Check for updates

Journal of KIIT. Vol. 19, No. 12, pp. 53-61, Dec. 31, 2021. pISSN 1598-8619, eISSN 2093-7571 53
http://dx.doi.org/10.14801/jkiit.2021.19.12.53

데이터 수정을 최소화하는 사용자 수준 파일 시스템 설계 및 구현

유영준*, 장성봉**, 고영웅***

Design and Implementation of User-Level File System for Minimizing Data Modification

Youngjun Yoo*, Sungbong Jang**, and Youngwoong Ko***

이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2021R1F1A106406911).

요 약

본 논문에서는 데이터 수정을 최소화하기 위하여 가변 블록 파일 시스템을 제안한다. 가변 블록은 블록 기반의 스토리지에 블록의 크기보다 작은 용량의 데이터를 저장할 수 있도록 파일 시스템의 메타데이터를 수정하여 만든 개념이다. 본 논문에서 제안하는 시스템의 유용성을 보이기 위해서 사용자 수준의 파일 시스템을 구현하였다. 실험 결과 데이터 읽기와 쓰기에 대해서 기존 시스템에 비해서 제안하는 가변 블록 기법이 높은 성능을 보이는 것을 확인하였다. 1MB 파일을 대상으로 파일 수정에 대한 실험을 수행한 결과 50배의 성능 향상을 보였다. 또한 1MB파일에서 가변 블록을 최대한 모두 사용하여 데이터 수정을 하였을 때에 기존 파일 시스템에 비해 10배 이상 빠른 수행 속도를 보였다.

Abstract

In this paper, we propose a variable block file system to minimize data modification. A variable block is a concept created by modifying the metadata of the file system so that data with a capacity smaller than the size of a block can be stored in block-based storage. To show the usefulness of the proposed system, we designed and implemented a user-level file system. As a result of the experiment, the proposed variable block scheme showed higher performance compared to the existing system for data reading and writing. In experiment on file modification for a 1MB file, the performance improved by 50 times. In addition, when data modification was performed using all the variable blocks in a 1MB file as much as possible, the execution speed was more than 10 times faster than that of the existing file system.

Keywords
file system, variable block, metadata, user-level, bumper area

* 한림대학교 컴퓨터공학
- ORCID: https://orcid.org/0000-0001-9578-1943

** 금오공과대학교 산학협력단 부교수
- ORCID: https://orcid.org/0000-0003-3187-6585

*** 한림대학교 컴퓨터공학 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-6292-0799

· Received: Nov. 08, 2021, Revised: Dec. 01, 2021, Accepted: Dec. 04, 2021

· Corresponding Author: Youngwoong Ko
Dept. of Software, Hallym University, HallymDaeHak-Gil 1, ChunCheon-Si, GangWon-Do, 24252, Korea
Tel.: +82-33-248-2329, Email: yuko@hallym.ac.kr

Ⅰ. 서 론

파일 시스템에서 쓰기 연산은 가장 느린 연산에 해당되며, 쓰기 성능을 향상시키기 위한 방법으로 일반적으로 버퍼(Buffer)를 사용한다. 쓰기 연산의 속도 문제를 해결하기 위해 다양한 방법의 연구 중에서는 하드웨어 기반의 다양한 기법이 제시되고 있다. 스토리지 클래스 메모리(Storage class memory)[1][2]는 비휘발성 메모리로 메인 메모리에 가까운 데이터 접근 속도를 보이며 바이트단위로 데이터를 처리할 수 있는 특징이 있다. 메모리로써 프로그램 수행을 위한 공간으로 분류되지만 비휘발성이라는 특징으로 인해 스토리지의 역할을 함께 수행하는 연구가 진행 중이다[3][4]. 스토리지 클래스 메모리로 인해 컴퓨팅 시스템은 기존의 구조와는 다른 새로운 구조들이 많이 고안되고 있으며 새로운 방식의 파일 인터페이스 개발의 가능성을 보였다.

하지만 아직 바이트 당 높은 가격을 보이며 용량역시 크지 않아 독단 적인 스토리지 역할로는 부족하다. 백업 프로그램이나 멀티미디어 편집 프로그램[5]과 같은 응용 프로그램에서는 데이터 편집 시 파일의 내용을 빠르게 바꾸기 위해, 데이터 수정이 발생한 이후 데이터만 다시 저장하도록 기능을 제공하고 있으나, 이러한 방법은 수정되는 파일의 위치에 따라 다른 성능을 보인다. 만약 수정되는 데이터의 위치가 파일의 전반부에 있을수록 더 많은 데이터가 다시 써진다. 즉, 삭제되는 데이터가 파일의 맨 앞에 있다면, 모든 데이터는 처음부터 다시 저장되는 기존의 시스템과 동일하다.

본 논문에서는 파일에서 데이터가 수정되는 상황에서 데이터가 다시 써지는 문제점을 해결하기 위해 가변 블록 파일 시스템(VBFS, Variable Block File System)을 제안한다. 가변 블록은 블록 기반의 스토리지에 가변적인 크기(Size)의 데이터를 저장하기 위하여 메터데이터를 일부 수정하는 기법이다. 이 때 블록의 남은 공간은 범퍼 영역이라는 패딩(Padding) 값을 추가함으로써 고정된 크기의 데이터를 유지한다. 본 논문에서 제안하는 시스템은 두 가지 측면에서 시스템 성능의 향상을 기대해 볼 수 있다. 첫 번째는 write()함수를 통해 파일의 내용을 쓰기 요청 보내는 과정을 빠르게 완료함으로써 어플리케이션의 응답속도를 향상시키는 것이다. 두 번째는 메타 데이터와 일부 수정이 발생한 블록만 디스크에 씀으로써 파일 시스템의 성능을 최적화한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 파일 시스템의 쓰기 성능을 향상시키기 위한 관련 연구들을 살펴본다. 3장에서는 사용자 수준 파일 시스템의 프로토타입 설계 및 구현에 대한 내용을 기술한다. 4장에서는 제안하는 시스템의 성능 평가를 진행하고 그 결과를 보인다. 그리고 5장에서는 구현 결과에 대한 평가 및 결론을 맺는다.

Ⅱ. 관련 연구

Write-Optimized Index(WOI)[6]를 이용한 파일 시스템은 쓰기에 최적화된 인덱스 구조인 B트리 구조를 사용함으로써 빈번한 마이크로 데이터 쓰기와 대용량 데이터 스캔의 성능을 크게 향상시켰다. 이로 인해 적은 데이터가 빈번하게 써지는 메일 서버나 바이러스를 확인하는 경우 파일 시스템의 성능이 크게 향상되어 주목을 받고 있다. BetrFS[7][8]는 기존 WOI를 적용한 파일 시스템의 코드 복잡성이나 FUSE[9]에서 중복적으로 쓰기 요청을 하는 기존의 시스템의 문제점을 해결하고 커널 수준에 적용한 파일 시스템이다.

또한 데이터 쓰기를 최적화 시키는 연구가 진행중이다. non-blocking 파일 쓰기 기법[10]을 통해 데이터 입출력 속도를 크게 줄였다. 사용자가 데이터를 디스크에 쓸 때, 쓰기 함수는 페이지 캐시를 먼저 탐색하며, 데이터가 없을 경우 디스크로부터 읽는 작업을 수행한다. 디스크에서 데이터를 읽는 속도는 상당히 느리므로, 커널은 페이지에 대해 패치 명령어를 디스크로 보내며, 데이터 패치가 완료되기 전, 페이지를 메모리의 버퍼 영역으로 복사한다. 그리고 패치가 완료되면 버퍼에 있던 내용을 페이지 캐시로 복사함으로써, 유저 어플리케이션이 대기하는 시간을 크게 줄인 기법이다.

NLE-FFS[11]는 스토리지 클래스 메모리에서 바이트 단위의 데이터 처리가 가능한 점을 이용해, 데이터 편집을 위한 새로운 파일 인터페이스들을 제안하였다. 하지만 이 시스템 콜을 스토리지 클래스 메모리에 최적화된 인터페이스이며 스토리지 클래

스 메모리는 바이트 당 가격이 비싼 단점이 있다.

변경된 데이터만 다시 쓰는 점에서 근본적인 아이디어는 중복제거[12][13][14][15]와 유사하다. 중복제거는 파일 저장 시 중복되는 데이터를 찾고, 중복되지 않은 데이터만 다시 씀으로써 데이터 쓰기와 디스크 공간을 최적화 하는 기법이다.

III. 사용자 수준의 가변 블록 파일 시스템 설계 및 구현

기존 파일 시스템에서는 파일의 일부를 수정했을 때 많은 양의 데이터를 다시 쓴다. 이러한 문제를 해결하기 위해 본 논문에서는 가변 블록 기법을 제안한다. 가변 블록은 일부 데이터가 변경된 상황에서 파일의 쓰기를 최소화하기 위해 수정된 블록만 다시 쓰는 기법이다. 이 때 블록은 삭제되는 데이터의 양에 따라 범퍼 영역(Bumper area)이라고 불리는 패딩 값을 추가하게 되며 일정 크기로 나눠진 블록임에도 블록 내의 데이터는 가변적인 길이를 갖는다. 이에 따라 이 블록을 가변 블록이라고 부르고 이러한 개념을 사용하는 파일 시스템을 가변 블록 파일 시스템이라고 부른다.

이번 장에서는 가변 블록을 파일 시스템에 적용하기에 앞서 가변 블록의 효율성을 확인하기 위해 사용자 수준 파일 시스템을 구현하였다. 커널 수준의 파일 시스템 분석과 구현은 많은 코드 라인으로 인해 오랜 시간이 걸리며 디버깅에 어려움이 존재한다. 또한 실행 중 시스템 콜이나 타이머 등 커널의 운영체제의 많은 요소들이 커널 수준의 개발을 더욱 더 복잡하게 만든다. 따라서 파일 시스템의 기능을 모방한 프로토타입을 먼저 구현한다. 프로토타입은 빠른 구현을 위해 파일 시스템의 핵심적 요소만 모방하였으며 파일 시스템의 비교적 간단한 구조인 ext2 파일 시스템의 구조를 사용하였다[16].

3.1 가변 블록 파일 시스템 개념

일반적으로 파일 시스템에서 파일을 편집할 때 디스크의 데이터는 유저 어플리케이션의 메모리로 복사된다. 그리고 일부 데이터가 삭제될 경우 삭제된 위치 이후 데이터는 모두 파일의 앞쪽으로 당겨져 데이터의 논리주소가 변하게 된다. 아주 적은 일부의 데이터가 삭제되어 많은 파일의 내용이 기존과 동일함에도 수정된 데이터 이후의 블록 내용은 달라진다. 따라서 데이터의 밀림을 막고 수정된 데이터의 위치를 최소화 시킬 방법이 필요하다.

그림 1은 본 논문에서 제시하는 핵심 아이디어인 가변 블록의 개념을 나타낸다. 가변 블록은 데이터를 저장하는 과정에서 삭제된 데이터만큼 범퍼 영역을 삽입하여 이후 데이터가 밀리는 현상을 방지한다. 이렇게 함으로써 이후 블록의 데이터들은 논리 주소를 그대로 유지하며, 오직 수정이 발생한 블록만 새로 저장함으로써 다시 쓰는 데이터를 최소화 할 수 있다. 예를 들어, 그림에서 E에 해당이 되는 부분의 일부 데이터가 삭제가 되었을 때에 다음 블록의 데이터가 변경이 되지 않도록 빈공간을 남겨둘 수 있게 파일 시스템의 메타데이터를 조작하는 방법이다.

3.2 사용자 수준 가변 블록 파일 시스템

사용자 수준 가변 블록 파일 시스템은 가변 블록을 커널에 적용하기에 앞서 가능성을 확인하기 위한 프로토타입 프로그램으로써, 파일 시스템의 기능을 모방한 작업을 수행한다. 커널 수준의 파일 시스템에는 많은 함수와 자료구조로 구성되어 있지만, 사용자 수준의 파일 시스템은 핵심적인 구성 요소만을 포함한다. 특히 사용자 수준 파일 시스템의 주된 목적은 일부 바이트가 삭제되는 제한적인 상황에서 모든 데이터를 다시 쓰는 기존의 시스템과의 차별성을 확인하며, 시스템의 성능 향상을 통해 커널 수준으로 적용 가능성을 확인하는 것이다. 따라서 사용자 수준 파일 시스템은 두 가지 방법을 통해 데이터를 저장한다.

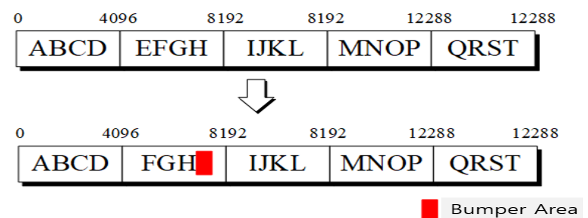

그림 1. 가변 블록 파일 시스템 개념
Fig. 1. Concept of variable block file system

첫 번째는 기존의 시스템과 동일하게 read()/write() 함수를 사용함으로써 수정 후 버퍼의 데이터를 모두 디스크에 쓰는 것이며 두 번째 방법은 가변 블록을 위한 별도의 라이브러리를 제공하는 것이다. 이 라이브러리는 미리 저장된 데이터 변경 정보에 따라 수정이 발생한 블록에 대해서만 쓰기 함수를 호출한다. 이 때 삭제된 데이터의 크기만큼 범퍼 영역을 추가하여 이후 데이터가 밀리는 현상을 막는다. 시스템의 자세한 계층도는 그림 2와 같다.

- Application: 파일의 일부를 수정하기 위한 어플리케이션으로써 파일 일부를 수정하기 특정 위치의 데이터를 삭제하는 별도의 명령어를 제공한다. 또한 가변 블록 적용을 위해 새로 추가된 VBFS 라이브러리를 호출해 가상 디스크 파일(Virtual disk file)에 수정된 파일 내용을 저장한다. 가변 블록은 블록 단위로 데이터를 처리하므로 편의상 모든 데이터는 읽기/쓰기 과정에서 4KB단위로 복사가 이뤄진다. 만약 가변 블록을 적용하지 않은 파일일 경우 기존 시스템 함수인 read/write함수를 호출함으로써, 변경된 데이터를 저장한다.

- VBFS 라이브러리: 가변 블록은 변경된 데이터가 디스크에 쓰거나 디스크로부터 읽는 과정에서 별도의 처리가 필요하다. 이러한 처리를 위해 기존의 read/write함수를 모방한 별도의 라이브러리 함수를 추가하였다. 이 함수의 자세한 기능은 알고리즘에서 설명한다.

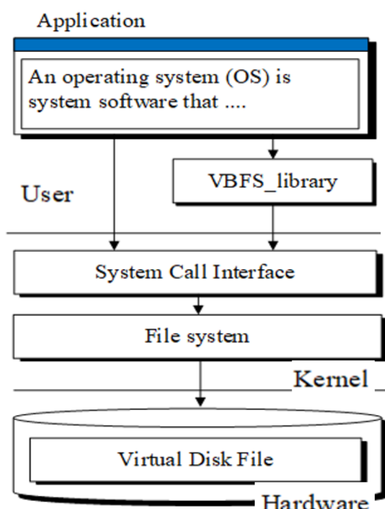

그림 2. VBFS 구조도
Fig. 2. VBFS architecture

-가상 디스크 파일: dd 명령어를 통해 만들어진 디스크 역할을 하는 파일이다. 기존의 파일 구조를 모방하여 만들었으며, 메타 데이터를 저장하는 메타 데이터 블록과 데이터를 저장하는 데이터 블록으로 나눠져 있다. 그리고 메타 데이터 블록은 다시 슈퍼 블록 영역과 아이노드 영역으로 나눠진다. 프로그램을 실행하면 먼저 메타 데이터 영역의 슈퍼 블록을 읽어 아이노드 정보를 확인한 후 메모리에 자료구조를 할당해 디스크의 아이노드 정보를 메모리에 저장한다. 또한, 실험용 파일을 저장한 디렉토리를 확인해 새로 추가된 파일을 가상의 디스크로 복사한다. 그리고 파일을 읽는 과정에서 함수를 통해 데이터 블록의 데이터를 읽는다.

- VBFS inode: ext2 아이노드를 모방해 만든 자료구조로 가상 디스크 파일에 저장되는 파일 데이터를 관리하기 이한 자료구조이다. 아이노드 구조와 유사하지만 프로토타입에 필요한 최소한의 메타데이터를 저장한다. 특히 가변 블록 정보를 저장하기 위한 VB 배열을 추가하였으며 이 자료구조는 삭제가 발생한 데이터의 오프셋과 삭제된 데이터의 크기를 저장한다. 이 정보는 데이터를 읽거나 쓰는 과정에서 범퍼 영역을 추가하거나 제거하는데 사용된다(본 연구에서는 프로토타입 구현을 위해 최대 10개의 가변 블록을 지원한다).

프로토타입은 직접 리눅스 파일 시스템에 데이터를 쓰는 것이 아닌 가상의 디스크 파일에 데이터를 쓰기 때문에 가상의 디스크 내에 데이터 블록이 어느 위치에 있는지 나타낼 별도의 자료구조가 필요하다. 이를 위해 ext2 파일 시스템의 아이노드 구조를 모방한 VBFS_inode를 추가하였다. 이 자료구조는 가상 디스크 파일의 메타 데이터 블록 영역에 저장되며, 파일의 이름, 크기, 생성 날짜 등을 포함한 메타 데이터와 파일의 내용이 저장된 데이터 블록의 인덱스를 저장한다. 추가적으로 수정된 블록의 정보를 저장한 VB가 추가됐으며, 인덱스와 범퍼 영역의 크기를 저장한다. 그림 3은 VBFS_inode 자료구조의 모습을 보이고 있다.

가변 길이 블록을 적용하기 위해 지원하는 라이브러리는 일반적인 파일 라이브러리에 포함되어 있

는 파일 열기(vbfs_open), 읽기(vbfs_read), 쓰기(vbfs_write), 닫기(vbfs_close) 등을 포함하며, 특히 VB 자료구조에 데이터를 저장할 set_vbinfo()함수가 추가되었다. 이 함수의 역할은 단순히 매개변수로 입력된 삭제된 데이터의 위치를 블록 인덱스로 변환해 삭제된 데이터의 크기와 함께 저장한다. 유저 어플리케이션에서는 가변 라이브러리 함수를 사용하기 전, set_vbinfo()를 호출해야만 가변 블록을 적용할 수 있으며, 가변 블록 정보를 저장한 후에는 vbfs_write() 함수를 호출하여, 수정된 파일의 내용을 저장한다 (편의상 유저 어플리케이션에서는 4KB단위로 데이터를 쓴다고 가정한다).

3.3 가변 블록 시스템 입출력 모듈

제안하는 시스템에서는 vbfs_write()를 이용하여 파일의 쓰기를 처리한다. 매개변수로는 파일 식별자(fd)와, 버퍼 그리고 데이터의 크기가 전달된다. 데이터 처리에 앞서 쓰기 함수는 가상 디스크 파일의 식별자(disk_fd)를 얻어 온다. 가상 파일의 식별자는 사용자 수준 파일 시스템이 실행될 때 최기화 단계에서 가상 파일을 열고 이에 대한 식별자를 프로그램 종료 시까지 변수를 유지한다. 그리고 본격적으로 가변 블록에 대한 처리를 수행하며 파일의 오프셋을 이용해 블록의 인덱스를 구한다. 그리고 이 인덱스를 미리 저장된 가변 블록 배열의 인덱스들과 비교하여 만약 배열에 동일한 인덱스가 있다면 가변 블록 처리를 수행한다. 버퍼로부터 받은 데이터 중 유효한 데이터의 크기만큼만 임시 버퍼로 복사하며 남은 빈 영역은 범퍼 영역으로 처리한다.

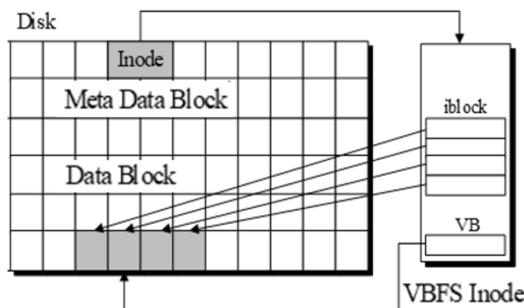

그림 3 VBFS inode 구조
Fig. 3. VBFS inode architecture

그리고 이 임시 버퍼를 기존 데이터 블록에 덮어씀으로써 파일의 내용을 변경한다. 이 때, 유효 데이터와 범퍼 영역으로 포함해 하나의 블록 크기만큼의 데이터가 write()함수를 통해 다시 써지며 오프셋은 다음 블록을 위해 블록의 크기만큼 증가한다. 하지만 사용자 수준에는 리턴 값을 유효 데이터만큼만 전달한다. 유저 어플리케이션은 받은 리턴 값만큼 버퍼의 주소를 이동시키며 이후 데이터를 계속해서 디스크에 쓰기 요청을 보낸다.

읽기 함수는 vlfs_read()로 구현되어 있다. 쓰기 함수 마찬가지로 파일 식별자, 버퍼 그리고 읽을 데이터의 크기를 매개변수로 받는다. 읽기 함수는 쓰기 함수와 달리 디스크의 데이터를 모두 읽어 어플리케이션에 복사해야 하기 때문에 모든 경우에서 디스크의 데이터를 읽어야 한다. 하지만 인덱스를 비교하였을 때 읽은 블록의 인덱스에 범퍼가 포함되어 있다면 tmp 변수를 통해 블록 내용을 읽고, 범퍼 영역을 제외한 데이터만 어플리케이션의 버퍼 변수에 복사한다. 반면 범퍼 영역을 포함하지 않은 데이터라면 블록의 데이터를 바로 어플리케이션의 버퍼로 복사한다. 이 때, 리턴 값은 범퍼 영역을 제외한 유효 데이터의 크기이며, 유저 어플리케이션은 다음 데이터 복사를 위해 이 리턴 값만큼 메모리의 오프셋을 증가시킨다.

Ⅳ. 성능 평가

4.1 실험 환경 구성

이번 장에서는 사용자 수준 가변 블록 파일 시스템의 성능을 측정한 결과를 보인다. 실험을 위해 우분투(Ubuntu)에 리눅스 커널 4.9.116버전을 설치해 실험을 진행하였으며, 컴퓨터의 사양은 중앙 처리 장치는 Intel® CoreTM i5-3470 Processor를 사용하고 메인 메모리는 8GB이다. 리눅스에서는 백그라운드 프로그램인 데몬(Daemon)들에 의해 지속적인 I/O가 발생하며, 이 I/O의 양은 불규칙한 양과 타이밍을 보인다. 따라서 최소한의 필요한 데몬만 실행하며, 네트워크 선을 제거한 상태로 실험을 진행하였다.

4.2 바이트 삭제 실험

먼저 처음 논문의 가정에 따라 일부 데이터가 삭제되는 실험을 진행하였다. 가변 블록의 목적은 대용량 파일을 수정할 경우 빠르고 적은 데이터 쓰기 요청으로 파일을 수정함을 목적으로 하지만 가변 블록의 성능 향상으로 인해 대용량의 파일과는 직접적인 비교가 힘들다. 따라서 적은 용량의 파일과 비교하는 실험을 진행하였다. 사용자 수준 가변 블록은 하나의 블록보다 작은 데이터 삭제를 제공하며 블록보다 작은 데이터가 업데이트 될 경우 바이트 수와 상관없이 하나의 블록이 업데이트 된다.

따라서 편의상 하나의 바이트를 삭제하는 실험을 진행하였다. 또한 기존 시스템의 경우 메모리의 모든 데이터 복사가 이뤄지므로 파일의 전반부에서 데이터가 삭제되는 실험을 진행한 반면 가변 블록은 여러 위치에서 데이터를 삭제하였다.

그림 4는 기존 시스템과 가변 블록의 시뮬레이션 결과를 보인다. 기존 시스템의 경우, 4KB 파일에서 일부 데이터를 삭제하고 다시 저장할 경우 약 24KB의 데이터가 써지며, 8KB 파일의 경우 28KB의 쓰기 연산이 발생한다. 즉, 20KB가량이 가상 디스크 파일과 아이노드 업데이트에 의해 발생하며, 추가적으로 다시 쓰이는 블록 수만큼 데이터 쓰기가 발생한다.

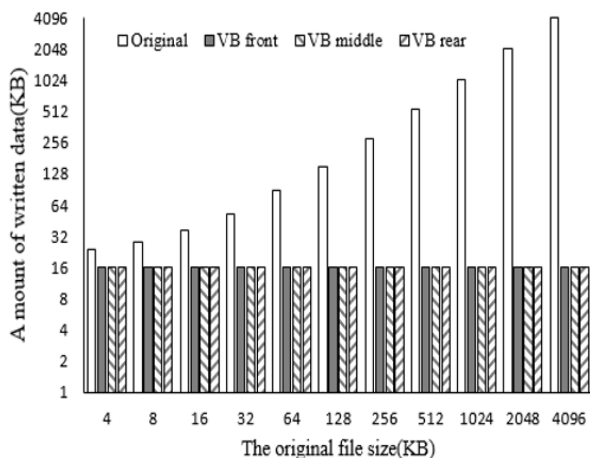

그림 4. 파일 크기와 데이터 삭제 위치에 따른 데이터 쓰기 실험 결과

Fig. 4. Experiment result for data write according to file size and data deletion location

반면, 가변 블록의 경우 다시 쓰이는 데이터와 상관없이 오직 데이터 수정이 발생한 블록만 업데이트됨으로써, 모든 원본 파일에 대해 16KB의 쓰기 연산이 발생하며, 기존의 시스템에 비해 파일의 크기가 커짐에 따라 우수한 성능 향상을 보임을 확인할 수 있다.

4.3 수정 위치에 따른 시스템 성능

표 1은 기존 시스템과 가변 블록 시스템에서 파일 내용 변경에 소요되는 시간을 측정한 결과를 보인다. 가변 블록의 경우 자세한 성능 측정을 보이기 위해 각 파일의 위치를 앞부분(FRT), 중간(MID), 끝부분(RER)으로 나눠 실험을 진행하였다.

실험 결과 약간의 오차는 있지만, 동일한 파일 크기와 수정 위치에 대해서는 유사한 시간이 소요됨을 확인 할 수 있다.

표 1. 파일 수정 수행시간 (단위 : ㎲)

Table 1. Execution time for file modification (unit:: ㎲)

FILE SIZE (KB)		4	8	16	32	64	128	256	512	1024
ORG		52	89	167	322	623	1278	2598	4963	9747
VB	FRT	51	51	52	54	58	66	85	122	187
	MID	51	51	52	54	58	68	85	122	189
	RER	51	51	52	55	58	68	85	122	189

또한, 파일의 크기에 따라 수행 시간이 점차 증가하는 모습을 보인다. 하지만 기존 시스템이 파일의 크기에 비례해 약 2배씩 수행 속도가 느려졌던 반면, 가변 블록에서는 비교적 적은 증가율을 보이며, 가장 작은 대상 파일인 4KB와 가장 큰 파일인 1MB파일이 3.7배의 차이를 보인다. 기존 시스템과 비교했을 때에도, 가변 블록은 우수한 성능을 보인다. 4KB파일 내용을 수정할 경우 두 시스템 모두 하나의 블록만 수정하기 때문에 각각 52.7㎲와 51㎲로 유사한 수행속도를 보인다.

하지만 파일의 크기가 커짐에 따라 수행속도는 점차 격차를 보인다. 32KB파일의 경우 각각 322.8㎲와 54~55㎲로 가변 블록을 적용할 경우 약 5.8초가 빨라지며, 128KB파일의 경우 기존 시스템은 1밀리초를 넘는 반면, 가변 블록은 평균 67.5초를 보이며 18.9배의 속도차이를 보인다. 특히 실험 대상 파

일 중 가장 큰 1MB파일에서는 약 51.8배의 차이를 보이며 월등히 빠른 속도를 보이고 있다.

4.4 가변 블록 수에 따른 시스템 성능

앞선 실험에서 보는 바와 같이 가변 블록 파일 시스템의 성능은 파일의 크기와는 상관없다. 하지만 수정되는 블록이 증가함에 따라 다시 쓰이는 데이터의 양과 시간은 증가한다.

그림 5는 가변 블록의 수가 시스템의 성능에 미치는 영향을 보인다. 실험을 위해 1MB파일에서 가변 블록의 개수를 늘려가며 최대 10개의 가변 블록을 실험하였다. 실험 결과 쓰인 데이터의 양은 하나의 블록을 업데이트할 때 앞선 실험과 마찬가지로 16KB의 데이터를 다시 쓰며 수정되는 블록의 수가 증가함에 따라 4KB씩 증가한다. 시간 또한 가변 블록의 수가 증가함에 따라 점차 증가하는 모습을 보이고 있다. 하나의 가변 블록을 적용할 경우 189.56㎲가 소요되며 5개의 가변 블록을 적용할 경우 343.68㎲, 그리고 10개의 가변 블록을 적용할 경우 532.74㎲가 소요된다.

이와 같이 가변 블록의 수가 증가함은 다시 쓰이는 블록의 수가 증가함을 의미하며 많은 시간을 소요한다. 하지만 기존 시스템에서 1MB 파일에서 일부 바이트를 지우고 다시 저장할 경우 9ms가 소요되는 것에 비하면 월등이 좋은 성능이라고 할 수 있다.

V. 결론 및 향후 과제

본 논문에서는 가변 블록을 소개하며 사용자 수준의 가변 블록 파일 시스템의 구조 및 주요 알고리즘을 설명하였다. 기존 전통적인 입출력 시스템의 경우 메타 데이터를 포함해 블록을 다시 쓰며 많은 데이터 복사가 발생했다. 하지만 가변 블록의 경우 수정한 블록만 다시 쓰기 때문에 매우 높은 성능을 보였다. 실험 대상 파일 내에서 가장 큰 파일 크기인 1MB파일에서 약 51.8배의 성능차이를 보이며 파일 수정에 아주 빠른 수행속도를 보였다. 가변 블록은 대용량 파일의 수정을 가정으로 시작한 아이디어이지만 적은 용량의 파일에서도 월등한 속도차이를 보이며 대용량 파일에서도 우수한 성능 차이를 유추해 볼 수 있는 결과이다. 또한 최대 10개의 가변 블록을 지원하며 1MB파일에서 모든 가변 블록을 다 사용하여 많은 위치에서 데이터 수정이 발생하더라도 532.74㎲의 수행속도를 보이며 기존 파일 시스템에 비해 빠른 수행속도를 보였다.

관련 연구에서 논의되었던 연구 기법 중에서 WOI[6]와 같이 파일 쓰기에 최적화할 수 있는 메타데이터를 사용하거나, 또는 중복 제거 및 파일 압축으로 성능을 향상하는 연구들이[16] 있었다. 하지만, 제안하는 연구와 같이 메타데이터를 이용한 가변 블록 방식과 유사한 연구 사례는 없는 것으로 판단된다.

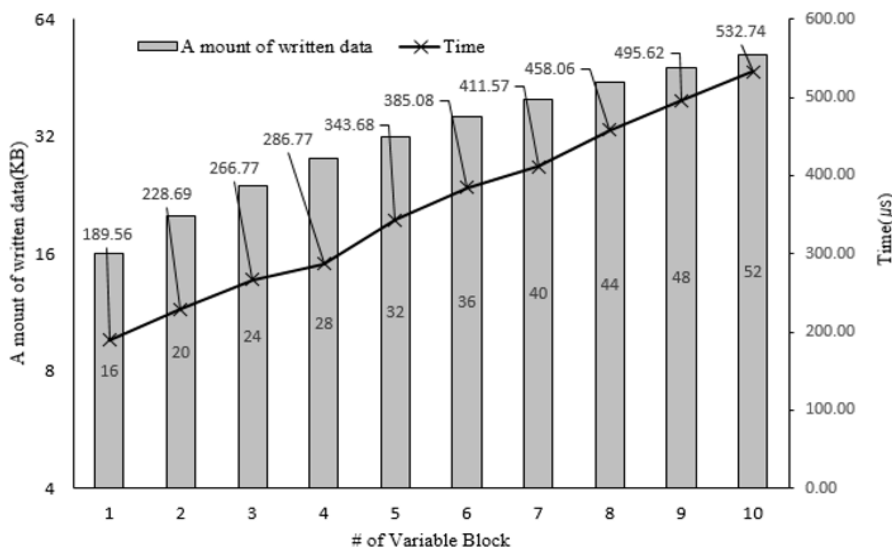

그림 5. 가변 블록 수에 따른 실험 결과 (1MB 파일)
Fig. 5. Experiment result for data write according to the number of variable blocks (1MB file)

본 논문에서 제안하는 사용자 수준 가변 블록 파일 시스템은 제한적인 상황을 가정하고 있다. 일부 바이트 단위만 수정하며 최대 10개 이내의 가변 블록을 지원함으로써 사용자가 많은 데이터를 삭제하는데 어려움이 있다. 또한 사용자 수준에서 시뮬레이션만 하는 프로그램으로써 실제 유저의 데이터를 직접적으로 수정하기에는 무리가 있다. 향후 본 연구의 제한점을 개선할 수 있는 커널 수준의 가변 블록 파일 시스템에 대한 연구를 진행하고자 한다.

References

[1] Burr G. W. et al., "Phase change memory technology", Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena, Vol. 28, No. 2, pp. 223-262, Mar. 2010. https://doi.org/10.1116/1.3301579.

[2] Subramanya R. Dulloor et al., "System software for persistent memory", In Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14). Association for Computing Machinery, New York, NY, USA, pp. 1-15, Apr. 2014. https://doi.org/10.1145/2592798.2592814.

[3] Liu Shiyong, Zhichao Cao, Zhongwen Guo, Guohua Wang, Xupeng Wang, Zhijin Qiu, and Xukun Qin, "NVMTFS: A Non-Volatile Memory Adaptive File System for Tiered Storage System", 2018 4th International Conference on Big Data Computing and Communications (BIGCOM), Chicago, IL, USA, pp. 201-206, Aug. 2018. https://doi.org/10.1109/BIGCOM.2018.00039.

[4] Jiaxin Ou, Jiwu Shu, and Youyou Lu, "A high performance file system for non-volatile main memory", In Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16). Association for Computing Machinery, New York, NY, USA, pp. 1–16, Apr. 2016. https://doi.org/10.1145/2901318.2901324.

[5] Seung Wan Jung, Seok Young Ko, Young Jin Nam, and Dae-Wha Seo, "Block Link file system supporting fast editing/writing for large-sized multimedia files in multimedia devices", 2012 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, pp. 457-458, Jan. 2012. https://doi.org/10.1109/ICCE.2012.6161942.

[6] Lu, L. Physical Separation in Modern Storage Systems, The University of Wisconsin-Madison., 2016.

[7] William Jannen, et al., "BetrFS: a right-optimized write-optimized file system", In Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15), USA, pp. 301–315, Feb. 2015.

[8] Jun Yuan, Yet al., "Optimizing every operation in a write-optimized file system", In Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST'16), USA, pp. 1–14, Feb. 2016.

[9] https://github.com/libfuse/libfuse. [accessed: Nov. 02, 2021]

[10] Daniel Campello, et al., "Non-blocking writes to files", In Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15). USENIX Association, USA, pp. 151–165, Feb. 2015.

[11] M. Seo, S. Ko, Y. Park, and K. H. Park, "NLE-FFS: a flash file system with PRAM for non-linear editing", in IEEE Transactions on Consumer Electronics, Vol. 55, No. 4, pp. 2016-2024, Nov. 2009. https://doi.org/10.1109/TCE.2009.5373764.

[12] FU, Min, et al., "Design Tradeoffs for Data Deduplication Performance in Backup Workloads", n Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15), USA, pp. 331-344, Feb. 2015.

[13] WANG, Chundong, et al., "NV-Dedup: High-Performance Inline Deduplication for Non-Volatile Memory", IEEE Transactions on Computers, Vol. 67, No. 5, pp. 658-671, 2018. https://doi.org/

10.1109/TC.2017.2774270.

[14] Byung Kwan Kim, Young Woong Ko, and Kwang Mo Lee, "Performance Enhancement for Data Deduplication Server Using Bloom Filter", The Journal of Korean Institute of Information Technology, Vol. 12, No. 4, pp. 129-136, 2014. http://dx.doi.org/10.14801/kiitr.2014.12.4.129.

[15] Mohamed, S. M. A., Wang, Y., "A survey on novel classification of deduplication storage systems", Distrib Parallel Databases, Vol. 39, No. 12, pp. 201-230, Mar. 2021. https://link.springer.com/article/10.1007/s10619-020-07301-2.

[16] Wang, K.., EXT2 File System. https://doi.org/10.1007/978-3-319-92429-8_11. 2018.

저자소개

유 영 준 (Youngjun Yoo)

2012년 2월 : 한림대학교 컴퓨터공학(공학사)
2014년 2월 : 한림대학교 대학원 컴퓨터공학(공학석사)
2019년 2월 : 한림대학교 대학원 컴퓨터공학(공학박사)
2019년 3월 ~ 현재 : (주)데이터커맨드 연구원
관심분야 : 운영체제, 클라우드 스토리지, 가상화시스템

장 성 봉 (Sungbong Jang)

1997년 2월 : 고려대학교 컴퓨터학과(이학사)
1999년 2월 : 고려대학교 대학원 컴퓨터학과(이학석사)
2010년 2월 : 고려대학교 대학원 컴퓨터학과(이학박사)
2012년 9월 ~ 현재 : 금오공과대학교 산학협력단 부교수
관심분야 : 빅데이터 익명화, 기계 학습, 딥러닝, 모바일 증강현실

고 영 웅 (Youngwoong Ko)

1997년 2월 : 고려대학교 컴퓨터학과(이학사)
1999년 2월 : 고려대학교 대학원 컴퓨터학과(이학석사)
2003년 2월 : 고려대학교 대학원 컴퓨터학과(이학박사)
2003년 9월 ~ 현재 : 한림대학교 컴퓨터공학 교수
관심분야 : 운영체제, 임베디드 시스템, 가상화, 클라우드 스토리지