qtar: 플래시 변환 계층 리매핑 기법을 이용한 최적화된 tar 명령어 구현

유정석O, 한상욱, 김지홍
서울대학교 컴퓨터공학부
{js_ryoo, shanehahn, jihong}@davinci.snu.ac.kr

qtar: Design and Implementation of the Optimized tar Command with FTL-level Remapping Technique

Jeongseok RyooO, Sangwook Shane Hahn, Jihong Kim
Department of Computer Science and Engineering, Seoul National University

요 약

tar는 여러 개의 파일들을 묶어 하나의 파일로 만들어 주는 명령어로써, 여러 종류의 파일 압축프로그램과 결합하여 보편적으로 사용되는 명령어이다. tar와 결합한 압축프로그램은 특히 압축 대상들 중에 작은 파일이 많이 포함된 경우 유용하게 사용된다. 하지만, tar는 오히려 작은 파일들을 처리할 때 성능이 가장 떨어지는 문제점이 있다. 본 논문에서는 이 성능 하락이 tar의 읽기 처리 과정에 있음을 파악하고 이를 플래시 변환 계층 리매핑 기법을 활용해 해결하여 qtar (Quick tar)를 구현하였다. 그리고 에뮬레이터를 통한 실험으로 qtar가 tar에 비해 최대 3.4배 빠르게 동작하는 것을 확인하였다.

1. 서 론

tar (Tape Archive)는 여러 개의 파일들을 테이핑(Taping)하여 하나의 아카이브(Archive) 파일로 만들어주는 리눅스 기본 명령어로써, 여러 종류의 파일 압축프로그램과 결합하여 보편적으로 사용되고있다. tar와 압축프로그램의 결합이 많이 쓰이게 된 이유는 크게 두 가지가 있다. 먼저 tar는 파일들을 테이핑할 때, 각 파일의 리눅스 파일 속성(uid, pid, permission 등)을 포함한다. 따라서, tar와 압축프로그램을 결합하면 이 속성들을 보존한 채 압축을 할 수 있다는 장점이 있다. 또한, tar와 결합한 압축프로그램은 대상 파일들의 크기와 관계없이 좋은 압축률을 갖는다. 압축프로그램 zip은 대상 파일 들을 먼저 하나씩 압축한 후 압축된 파일들을 테이핑하여 하나의 파일로 만들기 때문에, 대상 파일들 중에 작은 파일이 많이 포함되어 있는 경우 전체 압축률이 현저히 떨어지게 된다. 그러나 tar와 결합한 압축프로그램은 tar를 이용해 전체를 먼저 하나의 파일로 테이핑한 뒤 그 파일을 압축하기 때문에, 대상 파일들 중에 작은 파일이 많이 포함되어 있어도 전체 압축률에 영향을 받지 않는다.

하지만 tar는 대상 파일들을 하나씩 처리 하도록 구현되어있어 문제가 생긴다. tar는 하나의 파일을 읽어 tar파일 형식으로 변환하여 저장하고, 그 다음 하나의 파일을 읽어 같은 처리 과정을 거쳐 앞서 변환한 파일 뒤에 붙이기를 반복하며 진행된다. 이로 인해 tar의 읽기 요청은 파일 하나 단위로 발생하여, 만약 대상 파일들 중에 작은 파일이 많이 포함되어 있을 경우 극심한 I/O 성능 저하로 이어진다.

그림 1은 Samsung 840 Pro SSD의 I/O 요청 크기 별 성능을 나타낸 것이다. FIO 벤치마크 [1]를 이용해 총 1 GB의 I/O 요청을 다양한 크기로 나누어 발생시킨 후 대역폭을 측정

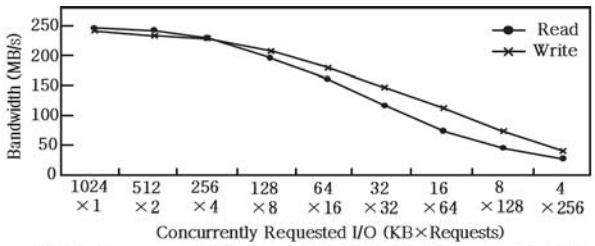

그림 1. Samsung 840 Pro SSD의 I/O요청 크기 별 성능

하였는데, 그 결과 I/O 요청의 크기가 클수록 대역폭이 증가하는 것을 볼 수 있었다. 따라서, 호스트(Host)에서 I/O 요청을 되도록 크게 만들어 내려주어야 SSD의 대역폭을 높일 수 있다는 것을 알 수 있다. 그러나 tar는 한 파일 단위로 읽기 요청이 발생하기 때문에 각 파일의 크기가 작을 경우 읽기 요청의 크기가 작아져 SSD의 대역폭이 크게 떨어지게 된다.

결론적으로, 작은 파일들이 많이 포함된 대상을 압축할 때 tar를 이용하는 것은 압축률 면에서는 유리하지만 I/O 성능 면에서는 비효율적인 것을 알 수 있다. 본 연구에서는 이러한 문제를 플래시 변환 계층(Flash Translation layer, FTL) 리매핑(Remapping) 기법을 활용해 해결하고 작은 파일들을 대상으로 압축률과 I/O 효율성이 모두 좋은 qtar (Quick tar)를 구현하였다. qtar는 1) 먼저 tar의 모든 대상 파일들의 크기의 합과 동일한 크기를 갖는 하나의 임시 파일을 생성하고 2) 리매핑 기법을 활용해 이 임시 파일이 모든 대상 파일들의 데이터를 테이핑 할 순서에 맞게 가리키게 만든다. 그리고 3) 테이핑이 시작되어 첫 번째 파일에 대한 읽기 요청이 발생하면, 데이터 버퍼를 생성하고 임시 파일을 통해 전체 데이터를 한번에 읽어 이 버퍼에 넣어두고 4) 이후에 발생하는 각 파일의 읽기 요청을 데이터 버퍼에서 바로 반환하도록 구현하여 tar에서

발생하던 I/O 성능 저하 문제를 없애고 SSD 최대 대역폭을 활용할 수 있도록 구현 하였다.

SSD 에뮬레이터 [2]에서 다양한 크기의 테스트 파일들을 tar와 qtar로 테이핑하는 시간을 비교해본 결과 qtar가 tar보다 최대 3.4배 빠르게 완료되는 것을 확인하였고, Linux 4.4.52 (725 MB, 파일 58779 개)를 테이핑하는 시간은 qtar가 tar보다 평균 2.4배 빠르게 완료되는 것을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 tar의 동작 과정과 플래시 변환 계층 리매핑 기법에 대해서 설명하고, 3장에서는 qtar의 구현과 동작 과정에 대해 설명한다. 이어서 4장에서는 SSD 에뮬레이터를 통한 실험 결과를 보이고, 5장에서 결론을 맺는다.

2. 배경지식

2.1 tar의 동작 과정

tar는 여러 파일들을 테이핑하여 하나의 아카이브 파일로 만드는 동작을 수행한다. 사용자는 테이핑할 대상 파일들의 이름 또는 그 파일들이 속해 있는 디렉터리들의 이름을 입력하여 tar를 실행시킨다. tar는 사용자가 입력한 이름들을 테이핑 대상 리스트(Target_List)에 저장하고 이 이름들을 하나씩 tar 파일 형식으로 변환한다. tar 파일 형식은 파일의 메타데이터를 저장하는 헤더 블록(Header block)과 파일의 데이터를 저장하는 데이터 블록(Data block)으로 이루어진다. 디렉터리는 데이터가 없으므로 헤더 블록으로만 변환되고, 파일은 데이터를 갖고 있으므로 헤더 블록과 데이터 블록으로 변환된다. 디렉터리에 속해 있는 파일은 디렉터리를 변환할 때 알아내어 마찬가지로 하나씩 변환한다. 이 파일 하나 단위로 진행되는 tar의 테이핑 과정으로 인해 파일의 데이터에 대한 읽기 요청 또한 파일 하나 단위로 발생하고, 테이핑 대상들 중에 작은 파일이 많을수록 작은 읽기 요청이 많이 발생하여 결국 전체 I/O 성능이 떨어지게 된다.

2.2 플래시 변환 계층 리매핑 기법

낸드 플래시 기반 저장 장치(NAND flash-based storage)는 덮어쓰기가 안되는 특징을 갖고 있기 때문에 별도의 플래시 변환 계층을 필요로 한다. 호스트에서 덮어쓰기를 하면, 플래시 변환 계층에서 새로운 물리 페이지 주소(Physical Page Address, PPA)에 덮어쓸 데이터를 저장한 후 이 주소를 대상 논리 블록 주소(Logical Block Address, LBA)에 매핑 시켜 덮어쓰기를 완료한다. 따라서, LBA와 PPA간의 매핑 정보가 계속 바뀌게 되고 플래시 변환 계층에서 이 정보를 주소 매핑 테이블(Address Mapping Table)에 저장하여 관리한다.

플래시 변환 계층 리매핑 기법이란, 이 주소 매핑 테이블의 정보만을 수정하며 물리적으로 데이터를 이동시킨 효과를 얻는 기법을 말한다 [3, 4]. 호스트에서 리매핑 명령어 remap(sLBA, tLBA)을 통해 소스(Source) LBA와 타깃(Target) LBA를 넘겨주면, 주소 매핑 테이블에서 소스 LBA의 PPA를 알아내고 타깃 LBA가 이 PPA를 가리키게 한 후 소스 LBA의 매핑 정보를 삭제하여, 마치 소스 LBA의 데이터를 타깃 LBA로 이동시킨 것 같은 효과를 얻게 된다.

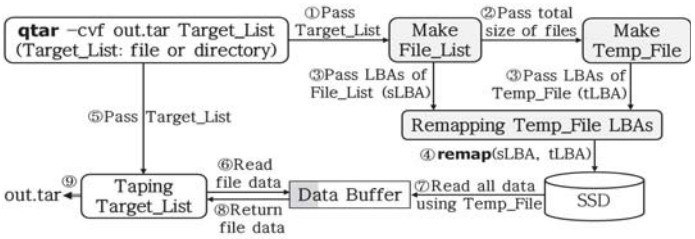

그림 2. qtar의 동작 과정

3. qtar (Quick tar)

3.1 qtar의 구현

qtar는 tar의 I/O 성능 하락 문제를 해결하기 위해 설계되었다. 기존 tar의 문제를 해결하기 위해서는 두 가지의 요구사항이 있다. 1) 먼저, 읽기 요청이 파일 하나 단위가 아닌 여러 파일 단위로 이루어져야 하고 2) 또한, 함께 요청된 파일들의 LBA가 모두 인접해 있어야 한다. 실제 디바이스로 전송되는 I/O는 I/O가 만들어진 파일 시스템에서 결정되는 것이 아니라 블록 I/O 계층에서 결정되기 때문이다. 예를 들어, 하나의 큰 파일에 대해 읽기 요청을 하더라도 그 파일의 LBA들이 많이 단편화(Fragmented)되어 있으면 블록 I/O 계층에서 여러 개의 읽기 요청으로 나누어지게 되고, 파일 시스템에서 여러 파일에 대한 읽기 요청을 하더라도 그 요청들의 LBA들이 인접해 있으면 블록 I/O 계층에서 하나의 읽기 요청으로 합쳐지게 된다. 따라서, tar의 읽기 문제를 해결하기 위해서는 tar의 읽기를 한 파일 단위에서 여러 파일 단위로 바꾸는 것뿐만이 아니라 모든 대상 파일의 LBA들 또한 인접하도록 모아 주어야만 한다. qtar는 이 두 가지 요구사항을 리매핑 기법을 통해 만족시켰다.

3.2 qtar의 동작 과정

qtar는 테이핑 전에 리매핑 과정을 추가하여 각 파일의 크기에 관계 없이 항상 좋은 I/O 성능을 갖도록 구현되었다. qtar의 전체적인 동작 과정은 그림 2와 같다. 테이핑 과정에서 데이터의 읽기 요청은 파일에 대해서만 발생하기 때문에 먼저 테이핑 대상 리스트(Target_List)에서 디렉터리들의 이름을 그 디렉터리 속 파일 이름들로 바꾸어 파일 리스트(File_List)를 생성한다. 그리고 이 리스트 속 파일들의 전체 크기 합을 구해 이와 같은 크기를 갖는 임시 파일(Temp_File)을 생성하고 리매핑을 통해 이 파일이 File_List 속 파일들의 데이터를 순서대로 가리키도록 만든다. (qtar의 리매핑 기법은 3.3절에서 추가적으로 설명한다.) 결과적으로, 리매핑이 완료된 후 Temp_File은 테이핑 과정에서 읽게 될 데이터를 그 데이터를 읽을 순서에 맞게 가리키고 있는 상태가 된다. 다음으로 테이핑이 시작되고 첫 번째 대상 파일에 대한 읽기 요청이 발생하면, qtar는 데이터 버퍼를 생성하고 Temp_File의 전체 데이터를 모두 읽어 이 버퍼에 저장한 후 각 파일 읽기 요청에 해당하는 데이터를 이 데이터 버퍼에서 꺼내 반환해준다. 만약 Temp_File의 크기가 너무 커서 전체가 메모리에 수용될 수 없으면, 수용 가능한 만큼만 먼저 읽어 처리한 후 나머지를 읽어온다. 이처럼 qtar는 전체 데이터를 한번에 읽어 처리하기 때문에 tar에서 대상 파일들의 크기가 작을 때 발생하던 I/O 성능 하락 문제가 발생하지 않게 된다.

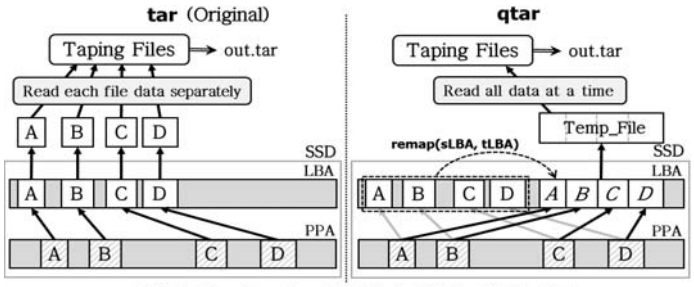

그림 3. tar와 qtar의 읽기 처리 과정 비교

3.3 qtar의 리매핑 기법

현재 상용화된 SSD의 플래시 변환 계층은 리매핑을 지원하지 않기 때문에, 본 논문에서는 SSD 에뮬레이터 [2]를 수정하여 리매핑을 지원하는 SSD를 구현하였다. 앞서 3.2절에서 설명한 것과 같이 qtar는 읽어야 할 파일들의 데이터를 임시 파일을 통해 대신 읽어 처리한다. 그리고 모든 테이핑 과정이 끝나면 이 임시 파일은 바로 삭제되게 된다. 다시 말해, qtar의 리매핑은 잠시 동안 하나의 파일이 다른 여러 파일들의 데이터를 모두 갖고 있는 것처럼 만들기 위해 사용된다. 이는 기존에 데이터의 이동을 위해서만 쓰이던 리매핑 기법을 새로운 각도로 이용한 것이다 [3, 4]. 그림 3을 보면 tar는 네 개의 파일들에 대한 데이터를 각각 따로 읽어오고 있는 것에 반해 qtar는 리매핑 기법을 통해 한번에 읽어오고 있는 것을 볼 수 있다.

4. 실 험

4.1 실험 환경

Intel i7-2600 CPU, 16 GB 크기의 메인 메모리로 동작하는 Linux 4.4.52 환경에서 SSD 에뮬레이터 [2]가 제공하는 4 GB 공간에 실험 대상 파일들을 저장한 후 tar와 qtar를 이용해 테이핑을 하는 성능을 비교하였다. 에뮬레이터는 remap 명령어를 지원하도록 수정하였다.

4.2 실험 결과

그림 4는 총 1 GB가 되는 다양한 크기의 파일들을 tar와 qtar로 테이핑하는데 걸린 시간을 tar를 기준으로 정규화(Normalize)하여 나타낸 것이다. qtar는 읽기 성능을 향상시키기 위해 리매핑을 이용한다. 이는 추가된 작업이기 때문에 만약 테이핑 대상이 읽기 성능을 개선할 여지가 없이 이미 큰 파일들로 이루어진 경우 오히려 qtar의 실행 시간이 tar에 비해 증가할 수 있다. 그러나 파일 크기가 1 MB 이하일 경우 qtar가 tar보다 항상 빠르고, 파일 크기가 4 KB일 경우 qtar가 tar보다 3.4배 빠르게 동작하였다.

그림 5는 Linux 4.4.52 (725 MB, 파일 58779 개)를 tar와 qtar를 이용해 테이핑한 결과를 비교한 것이다. 그림 5(a)는 tar와 qtar에서 발생한 읽기 요청들을 먼저 크기 순으로 분류한 후 각 요청들이 접근한 LBA를 시간 순으로 나타낸 것이다. tar는 파일 하나 단위로 읽기 요청이 발생하여 주로 작고 비순차적인 읽기 요청들이 발생하고 있는 것에 반해 qtar는 리매핑 기법으로 인해 주로 크고 순차적인 읽기 요청들이 발생하고 있는 것을 볼 수 있다. 이러한 읽기 요청

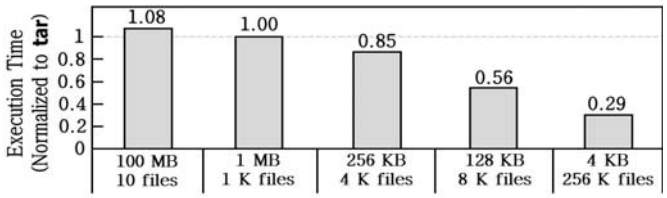

그림 4. 파일 크기 별 qtar의 실행 시간

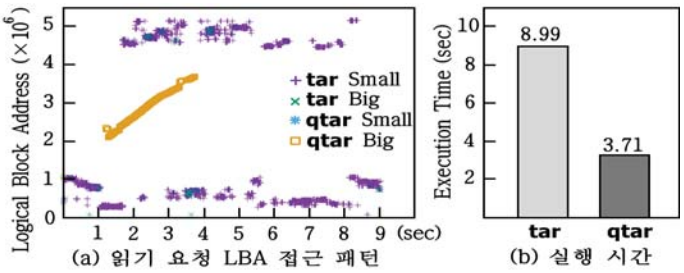

그림 5. Linux 4.4.52 테이핑 결과 비교

패턴의 변화는 I/O 성능을 크게 향상시켜 그림 5(b)와 같이 전체 실행 시간이 감소되는 효과를 얻게 된다. 여러 차례의 실험을 통해 Linux 4.4.52를 테이핑하는 시간이 qtar가 tar에 비해 평균 2.4배 빠른 것을 확인하였다.

5. 결 론

tar는 여러 개의 파일들을 테이핑할 때 보편적으로 사용되는 명령어이지만, 대상 파일들 중에 작은 파일이 많이 포함된 경우 성능이 급격히 하락하는 문제가 있다. 본 논문에서는 이 성능 하락이 tar가 작은 파일들을 읽을 때 발생하는 작고 비순차적인 읽기 요청으로 인해 SSD의 대역폭을 충분히 활용하지 못하는 데에서 오는 것을 확인하고, 플래시 변환 계층 리매핑 기법을 이용해 이 읽기 요청들을 크고 순차적인 읽기 요청으로 바꾸어 I/O 성능 하락을 없애고 qtar를 구현하였다. 그리고 SSD 에뮬레이터를 이용한 실험으로 qtar가 tar에 비해 최대 3.4배 빠르게 동작하는 것을 확인하였다.

감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다. 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015M3C4A7065645). (교신저자: 김지홍)

참 고 문 헌

[1] Flexible I/O Tester. https://github.com/axboe/fio.

[2] J. Sang-Woo et al., “BlueDBM: An Appliance for Big Data Analytics,” Proc. of the ACM/IEEE Annual International Symposium on Computer Architecture, pp. 1-13, 2015.

[3] C. Hyun-Jin et al., “JFTL: A Flash Translation Layer based on a Journal Remapping for Flash Memory,” ACM Transactions on Storage, Vol. 4, No. 14, 2009.

[4] K. Hyukjoong et al., “SHRD: Improving Spatial Locality in Flash Storage Accesses by Sequentializing in Host and Randomizing in Device,” Proc. of the USENIX Conference on File and Storage Technologies, pp. 271-283, 2017.