout Management Issue of Filesystem for Flash device." (2014).

[8] Lee, Changman, et al. "F2FS: A new file system for flash storage." Proceedings of the USENIX Conference on File and Storage Technologies (FAST). 2015.