스마트폰의 어플리케이션 업데이트 패턴을 고려한 데이터 중복제거 기법 연구

박대준°, 최동수, 신동군
성균관대학교 컴퓨터공학과
pdaejun@skku.edu, echosoul@skku.edu, dongkun@skku.edu

Deduplication Technique for Smartphone Application Update Scenario

Daejun Park°, Dongsoo Choi, Dongkun Shin
Department of Computer Engineering, Sungkyunkwan University

요 약

스마트폰의 어플리케이션은 어플리케이션 생태계의 발전에 따라 그 수가 많아지고, 업데이트 또한 잦아졌다. 어플리케이션의 업데이트는 낸드 플래시 메모리에 이전 버전을 삭제하고, 새로운 버전의 어플리케이션에 대한 쓰기 명령을 내린다. 따라서 사용자는 낸드 플래시 메모리에서의 상대적으로 느린 쓰기 명령에 의해 스마트폰의 성능의 저하를 느끼고 낸드 플래시 메모리는 반복되는 지우기/쓰기 동작에 의해 수명이 단축된다. 본 논문에서는 업데이트 되는 스마트폰 어플리케이션 데이터가 이전 버전과 큰 차이가 없다는 것에 착안하여 데이터 중복제거를 통해 업데이트 성능을 향상시키고 낸드 플래시 메모리의 수명을 향상시키는 기법을 제안하고 있으며, 실험을 통해서 어플리케이션들에 대한 중복 제거율을 관찰하였다.

1. 서 론

스마트폰의 확산에 따라 어플리케이션의 생태계가 바뀌고 있다. 정적이던 어플리케이션의 생태계가 스마트폰의 보급으로 인해 어플리케이션 마켓을 통해 누구에게나 열린 활발한 공간으로 변하였다. 이로 인해 어플리케이션의 종류와 수가 다양해졌다.

특히 사용자의 수요를 충족시키기 위해 어플리케이션의 업데이트가 잦아졌는데, 이러한 어플리케이션의 업데이트는 새로운 버전의 어플리케이션을 마켓에서 다운받아 낸드 플래시 메모리에 기록하는 방식으로 이루어진다.

스마트폰에서 쓰이는 대표적인 저장장치인 낸드 플래시 메모리(NAND Flash Memory)는 저 전력, 비휘발성, 충격 내구성이 큰 장점을 가지고 있다. 그러나 쓰기 성능이 읽기 성능에 비해 떨어지고, 물리적 특성에 의해 쓰고 지우는 횟수가 한정되어 있다. 현재 쓰이고 있는 업데이트는 이전 버전에서 변경된 부분만을 다운받아 기록하는 방식이 아니라, 변경되지 않은 부분도 포함된 완전한 어플리케이션을 다운받아 낸드 플래시 메모리에 기록한다. 그러나 어플리케이션의 버전 업데이트 시에 보통 이전버전의 코드와 크게 다르지 않고, 작은 범위의 수정이 이루어지기 때문에 이전 버전과의 파일을 비교하면 크게 다르지 않다.

스마트폰에 30개의 어플리케이션을 설치한다고 가정하고, 가장 인기 있는 어플리케이션 30개를 살펴보니[1] 평균 어플리케이션의 크기는 16MB, 업데이트 간격은 21일이었다. 이런 업데이트 패턴을 가진 어플리케이션을 항상 최신버전으로 유지하기 위해서는 한 달에 1,015MB에 달하는 데이터를 낸드 플래시 메모리에 쓰고 지우게 된다.

삼성의 갤럭시 S2(모델명 I9100)[2]를 기준으로 볼 때, 안드로이드에서 어플리케이션이 기록되는 데이터 파티션과 시스템 파티션의 크기가 총 2.6GB임을 참고하면, 한 달 동안 파티션의 37.8퍼센트에 달하는 공간에 어플리케이션의 업데이트를 위해 쓰고 지우는 동작을 수행하게 된다.

그러나 대부분의 업데이트가 전체 어플리케이션의 작은 부분의 수정이라는 것을 볼 때, 낸드 플래시 메모리에 불필요한 쓰기와 지우기 명령이 수행된다고 할 수 있다.

이러한 문제점을 해결하기 위해서 데이터 중복 제거 기법이 사용될 수 있다. 데이터 중복 제거 기법이란 중복되는 데이터를 검출하여 쓰기 명령에서 제외시키는 기법이다. 이 기법을 통해 어플리케이션의 업데이트를 수행할 경우에는 같은 데이터에 대하여 쓰기 명령을 내리지 않도록 해서 불필요한 쓰기와 지우기 명령을 줄일 수 있을 것이다. 낸드 플래시 메모리를 위해서 제안된 데이터 중복 제거 기법에는 CAFTL[3] 등이 있다. 하지만 기존 기법은 서버나 데스크탑 컴퓨터의 일반적인 데이터를 대상으로 했으며, 본 연구와 같이 스마트폰을 대상으로 한 중복 제거 기법은 제시된 적이 없다.

본 연구에서는 스마트폰의 어플리케이션 업데이트 패턴에서 얼마나 많은 중복 데이터가 존재하는 지 관찰하고, 스마트폰 어플리케이션 업데이트에 적합한 데이터

본 연구는 지식경제부 및 정보통신산업진흥원의 지원사업의 연구결과로 수행되었음

중복 제거 기법을 제안하고 있다. 또한, 실험을 통해 가변 크기 데이터 중복 제거 기법의 적절한 설정을 제시하고 있다.

2. 관련 연구

그림 1은 데이터 중복 제거 기법에 대한 설명이다. 처음 쓰기 요청이 들어온 데이터를 청크(chunk)라는 단위로 구분한다. 청크는 해쉬 함수를 통해 청크 고유의 핑거프린트[4](fingerprint)를 가지게 된다. 핑거프린트는 청크가 중복되는 지 확인하기 위해서 기존 데이터의 핑거프린트와 동일한 것이 있으면 같은 청크로 분류한다. 이미 동일한 청크가 저장되어 있으면 청크를 스토리지에 기록하지 않으며, 이미 저장된 청크로 접근을 하도록 매핑 정보를 생성한다.

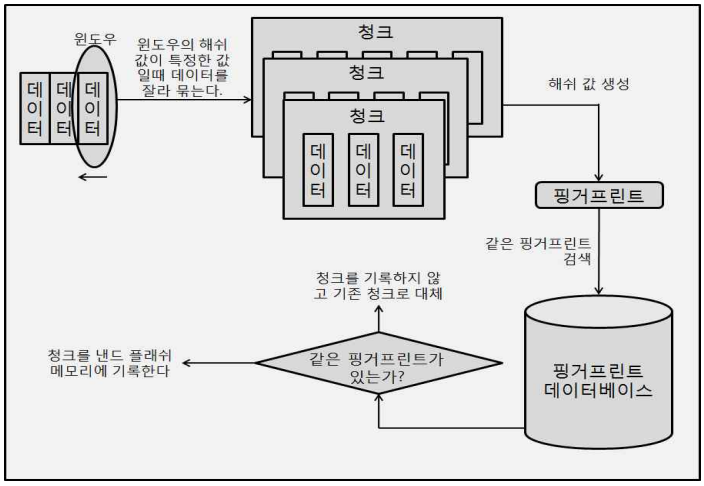

그림 1 데이터 중복 제거 기법의 기본 원리

데이터 중복 제거는 이런 방식을 통해 저장 공간을 적게 쓸 수 있고, 불필요한 쓰기 명령이 생기지 않아, 쓰기 성능이 좋지 않은 낸드 플래시 메모리에서 성능이 향상되는 결과를 보여준다.[5]

데이터 중복 제거에서는 청크의 크기가 고정적인지 가변적인지에 따라 고정 길이 데이터 중복 제거와 가변 길이 데이터 중복 제거로 구분된다. CAFTL을 포함한 낸드 플래시 메모리에서의 데이터 중복 제거 기법은 고정된 크기의 청크를 사용하게 되는데, 가장 큰 이유는 플래시 메모리에서 쓰기 명령의 단위인 페이지가 고정된 크기(4KB)를 가지고 있기 때문이다. 스마트폰 역시 낸드 플래시 메모리를 쓰고 있지만 본 논문에서는 스마트폰 어플리케이션 특성에 더 적합한 데이터 중복 제거기법을 찾기 위해 두 기법 모두 검토하였다.

3. 데이터 중복 제거기법

3.1 고정 길이 데이터 중복 제거와 가변 길이 데이터 중복 제거

고정 길이 데이터 중복 제거는 고정된 크기의 청크를 생성하고, 해당 청크에 대한 고유한 핑거프린트를 통해 청크의 중복을 검출하여 데이터 중복 제거를 하는 방법이다. 이때 중복된 청크의 검출에 쓰이는 핑거프린트는 충돌율이 매우 낮은 SHA-1[6]과 같은 해시 키 함수를 사용 한다.

가변 길이 데이터 중복 제거는 청크의 크기가 다양하다. 가변 길이 데이터 중복 제거에서는 청크를 나누기 위해서 데이터에 윈도우를 두고, 윈도우에 SHA-1을 이용해 해쉬값을 생성해나간다. 이때 생성된 해쉬값의 하위 비트가 특정한 값을 가지면 데이터를 자르고, 청크로 관리한다.

이때 특정한 값의 하위비트의 개수에 따라서 청크의 평균적인 크기가 정해지는데, 이 하위비트의 개수를 유효 비트수라 한다. 중복된 청크의 검출은 고정 길이 청크 데이터 중복 제거의 방식과 같다.

3.2 데이터 중복 제거율과 청크 크기

데이터 중복 제거에서 가장 중요한 것은 데이터 중복 제거율이다. 데이터 중복 제거율은 (중복된 데이터의 크기 / 총 데이터 크기) 로 표현할 수 있다. 데이터 중복 제거의 목적은 중복된 데이터를 많이 찾아내어 저장 공간을 활용하는 것이므로, 데이터 중복 제거율이 클수록 좋은 데이터 중복 제거이라고 할 수 있다.

고정 길이 데이터 중복 제거는 청크의 크기가 고정되어 있지만 가변 길이 데이터 중복 제거의 경우에는 유효 비트수에 따라 청크의 크기가 다양하다. 본 연구에서 대상으로 하고 있는 스마트폰 어플리케이션의 다른 버전 간 중복을 찾아내기 위해서는 고정 길이 기법보다는 가변 길이 기법이 더 유리하다. 그 이유는 버전 간 차이는 수십 바이트의 데이터가 어떤 지점에서 추가되거나 삭제된 경우가 많으므로 이런 경우에 고정 기법은 중복 데이터를 거의 찾지 못하게 된다.

하지만, 낸드 플래시 메모리의 특성도 고려해야 하는데, 플래시 메모리에서는 쓰기 단위가 페이지로 고정되어 있기 때문에 가변길이의 청크는 관리하기가 복잡해진다. 이러한 두 가지 사항을 고려한 중복 제거 기법의 설계가 필요하다.

다음 실험들에서는 데이터 중복 제거율과 청크의 크기를 기준으로 스마트폰 어플리케이션에 적합한 데이터 중복 제거 방법을 제시하고 있다.

4. 데이터 중복 제거 기법의 적용

4.1 고정 길이 데이터 중복 제거와 가변 길이 데이터 중복 제거 비교

고정 길이 데이터 중복 제거와 가변 길이 데이터 중복 제거를 통해 데이터 중복 제거율을 측정하였다. 실험시 설정은 고정 길이 데이터 중복 제거의 경우 청크 크기를 1KB로 설정하고, 가변 길이 데이터 중복 제거는 청크를 나누는 기준이 되는 유효 비트수를 9로 설정하였다. 어플리케이션은 apk파일 상태로 데이터 중복제거를 수행하였다. 측정한 어플리케이션은 Google maps과 Skype이다.

그림 2는 어플리케이션들의 여러 버전을 수집하여 버

전이 업데이트 될 때, 이전 버전과의 중복되는 부분에 대해 데이터 중복 제거를 수행하여 측정된 데이터 중복 제거율을 나타낸 것이다.

두 어플리케이션 모두 가변 길이 데이터 중복 제거에서 고정 길이 데이터 중복 제거보다 높은 데이터 중복 제거율을 보이므로 가변 길이 데이터 중복 제거를 사용하는 것이 더 유리하다는 것을 알 수 있다.

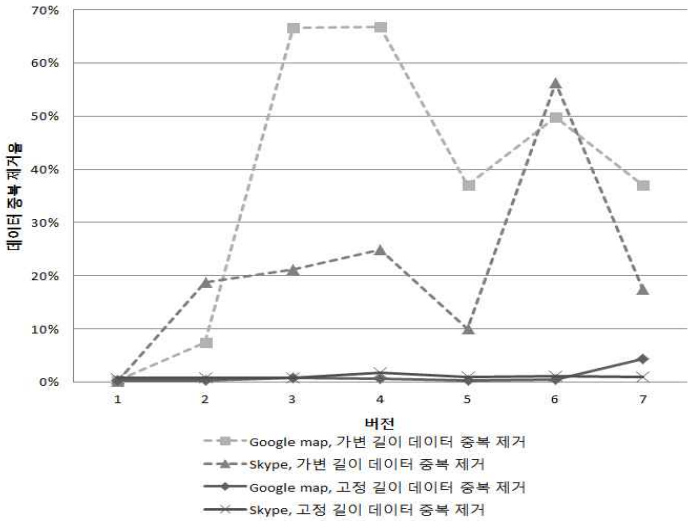

그림 2 고정 길이와 가변 길이 데이터 중복 제거 비교

4.2 유효 비트수와 평균 청크 크기

가변 길이 데이터 중복 제거는 청크 크기가 다양하다. 하지만, 대부분의 청크 크기가 플래시 메모리 페이지의 크기보다 작으면서도 내부 단편화가 작으면 효율적이다. 그러므로 평균 청크의 크기를 결정하는 요소인 유효 비트수를 변경해 가며 평균 청크의 크기를 관찰할 필요가 있다. 유효 비트수가 많아질수록 평균 청크의 크기는 커지게 되고 데이터 중복 제거율은 감소하게 된다.

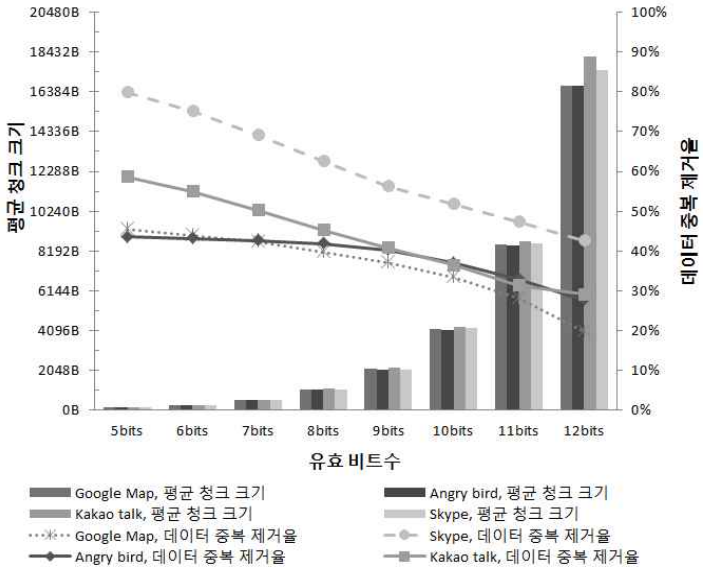

그림 3 유효 비트수의 변화에 따른 평균 청크 크기와 데이터 중복 제거율

본 연구에서는 유효 비트수에 따라 데이터 중복 제거율과 평균 청크 크기의 변화를 관찰하여 적절한 유효 비트수의 값을 찾기 위해 Google maps, Angry bird, Kakao talk, Skype에 대한 실험을 진행하였다. 실험에서 사용된 자료는 어플리케이션의 특정 버전에 데이터 중복 제거 기법을 적용한 결과이다. 그림 3에서 막대그래프는 평균 청크 크기, 꺾은선 그래프는 데이터 중복 제거율을 나타낸다. 자료를 보면 유효 비트수가 하나씩 늘어날수록 평균 청크 크기가 두 배씩 늘어나고 데이터 중복 제거율은 선형적으로 줄어들고 있다. 실험 결과를 살펴볼 때, 페이지 크기(4KB)보다 평균 청크 크기가 작은 9비트 이하의 유효 비트수는 페이지가 내부 단편화되기 쉽기 때문에 적합하지 않다. 그러나 계속 유효 비트수가 늘어날수록 데이터 중복 제거율이 높아지고 10비트의 유효 비트수에서 평균 청크 크기가 페이지 크기와 비슷한 값을 가지기 때문에 이 값이 적합하다.

5. 결론

스마트폰의 어플리케이션 업데이트가 자주 이루어지고 어플리케이션의 새로운 버전이 이전 버전에 비해 적은 데이터 변화를 가지는 것에 착안하여, 이에 적합한 데이터 중복 제거에 대해 연구하였다. 고정 길이 데이터 중복 제거 기법과 가변 길이 데이터 중복 제거 기법의 데이터 중복 제거율을 비교 했을 때 가변 길이 데이터 중복 제거 기법이 월등히 좋은 성능을 보여주었다.

가변 길이 데이터 중복 제거 기법을 대상으로 실험을 통해 데이터 중복 제거율과 평균 청크 크기를 만족시키는 유효 비트수를 결정하였다.

향후 연구로는 스마트폰 어플리케이션을 구성하는 대표적인 파일 형식인 apk의 포맷을 고려한 데이터 중복 제거를 통해 데이터 중복 제거율을 증가시키고, 가변 길이 청크를 플래시 메모리에서 효율적으로 기록하기 위한 FTL 기법을 연구할 계획이다.

참고문헌

[1] App brain, http://www.appbrain.com/

[2] http://www.samsung.com/global/microsite/galaxys2/html/specification.html

[3] Feng Chen, Tian Luo, Xiaodong Zhang, “CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives” FAST’11, 2011

[4] Michael Rabin, “Fingerprinting by random polynomials,” Center for Research in Computing Technology, Harvard Univ., Tech. Rep. TR-15-81, 1981.

[5] Dirk Meister, André Brinkmann. “dedupv1: Improving deduplication throughput using solid state drives (SSD)”. In Proc. IEEE MSST, 2010.

[6] FIPS 180-1, Secure Hash Standard, April 1995.