https://doi.org/10.5392/JKCA.2019.19.05.417

가상파일시스템에서 디렉토리 및 파일 변경 추적에 기반한 온라인 스냅샷 방법

Online Snapshot Method based on Directory and File Change Tracking for Virtual File System

김진수*, 송석일*, 신재룡**
한국교통대학교 컴퓨터공학전공*, 광주보건대학교 보건행정과**

**Jinsu Kim(biosvos@gmail.com)*, Seokil Song(sisong@ut.ac.kr)*,
Jae Ryong Shin(sjr@ghu.ac.kr)****

요약

스토리지 스냅샷(snapshot) 기술은 특정 시점의 데이터를 보전하고, 필요할 때 원하는 시점의 데이터를 복구하여 접근하기 위한 기술로 스토리지 보호 응용 개발에 필수적인 기술이다. 이 논문에서는 기존의 스냅샷 기술이 스토리지 하드웨어 벤더에 종속되거나 파일시스템이나 가상 블록장치에 종속되는 문제를 해결하고 온라인으로 스냅샷을 생성하기 위한 새로운 스냅샷 방법을 제안한다. 이 논문에서는 제안하는 스냅샷 기술은 가상파일시스템의 변경연산에 대한 로그를 추출하는 방식을 이용하여 파일시스템, 가상 블록장치, 하드웨어에 독립적으로 동작하는 스냅샷 기술을 설계하고 개발한다. 또한, 개발하는 스냅샷 기술은 디렉토리 및 파일 단위의 스냅샷을 스토리지 서비스 중단 없이 생성하고 관리 한다. 마지막으로 실험을 통해서 제안하는 스냅샷 기술의 스냅샷 생성시간 및 스냅샷으로 인한 성능 저하를 측정한다.

■ 중심어 : | 스토리지 | 스냅샷 | 가상파일시스템 | 복구 |

Abstract

Storage snapshot technology allows to preserve data at a specific point in time, and recover and access data at a desired point in time. It is an essential technology for storage protection application. Existing snapshot methods have some problems in that they dependent on storage hardware vendor, file system or virtual block device. In this paper, we propose a new snapshot method for solving the problems and creating snapshots on-line. The proposed snapshot method uses a method of extracting the log records of update operations at the virtual file system layer to enable the snapshot method to operate independently on file systems, virtual block devices, and storage hardwares. In addition, the proposed snapshot mehod creates and manages snapshots for directories and files without interruption to the storage service. Finally, through experiments we measure the snapshot creation time and the performance degradation caused by the snapshot.

■ keyword : | Storage | Snapshot | Virtual File System | Recovery |

* 이 논문은 2014년도 광주보건대학교 교내연구비의 지원을 받아 수행된 연구임(No. 3014020)

접수일자 : 2019년 04월 23일
수정일자 : 2019년 05월 03일
심사완료일 : 2019년 05월 03일
교신저자 : 신재룡, e-mail : sjr@ghu.ac.kr

I. 서 론

스토리지 스냅샷(snapshot) 기술은 특정 시점의 데이터를 보전하고, 필요할 때 원하는 시점의 데이터를 복구하여 접근하기 위한 기술이다. 기존 스냅샷 기술들은 리눅스 LVM(Logical Volume Manager)[1] 또는 DM(Device Mapper)[2] 등의 가상 블록 장치를 기반으로 동작하는 스냅샷 기술과, 디스크 어레이 등의 하드웨어에 통합된 기술로 구분할 수 있다.

블록 수준의 소프트웨어 스냅샷은 LVM이나 DM 등과 같은 가상 블록장치가 없으면 수행이 불가능하다. 스토리지 하드웨어에 통합된 스냅샷 기술은 스토리지 하드웨어에 종속되는 문제가 발생한다. 스토리지 하드웨어 업체는 고가의 선택 사양으로 스냅샷 기능을 제공하며 스냅샷 기능을 이용할 수 있는 전용 스냅샷 API를 같이 제공한다. 현재 스냅샷 생성 및 관리를 위한 표준화된 API는 없으며 스토리지 하드웨어 업체마다 모두 다른 API를 제공하고 있다. 따라서, 다양하게 사용되는 디스크 어레이 업체에서 제공하는 스냅샷 기능을 모두 지원하는 것은 부가적으로 많은 노력과 비용이 발생할 수밖에 없다.

이런 문제를 해결하기 위해서는 특정 환경 또는 스토리지 하드웨어에 독립적이며, 응용 프로그램을 고려하여 데이터의 정합성을 보장하는 스냅샷을 생성 및 관리하는 기술 개발이 필요하다. 이 논문에서는 파일시스템의 변경연산에 대한 로그를 추출하는 방식을 이용하여 가상 블록장치나 하드웨어에 독립적으로 수행하는 스냅샷 기술을 설계하고 개발한다. 또한, 개발하는 스냅샷 기술은 디렉토리 및 파일 단위의 스냅샷을 스토리지 서비스 중단 없이 생성하고 관리 한다.

이 논문의 구성은 다음과 같다. 2장에서 기존에 제안된 스냅샷 기술에 대해서 설명한다. 3장에서는 이 논문에서 제안하는 VFS(Virtual File System) 기반의 스냅샷 기술에 대해서 설명한다. 4장에서는 제안하는 스냅샷 기술에 대한 성능평가 결과를 기술하고 5장에서 결론을 맺는다.

II. 관련연구

스토리지 스냅샷 기술은 특정 시점의 데이터를 보전하고, 필요할 때 원하는 시점의 데이터를 복구하여 사용하기 위한 기술로 스토리지의 데이터 정합성을 유지시키며 데이터를 백업하는데 있어 필수 기술이다. 스냅샷은 일반적으로 데이터 미러링(Mirroring) 또는 CoW(Copy on Write)[3-6] 와 RoW(Redirect on Write)[7][8] 형태로 구현된다.

데이터 미러링은 일반적으로 하드웨어적으로 구현되며 많은 스토리지 용량이 요구된다. 이에 비해 CoW와 RoW 기반의 스냅샷은 변경되는 데이터만 유지하여 스냅샷 이미지 보전이 가능하기 때문에 상대적으로 작은 용량이 소요된다. CoW는 스토리지에 대한 스냅샷이 생성된 이후 원본 블록 또는 레코드가 변경될 때 원본 이미지를 다른 영역에 먼저 복사하고 원본을 수정하는 방법이다. 반면 RoW는 블록이나 레코드에 대한 변경을 별도의 공간에 기록하는 방법이다.

스냅샷은 리눅스 LVM(Logical Volume Manager)이나 DM(Device Mapper) 등 블록 계층에서 구현된 기술[3][4]들과, 디스크 어레이(Array)에 통합된 기술[5], 특정 파일 시스템에서 구현된 기술[6-8] 로 구분해 볼 수 있다. 블록 수준에서 구현된 기술은 소프트웨어적으로 구현된 것으로 리눅스 LVM 또는 DM 등을 사용하는 환경에서만 스냅샷 사용이 가능하다. 디스크 어레이 하드웨어에 통합되는 경우에는 하드웨어에 종속되게 되며, 특정 파일시스템에서 구현된 경우에는 파일시스템에 종속적일 수밖에 없다.

스토리지 업계에서는 NetApp, EMC VNX, VERITAS, Hitachi 등과 같은 스토리지 하드웨어 업체에서 별도의 선택 사항으로 스냅샷 기능을 제공하고 있다. 그러나, 스토리지 하드웨어 업체에서 제공하는 스냅샷 기능을 별도로 사용할 경우 응용 프로그램을 고려하고 대처하지 못하게 되면 데이터의 정합성을 보장 하지 못할 수 있다. 데이터 정합성을 유지하는 스냅샷을 생성하기 위해서는 스토리지 하드웨어 업체에서 제공하는 스냅샷 API를 이용하여 응용 프로그램을 고려하는 백업 응용을 개발해야 한다. 그러나, 현재 표준화된 스냅샷 API는 없으며 업체마다 모두 다른 API를 제공하고 있으므로 현장에서 사용되는 다양한 디스크 어레이 업체에서 제공하는 스냅샷 기능을 모두 지원하는 것은 많은 노력과 비용이 발생하는 문제가 있다.

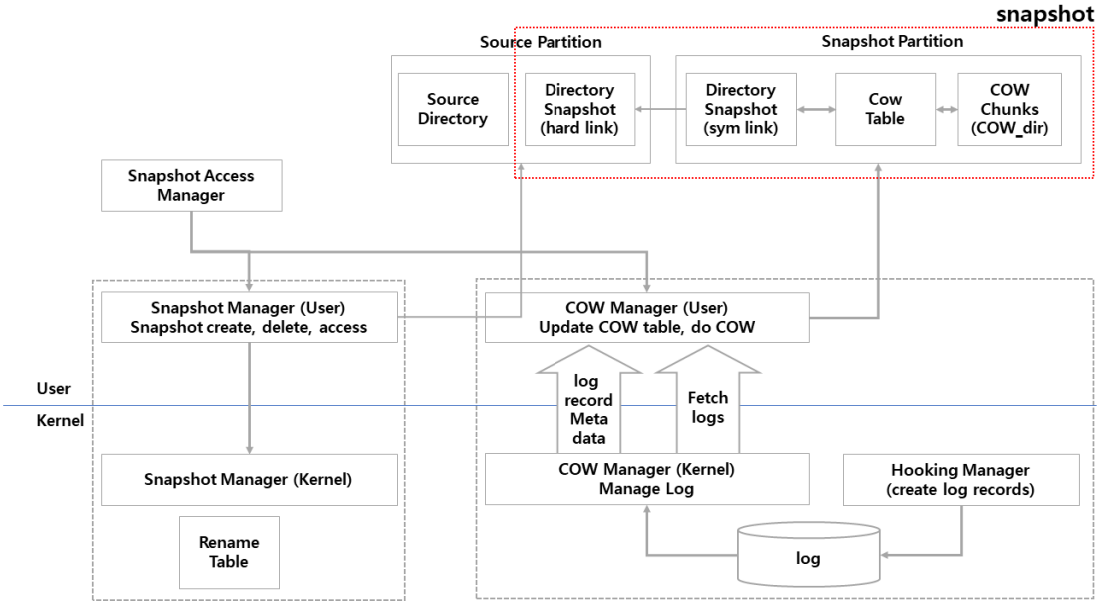


그림 1. 개발하는 스냅샷 방법의 구조

III. 제안하는 VFS 기반 스냅샷 방법

1. 제안하는 스냅샷 방법의 개요

앞서 기술한 바와 같이 이 논문에서 개발하는 스냅샷 방법은 VFS 계층에서 동작하는 소프트웨어 스냅샷이다. VFS 계층에서 스냅샷 대상 디렉토리나 파일에 대한 변경연산을 추적하여 로그를 생성하고 이를 이용하여 CoW 방식으로 스냅샷을 유지한다. 또한, 제안하는 스냅샷 방법은 온라인 스냅샷 생성을 지원한다. 즉, 스토리지 서비스를 중지하지 않고 스냅샷 생성하는 것이 가능하다. 제안하는 스냅샷 방법은 온라인 스냅샷 생성을 지원하기 위해서 리눅스 파일시스템의 하드링크 를 이용한 디렉토리 스냅샷 기법을 제안한다.

제안하는 스냅샷 방법은 스냅샷 생성 단계와 스냅샷을 유지하는 CoW 단계로 구성되며 [그림 1]과 같은 구조를 갖는다. Snapshot Manager는 커널과 사용자 영역에 걸쳐 있으며 스냅샷을 생성, 삭제, 접근하고 Rename Table을 유지하여 스냅샷 도중에 발생하는 rename 연산에 대응한다.

Snapshot Manager의 스냅샷 생성은 스냅샷 대상 디렉토리인 Source Directory가 포함된 Source Partition과 Snapshot 유지를 위한 Snapshot Partition를 포함한다. Source Partition은 Source Directory에 대한 Directory Snapshot도 포함한다. Snapshot Partition에는 Source Partition의 디렉토리 스냅샷에 대한 Symlink를 포함하고 있으며 Cow Manager가 생성하는 로그레코드를 처리하는 Cow Table과 Cow chunks를 포함하고 있다.

Snashot Manager에 의해서 스냅샷이 생성되면 COW Manager가 Source Directory의 변경연산을 추적하여 로그를 만들고 이를 이용하여 COW를 수행한다. Snapshot Access Manager는 생성한 스냅샷을 접근하는 기능을 제공하며 Snapshot Manager와 Cow Manager를 통해 접근을 수행한다.

2. 제안하는 온라인 스냅샷 생성 방법

[표 1]은 스냅샷 생성 방법에 대한 의사코드이다. 스냅샷 생성은 create_snapshot()을 호출하면서 시작된다. create_snapshot은 입력으로 source_path(source_directory의 full path), snapshot_path(snapshot directory 의 full path), mount_path(mount directory 의 full path), cow_path(cow directory 의 full path), create_time을 받는다.

표 1. 스냅샷 생성 알고리즘

```
create_snapshot(source_path, snapshot_path, mount_path, cow_path, create_time)
//create_time : create_snapshot을 호출할때의 시간, source_path와 결합하여 snapshot name이 됨.
{
   add_log(log, START_COPY_DIR_HARDLINK, source_path, snapshot_path, mount_path, cow_path, ss_datetime);

    source_path의 디렉토리 및 파일을 snapshot_path에 복사(파일은 hard link로);
      snapshot_path 를 mount_path 에 mount;

    log(END_COPY_DIR_HARDLINK, source_path, snapshot_path, mount_path, cow_path, create_time);
    redo_log(log, source_path, snapshot_path, mount_path cow_path, create_time);
    log(END_CREATE_SNAPSHOT, source_path, snapshot_path, mount_path, cow_path, create_time);
}

redo_log(LOG, source_path, snapshot_path, cow_path, create_time)
{
   log=get_log(START_COPY_DIR_HARDLINK, source_path, create_time);

   while(1)
   {
        log = get_next_log(log, source_path, create_time);

        switch(log->type){

        case CREAT, LINK, SYMLINK, MKNOD : //파일 생성, 링크 생성
             adjusted_path = snapshot_path + log->path[len(source_path):];
             if(is_file_exist(adjusted_path) == FILE_NOT_EXIST)
                  link(log->path, adjusted_path);

        case MKDIR : //디렉토리 생성
             adjusted_path = snapshot_path + log->path[len(source_path):];
             if(is_file_exist(adjusted_path) == FILE_NOT_EXIST)
                  mkdir(log->path, log->mode);

        case UNLINK : //파일 삭제
             adjusted_path = snapshot_path + log->path[len(source_path):];
             if(is_file_exist(adjusted_path) == FILE_EXIST)
                  unlink(adjusted_path);

        case RMDIR : //디렉토리 삭제
             adjusted_path = snapshot_path + log->path[len(source_path):];
             if(is_file_exist(adjusted_path) == FILE_EXIST)
                  rmdir(adjusted_path);

        case RENAME : //파일 및 디렉토리 명 변경 및 이동
             adjusted_path = snapshot_path + log->old_path[len(source_path):];
             if(is_file_exist(adjusted_path) == FILE_EXIST)
                  unlink(adjusted_path);

             adjusted_path = snapshot_path + log->new_path[len(source_path):];
             if(is_file_exist(adjusted_path) == FILE_NOT_EXIST)
                  link(adjusted_path);

        case END_COPY_DIR_HARDLINK: // multiple snapshot 이 허용되고 create_snapshot()이 서로
                                    // overlap 될 때는 create_time을 같이 확인해야 함
             if(log->source_path == source_path && log->create_time == create_time)
                  break;
        }
   }
}
```

source_path는 스냅샷을 생성하려는 대상 디렉토리 경로이며, snapshot_path는 생성된 스냅샷이 저장되는 경로, cow_path는 스냅샷 대상 디렉토리의 파일이 변경될 때 변경되기 전의 데이터를 청크(chunk) 단위로 저장하는 경로이다. create_time은 스냅샷 생성이 시작되는 시간이다. 각 스냅샷은 source_path와 create_time이 결합되어 유일하게 구분된다.

스냅샷 생성이 시작되면 제일 먼저 START_COPY_DIR_HARDLINK 로그 레코드를 로그에 추가한다. 이 로그 레코드는 source_path, snapshot_path, mount_path, cow_path, create_time를 같이 저장한다. 이후 source_path의 디렉토리 및 파일을 snapshot_path에 복사한다.

파일은 하드 링크 형태로 복사한다. 복사가 끝나면 snapshot_path를 mount_path에 마운트 하여 CoW를 위한 파티션에서 snapshot_path의 하드 링크를 통해 source_path의 파일들을 접근할 수 있도록 한다. 마지막으로, END_COPY_DIR_HARDLINK 로그 레코드를 추가하고 redo_log()를 호출한다.

redo_log()는 START_COPY_DIR_HARDLINK 로그 다음 로그 레코드부터 처리를 시작한다. 로그의 내용을 분석하여 source_path의 디렉토리 구조가 변경(creat, mkdir, mknod, link, symlink, rename, mkdir, unlink, rmdir 등)된 로그일 경우 snapshot_path에 이를 반영하여 source_path와 snapshot_path를 일치시킨다.

redo_log()가 끝나고 나면 END_CREATE_SNAPSHOT 로그를 기록하고 스냅샷 생성을 끝낸다. CoW는 END_CREATE_SNAPSHOT 로그가 기록된 이후의 source_path의 이미지를 보존하기 위해 수행되며 다음에서 자세히 기술한다.

[그림 2]는 redo_log()의 필요성과 동작 과정을 예로 보여 준다. [그림 2]의 오른쪽은 source_path(S) 의 디렉토리 및 파일을 1, 2 번 까지의 수행을 통해 snapshot_path(SS) 에 복사하고 "S/A/C/" 파일 및 “/S/A/D” 파일에 대한 하드링크를 생성하기 전에 "S/A/NF" 파일이 생성되는 상황을 보여준다.

source_path의 디렉토리 및 파일들은 snapshot_path에 깊이 우선 순서로 복사하며 한번 복사한 영역에 대해서는 재수행을 하지 않으므로 스냅샷 생성이 완료되더라도 MF 파일은 snapshot_path에 복사 되지 않는다.

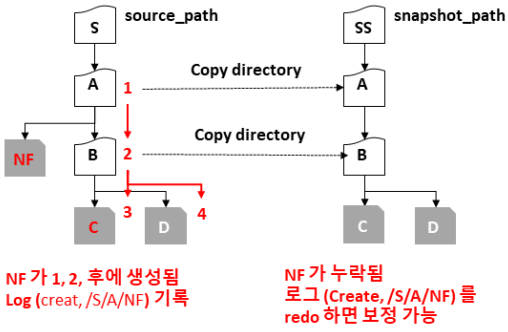


그림 2. redo_log() 의 필요성을 보여주는 사례

[그림 2]의 왼쪽은 source_path로부터 복사가 완료된 snapshot_path 이며 NF 파일은 미처 하드링크가 생성되지 않은 상태를 보여준다. 스냅샷 생성이 완료된 시점에서 source_path와 snapshot_path는 완전히 일치해야 하므로 이를 보정해야 한다. 복사하는 동안 source_path에 가해지는 모든 종류의 변경연산은 시스템 호출 가로채기 기법에 의해 로그에 기록되므로 creat("/S/A/NF") 는 로그에 남아 있을 것이다.

제안하는 스냅샷 방법에서는 이 로그레코드를 snapshot_path에 대해 재수행(redo) 하여 보정을 수행한다. 이와 같이 어떤 로그레코드에 대해서 보정을 수행해야 하는지를 판별하기 위해서 스냅샷 생성 및 완료 시간, 로그레코드 생성시간을 서로 비교한다.

만일 creat("/S/A/NF") 로그 레코드가 수행된 시간이 스냅샷 생성을 시작하기 전 이거나 생성이 끝난 후라면 보정을 수행할 필요가 없다. 앞의 creat로그를 redo하기 위해서는 creat를 다시 수행해야 하지만, 앞서 기술한 바와 같이 파일의 경우에는 snapshot_path에 복사하는 것이 아니라 하드링크를 생성하므로 NF 파일에 대한 하드링크를 생성하는 것이 redo 연산이 된다.

creat 시스템 호출과 mknod, link, symlink 시스템 호출은 모두 redo 연산을 통해 링크를 생성하여 해결 가능하다. 따라서 [표 1]의 알고리즘에서 creat, mknod, link, symlink는 같은 방식으로 처리한다.

3. VFS 시스템 호출 로그 기반 CoW 처리 절차

CoW는 스냅샷 대상인 Source Directory의 스냅샷 생성 시점의 이미지를 보존하는 기능을 수행한다. 이 논문에서 개발한 CoW방법은 기본적으로 파일을 논리적 청크(모든 청크들을 같은 크기) 들의 집합으로 본다. 스냅샷 대상 Source Directory의 파일들에 대해서 변경이 수행될때는 먼저 변경할 청크들을 찾아내고 이를 Snapshot Directory에 복제한 후 변경을 수행한다. 이때 변경된 청크들은 스냅샷 테이블에 기록한다.

또한, 파일 콘텐츠가 아닌 경로명이 변경되는 경우에는 디렉토리 테이블에 변경전 경로와 변경 후 경로를 기록한다. [그림 3]은 스냅샷 테이블과 디렉토리 테이블의 구조를 보여준다.

ss_path	cow_path	chunk_list	Org_file_size

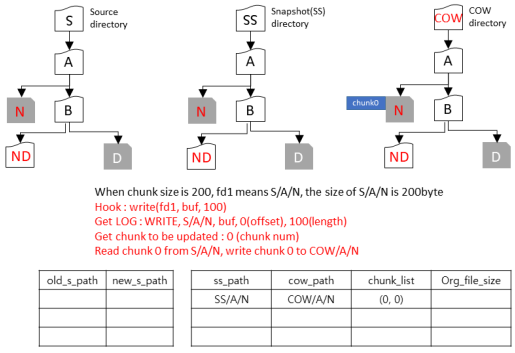
old_s_path	new_s_path

그림 3. 스냅샷 테이블과 디렉토리 테이블 구조

CoW를 실행하는 구체적인 절차는 다음과 같다. [그림 4]는 스냅샷이 생성되어 Source Directory에 대한 Snapshot Directory 와 CoW Directory가 생성된 상태이다. 이 그림에서 청크 크기는 200byte 이고 fd1은 경로 S/A/N의 파일 기술자를 나타낸다. 경로 S/A/N의 크기는 200byte 이다.

Source Directory 의 S/A/N 파일을 변경하는 write(fd1, buf, 100) 가 발생하면 CoW Manager 는 커널에서 실제 연산을 수행하기 전에 write 연산에 대한 로그를 생성한다. [그림 4]에서처럼 로그레코드의 내용은 {WRITE, S/A/N, buf, 0(offset), 100(length)}과 같다.

또한, 파일 S/A/N에서 write 연산에 의해서 변경되는 청크를 확인하고 해당 청크를 파일 S/A/N에서 읽어서 CoW Directory 에 복사한다. 이 그림에서는 0번 청크를 읽어서 COW/A/N 에 복사한다. 그리고, 청크를 복사한 내용을 스냅샷 테이블에 기록한다.



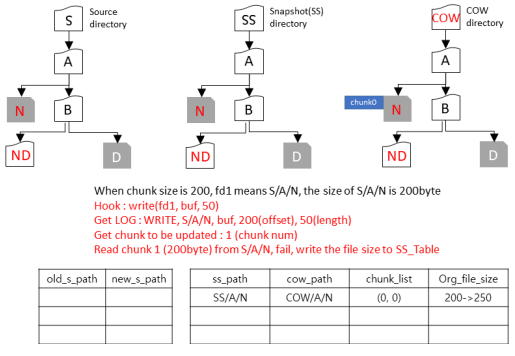
old_s_path	new_s_path

ss_path	cow_path	chunk_list	Org_file_size
SS/A/N	COW/A/N	(0, 0)	

그림 4. write 연산에 대한 CoW 처리 절차

[그림 5]는 wirte(fd1, buf, 50) 연산이 발생했을 때를 보여준다. CoW Manager는 역시 커널에서 실제 연산을 수행하기 전에 write 연산에 대한 로그를 생성한다. [그림 5]에서처럼 로그레코드의 내용은 {WRITE, S/A/N, buf, 200(offset), 50(length)} 과 같다.

다음으로는 해당 파일에서 변경될 청크를 찾아낸다. 청크 크기가 200byte 이므로 이 경우 청크 번호는 1번이 된다. 하지만, 이 경우에 파일의 크기가 200byte 이므로 write연산은 append에 해당한다. 따라서 CoW의 필요성은 없고 스냅샷 테이블의 Org_file_size 컬럼에 변경된 파일 크기인 250을 기록한다.



old_s_path	new_s_path

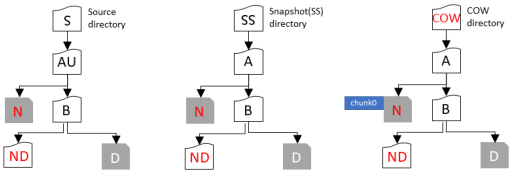
ss_path	cow_path	chunk_list	Org_file_size
SS/A/N	COW/A/N	(0, 0)	200->250

그림 5. append 연산 처리 절차

[그림 6]은 rename 연산에 대한 처리 절차를 보여준다. rename(S/A, S/AU)는 경로 S/A를 S/AU로 변경하는 연산이다. rename 연산이 발생할 경우 이를 기록하지 않으면 추후에 CoW 연산이나 스냅샷 접근 연산

을 정상적으로 처리할 수 없다.

CoW Manager는 rename(S/A, S/AU) 연산이 발생하면 이를 실행하기 전에 먼저 이에 대한 로그레코드를 생성한다. 그림에서처럼 로그레코드는 {RENAME, S/A, S/AU} 로 구성된다. CoW Manager는 이 로그레코드를 아래 그림처럼 디렉토리 테이블에 기록한다. 디렉토리 테이블의 역할은 경로가 변경된 파일에 대해서 CoW 연산을 수행하거나 추후에 스냅샷을 접근할 때 사용된다.



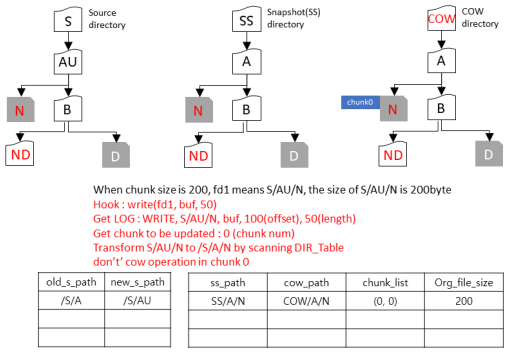
old_s_path	new_s_path
/S/A	/S/AU

ss_path	cow_path	chunk_list	Org_file_size
SS/A/N	COW/A/N	(0, 0)	200

그림 6. rename 연산에 대한 CoW 처리 절차

[그림 7]은 경로명이 변경된 S/AU/N에 대한 write 연산이 수행되는 절차를 보여주고 있다. CoW Manager는 write(fd1, buf, 50) 연산이 발생하면 커널에서 수행되기 전에 먼저 CoW 연산을 시도한다. 먼저 write 연산을 분석하여 로그레코드를 생성한다. 그림에서는 {WRITE, S/AU/N, buf, 100(offset), 50(length)}와 같은 로그레코드를 생성한다.

또한, S/AU/N에서 변경되는 청크를 찾는다. 이 그림에서는 0번 청크가 변경된다. 해당 청크에 대해서 CoW의 필요성을 확인하기 위해 먼저 디렉토리 테이블을 스캔하여 S/AU/N의 경로명이 변경되었는지 확인한다. 이 그림에서는 /S/A/N 경로가 원래 경로임을 확인한다. 다시 /S/A/N 경로에 대해서 스냅샷 테이블을 스캔해서 해당 청크의 CoW 여부를 확인한다. 이 그림에서는 해당 청크가 이미 CoW되어 있으므로 CoW를 수행하지 않고 종료한다.



old_s_path	new_s_path
/S/A	/S/AU

ss_path	cow_path	chunk_list	Org_file_size
SS/A/N	COW/A/N	(0, 0)	200

그림 7. rename 된 파일에 대한 write 연산에 대한 CoW 처리 절차

IV. 성능평가

이 논문에서는 제안하는 스냅샷 방법의 성능을 평가하기 위해 스냅샷 생성시간과, 스냅샷 생성 후 IO 성능 저하를 측정하였다. 스냅샷 생성 후 IO 성능 저하는 스냅샷 이미지를 유지하기 위해 대상 디렉토리 및 파일의 변경시 CoW 연산을 수행할 때 IO에 미치는 영향을 측정하기 위한 것이다.

[표 2]는 이 논문에서 수행한 실험의 HW 및 SW, 데이터 환경을 보여준다. 실험에 필요한 워크로드는 vdbench를 이용하였으며 블록크기는 4Kbytes, IO 주소는 균등 분포에 따라 생성하였다. IO 쓰레드 수는 8로 하였다.

먼저 [표 2]에서의 데이터로 스냅샷 생성시간을 측정했을 때 온라인 스냅샷 생성시간은 0.09 초였다. 스냅샷 생성시간은 파일 수나 디렉토리의 깊이에 따라 달라질수 있지만 대체적으로 1초 내외로 생성이 되었다. [그림 8]은 vdbench를 이용해서 스냅샷을 생성했을 때 (Snapshot) 와 생성하지 않았을 때 (Normal) 읽기-쓰기 연산의 비율을 0:100에서 90:10 변화시켜 가면서 IOPS를 측정한 것이다. 평균적으로 Normal 과 Snapshot의 IOPS 성능 차이는 약 13.5% 였다. 즉, Snapshot이 Normal보다 IOPS 성능이 13.5% 낮았다.

표 2. 실험환경

구분	내용
CPU	Archlinux, Kernel Ver. 4.9
Storage	Samsung SSD 960 EVO 250GB
CPU/RAM	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz / 24GiB
File System	ext4
Data	Number of files : 20,000
	Number of directory : 1,000
	Total size : 76 GB
Vdbench Parameters	xfersize=4k
	fileio=random, fileselect=random
	threads=8

Snapshot을 생성하고 vdbench를 실행시킬 때 성능이 더 낮아지는 이유는 시스템 콜을 가로채기 연산과 CoW 연산을 추가로 해야 하기 때문이다. 대부분의 스냅샷 방법들이 스냅샷을 생성했을 때 IO 성능이 저하되는 공통적인 원인이다. 리눅스 LVM 의 경우에는 같은 실험을 수행했을 때 스냅샷으로 인한 성능저하가 약 67% (쓰기 연산 100%)에 달하는 것을 확인할 수 있었으며 이에 비하면 제안하는 방법은 스냅샷이 생성된 상황에서도 성능저하가 상대적으로 매우 작음을 볼수 있었다.

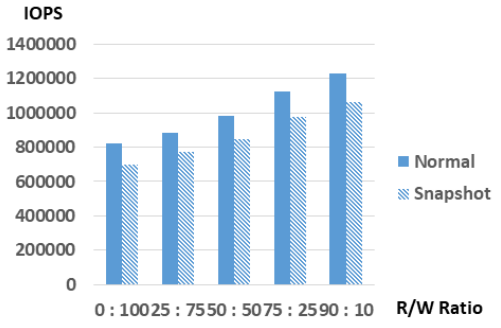


그림 8. 읽기-쓰기 비율에 따른 IOPS 비교

V. 결 론

이 논문에서는 가상파일시스템의 변경연산에 대한 로그를 추출하여 CoW 방식의 스냅샷 기술을 제안하고 구현하였다. 이를 통해서 제안하는 스냅샷 기술은 파일시스템, 가상 블록장치, 하드웨어에 독립적으로 동작할 수 있다. 또한, 하드링크를 기반으로 하는 디렉토리 스냅샷 방법을 통해 온라인으로 스냅샷을 생성하는 방법을 제안하였다. 이를 통해 제안하는 스냅샷 기술은 디렉토리 및 파일 단위의 스냅샷을 스토리지 서비스 중단 없이 생성하고 관리 할수 있다. 제안하는 스냅샵 방법을 리눅스 운영체제를 기반으로 구현하고 성능평가를 실시하여 스냅샷 생성시간과 스냅샷 생성으로 인한 입출력성능저하를 측정하였다. 측정결과 약 13.5%의 성능 저하가 일어남을 확인할 수 있었다. 같은 실험을 LVM 의 스냅샷에 대해 수행했을 때 약 67%의 성능저하가 발생하는 것에 비하면 적절한 수준이라고 판단된다. 제안하는 스냅샷 방법을 활용하면 기존 스토리지 하드웨어를 구매하거나 LVM 또는 스냅샷을 지원하는 파일시스템을 반드시 사용하지 않고도 다양한 데이터 보호 응용 개발이 가능하다.

참 고 문 헌

[1] P. Nayak and R. Ricci, *Detailed study on Linux Logical Volume Manager*, Flux Research Group University of Utah, 2013.

[2] L. Ellenberg, "Drbd 9 and Device-mapper: Linux Block Level Storage Replication," Proceedings of the 15th International Linux System Technology Conference, 2008.

[3] G. Navarro and M. Manic, "FuSnap: Fuzzy Control of Logical Volume Snapshot Replication for Disk Arrays," IEEE Transactions on Industrial Electronics, Vol.58, No.9, pp.4436-4444, 2011.

[4] W. J. Xiao, Q. Yang, J. Ren, C. S. Xie, and H. Y. Li, "Design and Analysis of Block-Level Snapshots for Data Protection and Recovery," IEEE Transactions on Computers, Vol.58, No.12, pp.1615-1625, 2009.

[5] E. K. Lee and C. A. Thekkath, "Petal: Distributed Virtual Disks," Proceedings of the 7th International Conference on Architectural

Support for Programming Languages and Operating Systems, Cambridge, MA, USA, pp.84-92, 1996.

[6] Z. Peterson and R. BURNS, "Ext3cow: a Time-shifting File System for Regulatory Compliance," ACM Transactions on Storage (TOS), Vol.1, No.2, pp.190-212, 2005.

[7] R. Strobl and O. Evangelist, "Zfs: Revolution in File Systems," Sun Tech Days, 2008.

[8] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-tree Filesystem," ACM Transactions on Storage (TOS), Vol.9. No.3, p.9, 2013.

[9] "Data Backup Apparatus and Method for The Same," Korea Patents, 1012377460000, 2013.02.21.

저 자 소 개

김 진 수(Jinsu Kim) 정회원

- 2015년 8월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2017년 2월 : 한국교통대학교 컴퓨터공학과(공학석사)

〈관심분야〉 : 스토리지, 분산 및 병렬 처리, 머신러닝 등

송 석 일(Seokil Song) 정회원

- 1998년 2월 : 충북대학교 정보통신공학과(공학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2003년 2월 : 충북대학교 정보통신공학과(공학박사)

〈관심분야〉 : 데이터베이스, 스토리지, 분산 및 병렬 처리 등

신 재 룡(Jae Ryong Shin) 정회원

- 1996년 2월 : 충북대학교 정보통신공학과(공학사)
- 1998년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2002년 8월 : 충북대학교 정보통신공학과(공학박사)
- 2003년 3월 ~ 현재 : 광주보건대학교 보건행정과 교수

〈관심분야〉 : 데이터베이스시스템, 실시간, 멀티미디어