Journal of The Institute of Electronics and Information Engineers Vol.53, NO.12, December 2016 https://doi.org/10.5573/ieie.2016.53.12.094
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2016-53-12-12

논리 볼륨 매니저를 이용한 파일 우선순위 기반의 하이브리드 저장장치 관리 시스템

(Priority-Based Hybrid File Storage Management System Using Logical Volume Manager)

최 훈 하*, 김 현 지*, 노 재 춘**

(Hoonha Choi, Hyeunjee Kim, and Jaechun No)

요 약

최근 고성능의 SSD(Solid State Drive)가 등장하면서 단일노드의 입출력 성능이 대폭 상향되었다. 이에 SSD를 기반으로 하는 차세대 저장장치 플랫폼이 주목을 받게 되었고, 고속 연산이 필요한 서버 또는 데이터 센터 등에서 SSD 기반 저장장치를 구축하는 시도가 증가하고 있다. 그러나 SSD는 단위용량당 비용이 고가이기 때문에, SSD 기반 저장장치를 구성하기에는 아직 어려움이 있다. 따라서, 본 논문은 저가와 큰 용량이 장점인 HDD(Hard Disk Drive)와 SSD가 통합된 저장장치에서 파일을 관리하는 소프트웨어 HyPLVM(Hybrid Priority Logical Volume Manager)을 소개한다. HyPLVM은 사용자가 접근하는 파일, 디렉터리를 분석하여 파일에 우선순위를 부여하고, 이 우선순위 값에 따라 높으면 SSD에, 낮으면 HDD에 저장되도록 관리한다. 이로써, 접근빈도가 많은 파일에 한에서 SSD로 구성된 저장장치와 버금가는 입출력 성능을 산출하면서 저장장치 구축비용을 절감한다.

Abstract

Recently, the I/O performance of a single node is rapidly improving due to the advent of high-performance SSD. As a result, the next-generation storage platform based on SSD has received a great deal of attention and such storage platforms are increasingly adopted to commodity servers or data centers that look for the high-bandwidth computation and I/O. However, building all SSD-based storage platform may not be cost-effective because the price per storage capacity is very high as compared to that of HDD. In this paper. we propose a hybrid file management solution, called HyPLVM(Hybrid Priority Logical Volume Manager), which combines the strength of SSD with the desirable aspects of low-price, high-storage capacity HDD. HyPLVM prioritizes the files and directories to be accessed by users, in order to determine the target storage device (SSD/HDD) in which files are allocated, while mitigating the cost of building storage platforms.

Keywords : Solid state drive, Hybrid drive, File priority, File replacement, Logical volume manager

Ⅰ. 서 론

최근 단일노드의 입출력 성능을 향상할 수 있는 flash-SSD[2~3], DRAM-SSD[4], PCM-SSD[5] 등 고성능 저장장치가 등장하면서 SSD를 기반으로 하는 차세대 저장장치 플랫폼이 주목을 받고 있다. 기존에는 높은 성능의 저장장치를 구축하기 위해 다양한 컴퓨팅 기술들[1]을 사용했지만, SSD 발전으로 인해 높은 입출력 성능을 산출할 수 있게 됨으로써, 고속 연산이 필요로 하는 서버 또는 데이터 센터 등에서 SSD 기반 저장장치

* 정회원, 세종대학교 컴퓨터공학과 (Department of Computer Engineering, SejongUniversity)
** 교신저자, 세종대학교 컴퓨터공학과 (Department of Computer Engineering, Sejong University)
© Corresponding Author (E-mail: jano@sejong.ac.kr)
※ 본 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2014R1A2A2A01002614).
Received ; August 20, 2016 Revised ; September 22, 2016 Accepted ; November 22, 2016

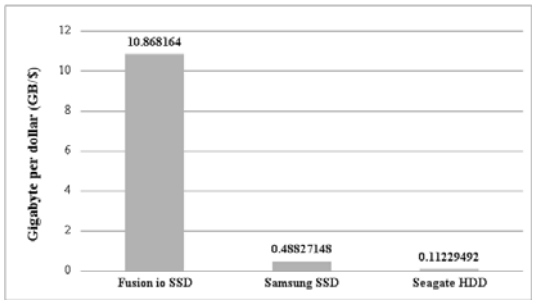

그림 1. 단위용량당 비용 비교
Fig. 1. Cost per storage capacity comparison.

를 구축하는 시도가 증가하고 있다[6].

HDD는 시크타임을 포함한 기계적 오버헤드가 발생하기 때문에, 임의적 읽기와 쓰기가 느리고, 전력소모가 크며, 외부 진동에 약하다는 단점이 있다[7]. 그에 비해, SSD는 전자기적으로 동작해 소비 전력과 소음이 적은 장점이 있다. 또한, 플래시 메모리로 구성되어있어 외부의 진동에 강하고, 높은 입출력 성능을 산출할 수 있어 HDD를 대체하는 저장매체로 자리 잡고 있다[10].

그러나 SSD는 단위용량당 비용이 HDD와 비교하면 상당히 고가이기 때문에, SSD로 저장장치를 구성하기에는 아직 많은 어려움이 있다. 한 예로, SSD 저장장치에 주로 쓰지 않는 파일이 지속해서 누적된다면, 정작 주로 쓸 파일이 저장될 공간이 줄어들어 자원낭비가 발생하게 된다. SSD의 자원낭비는 단위용량당 비용이 고가인 만큼 전체 시스템에 적지 않은 비용손실이 초래되기 때문에 해결책이 필요하다. 그림 1은 본 논문에서 사용한 PCIe SSD와 SATAIII SSD, HDD의 단위용량당 비용(GB/$)을 비교하였다.

본 논문은 SSD와 HDD가 통합된 단일 노드를 관리하는 소프트웨어인 HyPLVM을 소개한다. HyPLVM는 먼저 사용자 레벨에 해당하는 모든 파일의 접근빈도를 분석한다. 분석을 통해 파일마다 우선순위가 부여되고, 우선순위에 따라 SSD와 HDD에 파일을 저장한다. 또한, HyPLVM이 우선순위 높은 파일만 SSD에 저장되도록 단일노드를 관리해 줌으로써, 저비용으로 고성능 저장장치를 구성할 수 있다.

본 논문의 구성은 2장에서 논문의 이해를 돕기 위한 관련 기술을 설명한다. 다음으로 3장에서는 본 연구에서 구현한 HyPLVM을 소개하고, 4장에서는 HyPLVM의 성능을 측정한다. 마지막으로 5장에서는 결론을 맺고 향후 연구방향을 제시한다.

Ⅱ. 관 련 연 구

2.1 하이브리드 저장장치

최근 SSD의 우수한 입출력 성능 장점과 HDD의 저가의 큰 용량을 혼합한 하이브리드 저장장치와 소프트웨어에 대한 연구가 활발히 진행되고 있다[14~15]. 하이브리드 저장장치 관련 연구들이 초점을 맞추고 있는 것은 (1) SSD의 빠른 속도는 HDD의 느린 속도를, HDD의 많은 용량은 SSD의 적은 용량을 서로 보완하는 활용법[8]과, (2) 기계기반인 HDD의 반영구적 수명과 다르게 플래시 메모리 기반인 SSD는 비교적 짧은 수명을 가진 점[9]이다.

(1)은 논문[8]에서 Ext4 파일시스템의 메타데이터는 분산 저장되기 때문에, HDD를 Ext4 파일시스템으로 사용하면 느린 임의적 접근이 빈번하게 발생하게 되고, 이로 인한 HDD 성능저하를 지적하였다. 이를 해결하기 위해 논문[8]은 메타데이터를 SSD에 할당하고 접근함으로써, HDD의 느린 임의적 접근 발생 빈도를 줄여 HDD의 성능을 향상하는 SSD-HDD 하이브리드 파일시스템을 제안하였다.

(2)는 SSD의 쓰기가 빈번할 경우 수명 단축과 관련되어 있다. 이를 해결하기 위하여 논문[8]은 SSD를 캐시로 사용할 때 SSD의 데이터를 변경시키는 블록 교체 기법의 중요함을 강조 하였고, LARC, LRU 알고리즘의 문제를 파악해 블록 재사용 간격을 고려하는 블록 교체 기법 BRI(Block Replacement Algorithm with Interval Reuse)을 제안하였다.

2.2 LVM

LVM(Logical Volume Manager)는 다수의 저장장치 혹은 파티션을 하나의 논리적 저장장치로 구성해 동적으로 저장장치를 추가하거나 파티션 구성을 쉽게 변경할 수 있는 유연성을 가지고 있다. 저장장치 혹은 파티션으로 하나 이상의 PV(Physical Volume)을 생성하고, 만들어진 PV을 VG(Volume Group)으로 묶어 논리적 저장장치를 구성한다. 구성된 논리적 저장장치로부터 공간을 매핑 받아 LV(Logical Volume)을 생성할 수 있으며, 매핑 된 LV는 물리적 저장장치와 같이 장치명으로 접근할 수 있고, 원하는 파일시스템으로 포맷해 디렉터리에 매핑하고 사용할 수 있다[11].

물리적 저장장치로 구성된 시스템에서 파티션 정보를 변경하거나 저장장치의 증설이 필요하다면 백업이 필요하기 때문에 많은 시간이 소요된다. 반면에, LVM

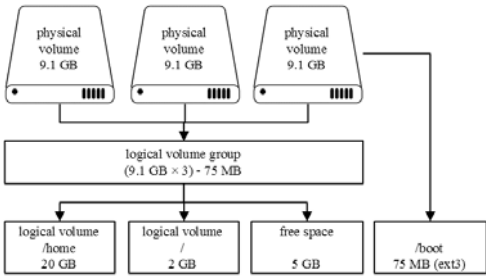

그림 2. LVM 내부구조
Fig. 2. LVM internal structure.

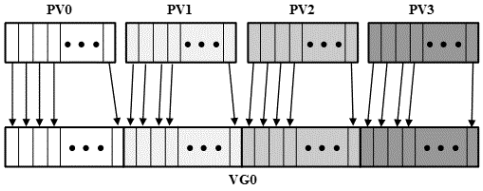

그림 3. LVM 순차매핑
Fig. 3. LVM linear mapping.

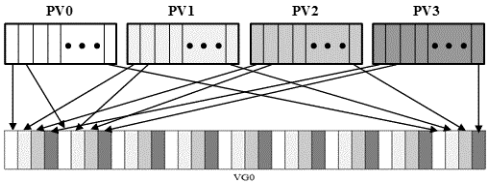

그림 4. LVM 분산매핑
Fig. 4. LVM striped mapping.

은 유연성의 장점이 있으므로, 남아있는 VG의 공간으로 백업하지 않고 LV를 확장할 수 있어 쉽게 파티션의 정보를 변경할 수 있다. 또한, 새로운 저장장치를 PV로 만들고 VG에 추가함으로써 백업이 필요하지 않은 저장장치 증설이 가능하다[12]. 그림 2는 LVM의 내부구조를 나타낸다.

LVM은 LVM1, LVM2 버전이 있으며, LVM1는 리눅스 2.4.x 시리즈에서 사용되고, LVM2는 2.6.x 이상 버전에서 사용된다. LVM1는 RHEL 4, 분산(striped) 볼륨 확장, 볼륨 미러링 등을 지원하지 않으며, PV와 LV의 최대 개수는 각각 256개이고, PV의 최대 크기는 2TB이다. 반면에 LVM2는 앞서 상술한 기능을 모두 지원하며, PV와 LV의 최대 개수는 2^32개, PV의 최대 크기는 8EB로 증가하였다. 그림 3과 그림 4는 순차, 분산 매핑을 간략하게 표현하였다.

다수의 PV를 VG에 매핑하는 방법은 순차(linear), 분산 두 가지 방법이 있다. 순차매핑은 여러 PV가 순서대로 VG에 매핑되며, 분산매핑은 여러 PV가 일정 크기씩 순서대로 반복되면서 매핑된다[13].

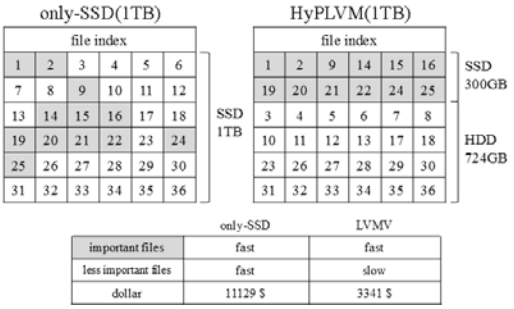

그림 5. HyPLVM와 only-SSD의 비교
Fig. 5. HyPLVM and only-SSD comparison.

Ⅲ. 설계 및 구현

3.1 HyPLVM 개요

HyPLVM은 HDD와 SSD가 통합된 하이브리드 저장공간에서 파일들의 우선순위를 판단하고, 우선순위가 높은 파일을 SSD로, 낮은 파일을 HDD로 저장하며, 자주 쓰는 파일들 한에서는 SSD로 구성된 저장공간과 버금가는 입출력 성능을 산출하면서 저장공간 구축비용을 절약하는 목적이 있다. 그림 5은 1TB의 용량을 HyPLVM과 SSD만 장착된 노드를 그림 1를 참고하여 비교하였다.

파일의 우선순위를 판별하기 위하여 HyPLVM은 모든 디렉터리 하위에 있는 파일들의 상태를 감지하며, 사용자가 파일에 접근하는 순간 우선순위를 부여, 갱신된다. 파일 우선순위는 3.3.1장에서 자세하게 설명한다.

파일 저장공간은 파티션-0, 파티션-1, 파티션-2로 이루어지며, 파티션-0은 PCIe SSD로, 파티션-1는 SATAIII SSD로, 파티션-2는 HDD로 구성되며, 우선순위가 높은 파일이 파티션-0, 1에, 그 외 파일은 파티션-2에 주로 저장된다.

HyPLVM은 파티션-0, 1의 포화를 방지하기 위해 한계용량을 초과하면 파티션-0, 1의 파일을 파티션-2로 저장하는 파일교체를 수행한다. 이때, 교체되는 파일은 앞서 상술했듯이 우선순위가 낮은 파일이 교체대상으로 선정되고, 선정된 파일을 교체함으로써 파티션-0, 1의 여유 공간을 확보한다. 파일교체는 3.3장에서 자세하게 설명한다. HyPLVM은 유저(User level), 논리(Logical level), 물리(Physical level) 레벨로 구분되며, 그림 6은 HyPLVM의 전체 흐름과 구조를 나타낸다.

3.1.1 유저 레벨

유저 레벨은 사용자가 직접 HyPLVM을 사용하는 영역으로, 리눅스 기본 쉘과 연동되어 동작한다. 사용자가

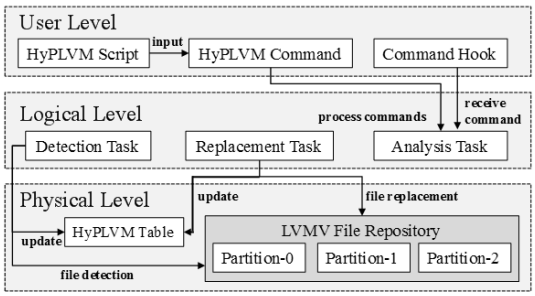

그림 6. HyPLVM의 전체구조
Fig. 6. An overall structure of HyPLVM.

파일 생성, 추가, 접근 명령 등을 수행하면, 이를 감지해 입력된 명령어를 파이프를 통해 논리 레벨로 전송한다. 유저 레벨에서 사용자는 HyPLVM 구동 및 종료 등의 관리 명령어를 실행해 HyPLVM을 제어할 수 있고, HyPLVM 스크립트로 저장공간을 구성할 수 있다.

3.1.2 물리 레벨

물리 레벨(Physical level)은 저장공간과 테이블로 구성되며, 저장공간은 LVM 기반으로 구현되어 있어 장치 추가와 파티션 정보 변경이 유연하다. 테이블에는 항목 3.1.3의 감지, 교체작업에 의해 파티션 구성과 용량, 디렉터리 구성과 용량, 파티션과 디렉터리 간의 매핑관계, 디렉터리 하위에 있는 파일의 개수와 교체정보 등이 기록된다.

3.1.3 논리 레벨

논리 레벨(Logical level)은 분석(Analysis task), 감지(Detection task), 교체(Replacement task) 3가지의 작업으로 구성되며 유저 레벨과 물리 레벨의 중간에서 상호작용 역할을 한다. 분석작업은 파이프를 통해 유저 레벨로부터 전송받는 명령어로 파일 리스트에서 사용된 파일을 찾고, 파일 우선순위를 갱신한다. 교체작업은 파티션-0, 1의 우선적 공간 활용을 위한 파일교체를 수행하며, CPU 코어 개수의 2배 만큼 생성되고 파일교체 양에 따라 다수의 교체작업을 수행한다. 교체작업들은 풀에 보관되고, 필요에 따라 대기, 실행상태로 변환해 자원을 효율적으로 사용한다.

감지작업은 유저 레벨에서 파일의 생성, 삭제가 발생하여도 분석 및 교체작업에서 바로 대처 할 수 있게 주기적으로 디렉터리를 순회해서 파일 정보, 디렉터리 정보를 갱신한다. 또한, 디렉터리 순회를 통해 파일교체가 필요한 디렉터리가 발견되면, 풀에서 하나의 교체작업을 실행상태로 변경한다.

표 1. HyPLVM 스크립트 예시
Table 1. HyPLVM script examples.

hyp_partition [partition number] [device name] hyp_directory [attribute] [partition number] [attribute] : path, size, limit, fs
hyp_partition 0 /dev/fioa1 hyp_partition 1 /dev/sda1 hyp_partition 2 /dev/sdb /dev/sdc hyp_directory “path:/mnt/part0, size:100%, limit:80%, fs:ext4” 0 hyp_directory “path:/mnt/part1_1, size:140G, limit112G, fs:ext4” 1 hyp_directory “path:/mnt/part1_2, size:ALL, limit:80%, fs:ext4” 1 hyp_directory “path:/mnt/part2, size:ALL, fs:ext4” 2

3.2 저장공간

3.2.1 HyPLVM 스크립트

저장공간을 구축하기 위해 사용자가 정해진 명령으로 저장공간의 정보를 작성하는 스크립트이다. 작성하는 정보로는 각 파티션 마다 사용될 저장장치 이름, 사용할 디렉터리의 경로, 전체용량, 한계용량, 파일시스템, 파티션과 디렉터리의 매핑 관계가 있다. 작성된 스크립트는 HyPLVM이 분석하고 저장공간 구현/취소 명령어를 생성한 뒤 명령스택에 한 노드로 저장한다. 이후 명령스택은 항목 3.2.2에서 사용되며, 표 1은 스크립트 명령어 사용법과 작성 예시이다.

표 1의 스크립트는 본 논문에서 사용한 저장장치를 사용했으며, /dev/fioa1가 Fusion io사의 PCIe SSD이고, /dev/sda1는 Samsung사의 SATAII SSD이고, /dev/sdb와 /dev/sdc는 Seagate사의 HDD이다. 4개의 저장장치로 3개의 파티션을 구성하고, 4개의 디렉터리를 구성했다. 위 예시에서 한 예로 /mnt/part1_1는 파티션-1에 매핑되는 ext4 파일시스템의 디렉터리이고 전체용량은 140GB이며, 한계용량은 112GB가 된다.

3.2.2 저장공간 구축

저장공간 구축에 사용되는 스택은 명령스택과 명령취소스택이 있다. HyPLVM은 저장공간을 구축할 때 명령스택에서 하나씩 노드를 빼내어 구현명령을 실행하고 취소명령을 명령취소스택에 넣는다. 이때, 구현명령이 실패하면 명령취소스택의 노드를 차례대로 빼내어 취소명령을 모두 실행해 원상복구를 한다.

명령스택의 구현명령을 모두 실행하면 저장공간과 테이블이 성공적으로 생성되고, 저장공간의 초기 정보를 테이블에 기록한다. HyPLVM이 동작하면서 테이블의 정보는 주기적으로 갱신되며, 저장공간의 정보가 필요한 항목 3.3을 포함한 모든 기능은 테이블에 접근해서 정보를 얻는다. 다음 표 2는 저장공간 구축기의 의사코드이다.

표 2. HyPLVM 파일 저장공간 구축기 의사코드
Table 2. HyPLVM file-repository builder pseudocode.

```
function lvmv_build
    command generate and push at command stack
    loop command stack is not empty?
        pop command at command stack
        run build command
        if success?
            push command at undo stack
        else
            loop undo stack is not empty?
                pop command at undo stack
                run undo command
            end loop
            goto error
        end if
    end loop
    return success
    error: return failure
end function
```

표 3. 명령 훅 스크립트
Table 3. Command hook script.

```
LAST_COMMAND=`tail -1 /etc/lvmv/cmd_dump`
CURRENT_COMMAND=`history 1`

if [ "$LAST_COMMAND" != "$CURRENT_COMMAND" ]
then
    echo "$CURRENT_COMMAND" >> /etc/lvmv/cmd_dump
fi
```

3.3. 파일교체

3.3.1 파일 우선순위 분석

이 항목에서는 논리 레벨의 분석작업을 상세하게 기술한다. 파일 우선순위는 리스트로 구현되었으며 노드가 헤드에 가까울수록 우선순위가 낮고, 노드가 헤드로부터 멀어질수록 우선순위가 높게 구분한다. 파일 우선순위 리스트는 LRU 알고리즘 방식으로 동작하며, 사용자가 접근한 파일은 파일 우선순위 리스트에서 해당하는 노드를 찾고 마지막으로 이동해 우선순위가 높아진다.

HyPLVM은 파일 우선순위 리스트의 수많은 노드 대상으로 검색하기 위해서 해시테이블을 사용한다. 해시테이블은 100,007의 키를 갖는 배열로 구현했으며, 키는 파일의 절대경로가 되고, 값은 파일 우선순위 리스트 노드와 파일 리스트 노드가 된다. 리스트 단순 검색이 아닌 해시테이블 검색을 통해 원하는 노드에 직접 접근을 할 수 있으며, 이때 알고리즘 복잡도는 상수시간으로 O(1)에 가깝다. 파일 관련 리스트는 총 3개가 존재하며 항목 3.3.2에서 자세하게 다룬다.

사용자가 파일에 접근하는 순간과 어느 파일에 접근했는지 파악하기 위하여, HyPLVM은 명령 훅(Command Hook) 스크립트를 리눅스 기본 쉘과 연동한다. 명령 훅 스크립트는 사용자 명령어를 가로챌 수 있으며, 가로챈 명령어를 파이프 통신으로 논리 레벨의 분석작업으로 전송한다. 다음 표 3는 명령 훅 스크립트이다.

3.3.2 저장공간 감지

이 항목에서는 논리 레벨의 감지작업을 상세하게 기술한다. HyPLVM에는 파일 관련 리스트는 순서가 중요한 파일 우선순위 리스트와 순서가 중요하지 않은 2개의 파일 리스트가 있다. HyPLVM은 주기적으로 디렉터리를 순회하면서 2개의 파일 리스트로 파일과 디렉터리의 상태를 파악하고, 테이블에 정보를 갱신한다.

첫 디렉터리를 순회하면서 모든 파일을 우선순위 리스트와 하나의 파일 리스트(이하 리스트A)에 추가하고, 파일의 절대경로를 키로, 2개의 노드를 값으로 하여 해시테이블에 입력한다. 첫 순회가 끝나면 해시테이블과 파일 우선순위 리스트, 리스트A는 데이터가 있고 다른 파일 리스트(이하 리스트B)는 비어있다. 일정 주기가 지나고 다시금 순회할 때 아래 방법으로 파일들의 상태를 파악한다.

디렉터리를 순회할 때 모든 파일을 해시테이블에 검색하여 리스트A 노드를 찾아 노드를 리스트B로 이동한다. 만약 노드를 찾지 못하면, 새로운 파일이라 파악하고 우선순위 리스트와 리스트B에 추가하고 해시테이블에 입력한다. 이때, 우선순위 리스트의 마지막에 추가되어 처음 생성된 파일의 우선순위가 가장 높다.

파일이 삭제되면 디렉터리 순회할 때 없기 때문에 이 방법으로 순회를 종료하면 삭제된 파일은 리스트A에 남아있게 된다. 리스트A에 남아있는 노드들은 파일이 삭제되었음을 파악하고 우선순위 리스트와 해시테이블에서 삭제하고 리스트A를 비운다. 그리고 다음 순환에는 리스트A와 리스트B를 바꾸어 디렉터리를 순회한다. 한 번의 순회를 종료하면 디렉터리의 현재용량을 한계용량과 비교해 교체작업을 실행 상태로 변경한다.

3.3.3 파일교체 알고리즘

이 항목에서는 논리 레벨의 교체작업과 파일교체 알고리즘을 상세하게 기술한다. 파일교체는 파티션-0, 1 디렉터리 내에서 항목 3.2.1의 분석으로 선정된 우선순위가 낮은 파일들이 파티션-2로 저장되고, 파티션-0, 1 디렉터리를 일정 용량으로 유지해 자주 사용하는 파일이 저장될 용량을 확보하여 고비용인 파티션-0, 1의 공간 활용도를 증대시킨다. 파일교체는 파티션-0, 1의 용량이 표 1의 스크립트에서 설정한 한계용량을 초과하면 실행되며, 표 1 스크립트의 경우 /mnt/part0는 용량이 80%를 초과하면 파일교체가 실행된다.

파일교체는 파티션-0, 1 디렉터리의 파일 우선순위 리스트에서 교체될 파일들을 디렉터리 용량이 한계용량

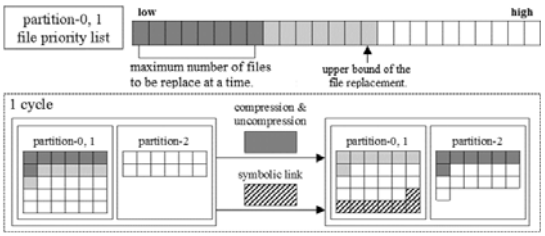

그림 7. 파일교체 알고리즘
Fig. 7. File replacement algorithm.

을 초과하기 전까지 혹은, 교체될 파일들의 용량 합계가 4GB를 넘을 때까지 가져와 한 파일로 묶는다. 묶어진 한 파일을 압축하고 파티션-2 디렉터리를 지점으로 파일명이 중복되지 않게 압축 해제한다. 이 방법을 압축복사라 하며, 여기까지 진행되었다면 파티션-0, 1과 파티션-2 디렉터리에 같은 파일이 존재하게 된다.

사본 위치인 파티션-2 디렉터리의 파일들을 원본 위치인 파티션-0, 1 디렉터리 내 파일에 링크로 덮어씌워 원본 위치로 접근할 수 있게 한다. 여기까지가 한 번의 파일교체가 하는 동작이며, 파일교체는 파티션-0, 1 디렉터리의 용량이 한계용량을 초과하지 않을 때까지 반복적으로 실행된다. 다음 그림 7은 파일교체 알고리즘을 도식화한 것이다.

Ⅳ. 성능 결과

4.1 성능 측정 개요

HyPLVM은 구 버전과 개선 버전이 있다. 구 버전은 단순 리스트 검색으로 파일을 찾고, 개선 버전은 해시 테이블 검색으로 파일을 찾는다. 파일교체도 구 버전은 단순복사를 사용하지만 개선 버전은 압축복사를 사용한다. 때문에 개선 버전이 구 버전보다 비교적 높은 성능을 발휘하며, 성능 측정은 구 버전과 개선 버전의 성능 비교이다.

성능 측정은 파일교체 중 디렉터리 현황, 파일교체의 소요시간, 초당 교체되는 파일 개수, 파일교체의 속도를 측정하며, 선정된 파일크기는 4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB이다. 벤치마크는 postmark를 사용했으며, 파일을 생성 후 지우지 않고, 읽기 성능을 측정하는 구간에서 파일의 접근빈도를 수집하도록 수정하였다. 또한, postmark가 파티션-0, 1에 동시에 파일작업 할 수 있도록 수정해 HyPLVM에 최적화되었고, 성능 측정에 대한 로직은 수정하지 않았다.

성능측정에 사용된 환경은 표 4와 같으며, 저장공간

표 4. 성능측정 환경
Table 4. Performance measurement environment.

CPU	AMD FX(tm)-8350 Eight-Core Processor
Memory	Samsung DDR 16 GB (8GB 2Rx8 PC3-12800U-11-13-B1 2E)
Disk	Funsion IO SLC ioDrive 80GB 1E
	Samsung SSD 850 Pro 256GB 1E
	Seagate Desktop HDD ST1000DM003 1TB 2E
OS	Ubuntu 3.13.0-24-generic
Tools	GCC 4.8.2
	LVM2 2.02.106(2)-git
	postmark 1.51

표 5. 파일 저장공간 파티션 설정
Table 5. File-repository partition configuration.

	device	directory count	capacity	limit
partition-0	fusion io SSD 1E	1	2.5GB	1.0GB
partition-1	intel SSD 1E	1	3.5GB	1.4GB
partition-2	intel HDD 2E	1	6GB	-

표 6. postmark에서 생성되는 파일 전체 개수와 용량
Table 6. Total file numbers and sizes in postmark.

file size	file count	create capacity
4KB	1049242	4.00GB
8KB	524200	3.99GB
16KB	262060	3.99GB
32KB	130980	3.99GB
64KB	65120	3.97GB
128KB	33200	4.05GB
256KB	15940	3.89GB
512KB	8220	4.01GB

은 표 5와 같이 구축하였다. postmark가 파티션-0, 1에 파일을 생성하는 개수와 그 용량은 표 6과 같으며 각 계층에 생성될 파일의 용량은 2GB로 postmark가 생성하는 파일의 전체 용량이 표 5에서 설정한 한계용량을 초과하기 때문에 교체가 발생하게 된다.

4.2 파일교체 현황

파일교체를 통해 저장공간을 일정 용량으로 유지하는지 측정하기 위해 각 파일 크기마다 파일교체 현황을 기록하였다. 파일크기는 대표적으로 가장 작은 4KB와 가장 큰 512KB만 그래프로 표현하였고, 나머지 파일크

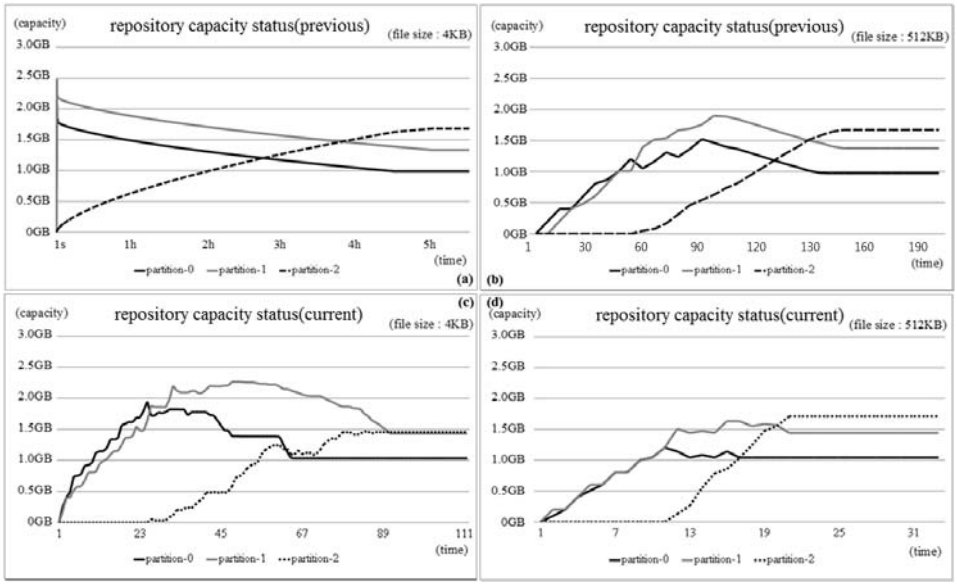

그림 8. 파일교체 성능
Fig. 8. File replacement measurement.

표 7. 파티션-0의 각 파일크기 별 파일교체 결과
Table 7. File replacement results on the partition-0.

	4KB	8KB	16KB	32KB
write time	23.1	18.75	14.72	9.39
previous	15400	17925	6505	1410
current	68	41	39	23
	64KB	128KB	256KB	512KB
write time	9.27	9.49	12.84	15.36
previous	555	295	160	130
current	20	20	19	19

표 8. 파티션-1의 각 파일크기 별 파일교체 결과
Table 8. File replacement results on the partition-1.

	4KB	8KB	16KB	32KB
write time	23.1	18.75	14.72	9.39
previous	17160	18345	7235	1375
current	91	65	52	28
	64KB	128KB	256KB	512KB
write time	9.27	9.49	12.84	15.36
previous	600	305	165	135
current	23	22	21	21

기는 그래프가 많아 논문에 모두 표현하기가 어려워 간단한 도표로 표현하였다.

그림 8과 같이 파티션-0은 1.0GB, 파티션-1은 1.4GB를 초과하지 않게 각 파티션의 용량이 파티션-2로 파일교체를 하면서 유지된다. 개선 버전(c), (d)은 구 버전(a), (b)보다 완료 시간이 512KB의 경우 110초가량 빠르고, 4KB의 경우 비교할 수 없을 정도로 빠르다. 두 버전 모두 4KB(a), (c)는 파일개수가 많아 512KB(b), (d)보다 비교적 낮은 성능을 보인다.

다음 표 7과 표 8은 postmark의 파일쓰기가 완료된 시간, 구 버전과 개선 버전의 파일교체가 완료된 시간을 여러 파일 크기로 테스트한 결과이다. 개선 버전이 구 버전보다 모든 파일크기의 교체 완료시간이 최소 110초가량 빠르다.

4.3 파일교체 속도

지속해서 저장공간에 파일이 저장될 때 파일교체의 속도가 느리다면 언젠가는 저장공간에 더 이상 파일을 저장할 수가 없는 문제가 발생한다. 이 문제가 발생하지 않게 파일교체 속도를 최대한 파일쓰기 속도만큼 성능이 산출할 수 있게 설계하였다.

이 항목에서는 파일교체 속도가 얼마나 빠른지 측정하고, 구 버전과 개선 버전을 서로 비교한다. 항목 4.2와 같은 환경에서 테스트를 진행하였고, 각 파일 크기마다 파일교체 속도와 단위 시간당 교체되는 파일의 개수 중점으로 측정하였다.

그림 9는 파일교체 속도 측정에 대한 결과를 구 버전과 개선 버전을 비교하고 그래프로 표현하였으며, (a), (b)는 단위시간당 교체되는 파일 개수의 비교이고, (c), (d)는 교체속도의 비교이다. 단순 복사를 통해 파일교체하는 구 버전과 비교하여 개선 버전은 최소 18mb/sec,

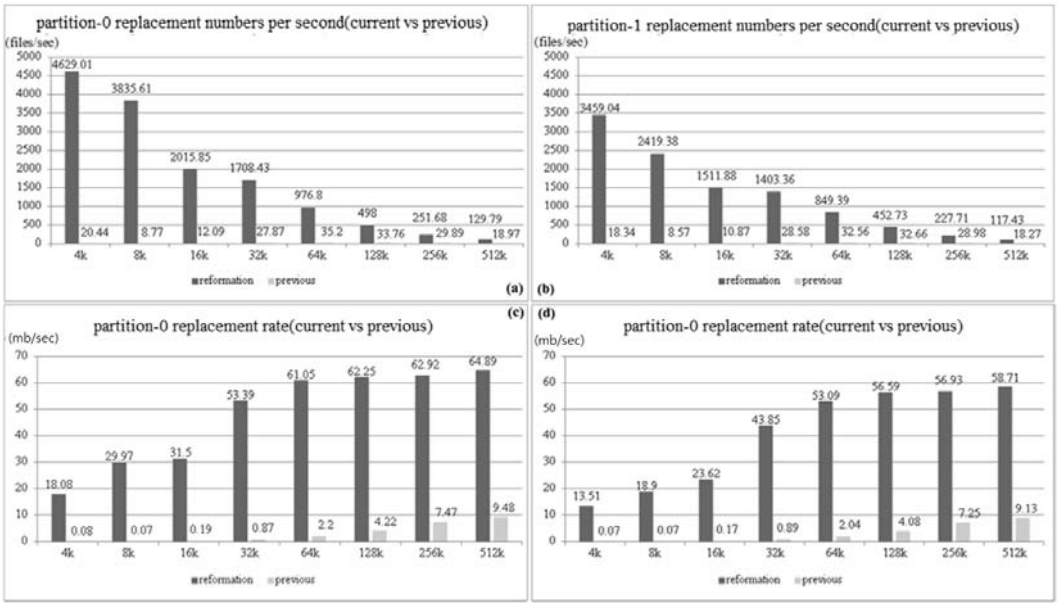

그림 9. 파일교체 속도 비교
Fig. 9. File replacement speed measurement.

최대 50mb/sec 만큼 빠른속도를 보였고, 압축복사를 사용하는 개선 버전은 구 버전과 비교하여 단위시간당 파일을 최소 110개, 최대 4,600개 더 많이 교체하였다.

4.4 파일 우선순위에 따른 파일교체

파일교체는 파일접근빈도를 분석해 파일 우선순위를 측정하고, 이 기반으로 우선순위가 낮은 파일을 먼저 교체되어야 한다. 이 항목에서는 파일교체된 파일이 접근빈도가 낮은 파일임을 증명하고, 접근빈도에 따라 파티션-0, 1, 2에 올바르게 저장되는지 증명한다.

성능측정 환경은 항목 4.2와 같으며 측정에 사용되는 벤치마크 postmark는 파일 생성작업을 제외하고 파일 읽기, 쓰기 작업을 랜덤으로 동작하도록 수정하였다. 이 랜덤 작업에서 랜덤적 파일접근이 발생하며, postmark를 여러 번 수행 후 파일마다 최근 접근한 시간을 분석하였다.

그림 10은 100만 개 파일을 poatmark에서 약 1,000만 번의 접근하고, 각 파티션 마다 최근에 쓰인 12만 개 파일을 모아서 최근 접근 시간 기준 오름차순으로 정렬하고 그래프로 표현 한 것이다. 파티션-0의 파일들은 최대 1초 이내에 사용한 최근에 접근된 파일들이 주로 저장되어있고, 파티션-2는 최대 12초 혹은 그 이상 동안 사용이 없었던 파일이 주로 저장되어있다.

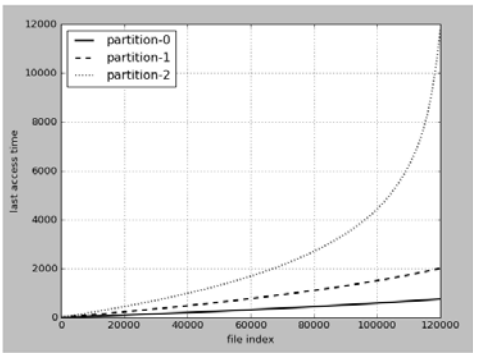

그림 10. 파일 우선순위와 접근빈도
Fig. 10. File priority and access frequency.

V. 결 론

본 논문에서는 SSD와 HDD가 통합된 시스템에서 파일의 접근빈도를 분석해 주로 쓰는 파일과 쓰지 않은 파일을 구분하고, 접근빈도가 높은 파일만 SSD에 저장되도록 관리하는 소프트웨어 HyPLVM을 제안하였다. 사용자는 HyPLVM을 통해서 SSD와 HDD가 통합한 적은비용의 시스템으로 HDD 용량만큼의 파일을 저장하고, SSD만큼 입출력 성능을 산출 할 수 있게 되었다.

하지만 지금 HyPLVM의 파일 우선순위는 LRU 알고리즘을 적용하고 있다. LRU는 최근에 사용한 파일에 대해서만 우선순위를 부여하는 문제가 있다. 예를 들면, (1)접근빈도가 많은 파일이 장시간 사용되지 않아 우선순위가 낮아진 파일과 (2)접근빈도도 낮고 장시간 사용되지 않아 우선순위가 낮아진 파일이 있을 때, LRU 알

고리즘은 (1)와 (2)파일 모두 같은 취급을 한다. 이에 알고리즘을 개선하여 파일 우선순위를 새로 적용하고 사용자가 사용하는 파일을 융통성 있게 교체하는 연구가 필요하다.

REFERENCES

[1] Jeong-Su Park, Yu-Mi Bae, Sung-Jae Jung, “Technical analysis of Cloud Storage for Cloud Computing”, KIICE, Vol. 17, no. 5, pp. 1129-1137, May 2013.

[2] Yaffs. Available on July 2011 from “http://www.yaffs.net/”

[3] D. Woodhouse. Jffs: The journalling flash file system. In The Ottawa Linux Symposium, RedHat Inc, 2001.

[4] TAILWINDSTORAGE. Extreme 3804, “http://tailwindstorage.com/products/”

[5] Fusion-IO, “iodrive octal data sheet, http://www.fusionio.com/data-sheets/iodrive-octal-data-sheet/”

[6] D. G. Andersen and S. Swanson, “Rethinking flash in the data center”, IEEE Micro, Vol. 30, no. 4, pp.52-54, Jul. 2010.

[7] F. Douglis, R. Cáceres, F. Kaashoek, K. Li, B. Marsh, J. Tauber, “Storage alternatives for mobile computers”, Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation, 1994.

[8] Jin-Yong Ha, Jin-Soo Kim, “Hext4: A Filesystem for SSD-HDD Hybrid Storage”, KIISE, pp. 1403-1405, Jun, 2014.

[9] Sanghyun Yoo, Kyung Tae Kim, Hee Yong Youn, “Block Replacement Scheme based on Reuse Interval for Hybrid SSD System”, KSII, Vol, 16, no. 5, pp. 19-27, Oct, 2015.

[10] Amplicon, “Benefits of SSD vs. HDD, https://www.amplicon.com/docs/white-papers/SSD-vs-HDD-white-paper.pdf”

[11] AJ Lewis, “LVM HOWTO, http://www.sistina.com”

[12] David Teigland, “Volume Manager in Linux”, Sistina Software, Inc., 2001.

[13] redhat, “The Linux Logical Volume Manager, http://www.redhat.com/magazine/009jul05/features/lvm2/”

[14] Siwoo Byun, “A Hetero-Mirroring Scheme to Improve I/O Performance of High-Speed Hybrid Storage”, KAIS, Vol. 11, No. 12, pp. 4997-5006, 2010.

[15] Sanghyun Yoo, Hee Yong Youn, “Cache replacement algorithm based on time and hit ratio in NAND flash memory”, KSII, pp. 231-232, Oct 2014.

저 자 소 개

최 훈 하(정회원)

2012년 국가평생교육진흥원 컴퓨터공학과 학사 졸업.

2014년～현재 세종대학교 컴퓨터공학과 석사과정.

<주관심분야 : 리눅스, 디바이스 드라이버, 가상화>

김 현 지(정회원)

2013년 세종대학교 컴퓨터공학과 학사 졸업.

2014년～현재 세종대학교 컴퓨터공학과 석사과정.

<주관심분야 : 리눅스, 디바이스 드라이버, 가상화>

노 재 춘(정회원)－교신저자

세종대학교 컴퓨터공학과 교수

1985년 이화여자대학교 전자계산학과 학사 졸업.

1999년 Syracuse University 전산학과 박사 졸업.

<주관심분야 : 클라우드컴퓨팅, 파일시스템, 데스크탑 가상화>