MADE (Minhash-Assisted Delta Compression Engine) : 델타 압축 기반의 낸드 플래시 저장장치 내구성 향상 기법

Improving the Lifetime of NAND Flash-based Storages by Min-hash Assisted Delta Compression Engine

저자
(Authors)

권혁준, 김도현, 박지성, 김지홍
Hyoukjun Kwon, Dohyun Kim, Jisung Park, Jihong Kim

출처
(Source)

정보과학회논문지 42(9), 2015.9, 1078-1089 (12 pages)
Journal of KIISE 42(9), 2015.9, 1078-1089 (12 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE06507533

APA Style

권혁준, 김도현, 박지성, 김지홍 (2015). MADE (Minhash-Assisted Delta Compression Engine) : 델타 압축 기반의 낸드 플래시 저장장치 내구성 향상 기법. 정보과학회논문지, 42(9), 1078-1089.

이용정보
(Accessed)

서울대학교
147.46.240.126
2015/12/30 15:16 (KST)

저작권 안내

DBpia에서 제공되는 모든 저작물의 저작권은 원저작자에게 있으며, 누리미디어는 각 저작물의 내용을 보증하거나 책임을 지지 않습니다.
이 자료를 원저작자와의 협의 없이 무단게재 할 경우, 저작권법 및 관련법령에 따라 민, 형사상의 책임을 질 수 있습니다.

Copyright Information

The copyright of all works provided by DBpia belongs to the original author(s). Nurimedia is not responsible for contents of each work. Nor does it guarantee the contents.
You might take civil and criminal liabilities according to copyright and other relevant laws if you publish the contents without consultation with the original author(s).

ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 42, No. 9, pp. 1078-1089, 2015. 9
http://dx.doi.org/10.5626/JOK.2015.42.9.1078

MADE (Minhash-Assisted Delta Compression Engine) : 델타 압축 기반의 낸드 플래시 저장장치 내구성 향상 기법

(Improving the Lifetime of NAND Flash-based Storages by Min-hash Assisted Delta Compression Engine)

권 혁 준 † 김 도 현 † 박 지 성 †† 김 지 홍 †††
(Hyoukjun Kwon) (Dohyun Kim) (Jisung Park) (Jihong Kim)

요 약 본 연구에서는 쓰기 데이터양 감소를 통해 낸드 플래시 기반 저장장치의 수명향상을 도모할 수 있는 MADE(Min-hash Assisted Delta-compression Engine) 모듈을 제안한다. MADE 모듈은 델타압축 기법(delta compression)을 통해 중복되는 데이터 패턴을 최소화하여 실제 낸드 플래시에 인가되는 쓰기 명령 횟수를 획기적으로 줄일 수 있을 뿐만 아니라, 중복제거기법(deduplication) 및 무손실압축기법(lossless compression)의 통합적용과 유사한 효과를 볼 수 있도록 설계되었다. 또한 델타압축기법 과정 중 필요한 참조 페이지 탐색 및 압축 기법을 최적화하여, 저장되는 데이터양을 최대한 줄이는 동시에 부가적인 오버헤드를 최소화 하였다. 시뮬레이션 결과, MADE가 적용된 플래시 변환계층(Flash Transition Layer, FTL)은 실제 낸드 플래시 칩에 저장되는 데이터를 최소 50% 줄일 수 있었으며, 순차적인 중복제거기법과 무손실압축기법을 단순 통합하여 적용한 경우에 비해 추가적으로 12%의 쓰기 데이터양을 감소시킬 수 있었다.

키워드: 낸드 플래시 저장장치, 델타압축, VCDIFF, min-hash, locality-sensitive hash, 중복제거기법, 무손실압축

Abstract In this paper, we propose the Min-hash Assisted Delta-compression Engine(MADE) to improve the lifetime of NAND flash-based storages at the device level. MADE effectively reduces the write traffic to NAND flash through the use of a novel delta compression scheme. The delta compression performance was optimized by introducing min-hash based LSH(Locality Sensitive Hash) and efficiently combining it with our delta compression method. We also developed a delta encoding technique that has functionality equivalent to deduplication and lossless compression. The results of our experiment show that MADE reduces the amount of data written on NAND flash by up to 90%, which is better than a simple combination of deduplication and lossless compression schemes by 12% on average.

Keywords: NAND flash devices, delta compression, VCDIFF, min-hash, locality sensitive hash, deduplication, lossless compression

· 이 연구를 위해 실험데이터와 시뮬레이션 툴을 제공해 주신 서울대학교 임베디드 시스템 연구실에 감사드립니다.
· 이 논문은 2014년도 서울대학교 학부생 연구지원사업의 지원을 받아 수행된 연구임
· 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2013R1A2A2A01068260)
· 이 논문은 제41회 동계학술발표회에서 'MADE(Min-hash Assisted Delta-compression Engine): 델타 압축 기반의 NAND 플래시 저장장치의 내구성 향상 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
kwonhjs@snu.ac.kr
dohyun21@snu.ac.kr
†† 비 회 원 : 서울대학교 컴퓨터공학부
jspark@davinci.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수(Seoul National Univ.)
jihong@davinci.snu.ac.kr
(Corresponding author임)

논문접수 : 2015년 1월 8일
(Received 8 January 2015)
논문수정 : 2015년 5월 14일
(Revised 14 May 2015)
심사완료 : 2015년 5월 29일
(Accepted 29 May 2015)

Copyright©2015 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지 제42권 제9호(2015. 9)

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

1. 서 론

클라우드 컴퓨팅이나 빅 데이터 시스템과 같은 컴퓨팅 환경에서는 저장장치에 대한 성능, 전력, 내구성 등 모든 방면에서 높은 수준의 사양이 요구된다. 하지만 전통적으로 사용되어온 보조저장장치인 하드디스크는 중앙처리 장치(CPU)나 주기억장치(RAM)에 비해 상대적으로 성능 개선의 속도가 늦어 시스템 전체의 성능을 제한하는 병목으로 작용하고 있으며 이를 해결하기 위해 모바일 시스템뿐만 아니라 일반적인 컴퓨팅 영역에서 낸드 플래시 기반 저장장치를 적용하기 위한 연구가 활발히 진행되고 있다.

낸드 플래시 기반의 저장장치는 뛰어난 읽기/쓰기 성능뿐만 아니라, 적은 전력소모 및 발열, 그리고 작은 폼팩터를 가지는 등 고성능 시스템의 요구사항을 다방면에서 만족하지만 고유한 물리적인 특성으로 인하여 제한적인 수명을 갖는다는 치명적인 단점을 가지고 있다. 이는 고집적화를 위한 공정 미세화와 다치화 기술(Multi-level cell)의 적용에 따른 부작용으로 심화되는 추세이다. 따라서 단위 용량 당 가격이 하락하고 있음에도 불구하고 그 확산이 제한적인 실정이며, 따라서 낸드 플래시 기반 저장장치의 수명문제를 해결하는 것은 학계/산업계에 있어 중요하고 필수적인 연구주제 중 하나이다.

초기 낸드 플래시 기반 저장장치의 수명향상을 위한 연구로는, 물리적 페이지의 할당, 가비지 컬랙션(Garbage Collection), 그리고 쓰기 평준화(Wear-leveling)과 같은 FTL(Flash Transition Layer)이 수행하는 작업들을 최적화하여 실제 낸드 플래시 칩에 인가되는 쓰기데이터 양의 확대 비율(Write Amplification Factor)을 최소화하는 간접적인 방향이 주로 제안되었다. 하지만 낸드 플래시 메모리의 내구성 악화가 심해짐에 따라 저장되는 데이터의 양을 직접적으로 줄이는 방법들 또한 제안되고 있다.

이러한 직접적인 방법으로는 기존 하드디스크 기반 저장장치 시스템에서 공간 절약의 목적으로 활용되었던 무손실압축기법(lossless compression)[1], 중복제거기법(deduplication)[2], 그리고 델타압축기법(delta compression)[3] 등이 FTL에 적용되었고, 실제 저장장치에 인가되는 쓰기 데이터양을 감소시켜 주목할 만한 내구성 향상을 이룰 수 있었다.

이 기법들은 각기 대상으로 하는 데이터의 유형과 기법의 효과 측면에서 차이점을 갖기 때문에 세 기법을 통합하여 적용한다면 저장되는 데이터의 양을 더욱 줄일 수 있을 것으로 기대된다. 하지만 이러한 기법은 아직 제안된 바 없으며 기법의 단순 통합만으로는 오버헤드가 과다해지기 때문에 이를 실시간 저장장치에 활용하기에 어려움이 있다. 그러므로 각 기법을 통합하면서 오버헤드를 최적화하는 방향의 연구가 필요하다.

이러한 필요성에 의해 이 연구에서는 오버헤드를 현실적인 수준으로 유지하면서 각 기법들의 통합적인 효과를 얻을 수 있는 방법으로 유사 페이지 탐색모듈과 델타압축모듈의 통합인 MADE 모듈을 제안한다. MADE 모듈의 델타압축모듈은 참조데이터 없이 압축 대상이 되는 데이터 내부를 자체 참조하여 압축을 수행할 수 있으며, 서로 상승효과가 있는 유사 페이지 탐색 모듈과 델타압축모듈을 조합함으로써 델타압축기법이면서 무손실압축기법과 중복제거기법의 효과를 함께 낼 수 있도록 설계되었다. 또한 유사 페이지 탐색모듈과 델타압축모듈이 수행하는 연산의 주요한 부분을 통합하여 오버헤드 또한 크게 줄일 수 있었다.

실험결과 MADE 모듈은 중복제거기법과 무손실압축기법을 순차적으로 적용한 경우보다 더 많은 쓰기 데이터양을 저감할 수 있었다. 특히 엔트로피는 높으면서 완전히 동일한 데이터보다는 유사한 데이터가 많은 특성을 갖는 트레이스(trace)에 대해 중복제거기법과 무손실압축의 단순조합에 비해 독보적인 쓰기 데이터양 저감 성능을 보여주었다. 이처럼 MADE 모듈은 상대적으로 적은 오버헤드로 기존 기법들의 단순 조합을 넘어서는 쓰기 데이터양 저감성능을 보일 수 있었으며, 기존 조합들이 잘 처리하지 못하는 유형의 데이터에 대해서도 높은 성능을 보여주었다. 따라서 MADE 모듈이 처리할 수 있는 데이터의 범위는 기존의 중복제거기법과 무손실압축기법의 범위를 상당부분 포함하며, 이들이 처리하지 못하는 새로운 영역에까지 이른다고 볼 수 있다.

2. 배경지식 및 선행연구

2.1 중복제거기법(Deduplication)

중복제거기법은 이전에 기록되어있는 데이터 블록이 현재 쓰려는 데이터 블록과 완벽히 일치할 경우 이를 새로운 플래시 저장 공간에 쓰지 않고 주소 사상(mapping)을 통해 관리함으로써 쓰기 데이터양을 줄이는 방법이다. 낸드 플래시 메모리는 in-place 업데이트가 불가능한 특성을 가지기 때문에, 보통의 낸드 플래시 기반 저장장치는 일반적으로 펌웨어의 일종인 FTL을 두어 논리 페이지 주소에 대한 실제 물리 페이지 주소를 사상하는 방법을 사용한다. 중복제거기법이 적용된 FTL에서는 쓰기 요청된 페이지와 동일한 페이지가 이미 저장되어있을 때 실제 낸드 플래시 칩에 데이터를 기록하는 대신 요청된 논리 페이지에 해당하는 주소를 이미 저장되어 있는 물리 페이지의 주소에 사상함으로써 데이터의 중복 저장을 막을 수 있다.

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

이처럼 중복제거기법은 이미 저장돼있는 데이터와 완벽히 동일한 데이터를 대상으로 하며, 조건에 맞는 데이터는 주소 사상만으로 쓰기 명령을 대체하기 때문에 대상 데이터의 범위는 좁지만 성공했을 때의 압축률은 매우 높다. 하지만 이미 저장된 데이터 중 요청된 데이터와 동일한 데이터를 찾는 동작은 단순히 기법을 적용했을 경우 방대한 비교연산을 요구하게 되고, 이를 최적화하는 것이 기법 적용에서 핵심적으로 풀어야할 문제가 된다. CaFTL[2]은, 낸드 플래시 기반 저장장치의 수명 향상을 위해 실시간 중복제거기법을 적용한 최초의 연구이며, 데이터의 직접비교를 통한 동일 데이터 검색 대신 강력한 해시함수를 이용하여 적은 에러율로 효율적인 검색을 가능하게 하였다.

이와 같은 중복제거기법은 완전하게 동일한 데이터의 중복이 많은 경우에는 충분히 큰 효과를 보이지만, 그렇지 않은 환경에서는 많은 부가적인 연산에도 불구하고 거의 효과를 보지 못할 수도 있다는 한계점을 가진다.

2.2 무손실압축(Lossless compression)

일반적으로 컴퓨팅에서 널리 사용되는 무손실압축 기법을 낸드 플래시 기반 저장장치의 수명 향상을 위해 FTL에 적용하는 연구 또한 진행된 바 있다. 무손실압축기법은 동일하거나 유사한 참조데이터 없이, 요청된 데이터 내의 중복된 패턴을 효과적으로 인코딩하여 데이터 크기를 줄이기 때문에 세 기법 중 가장 넓은 적용범위를 가지고 있다. 특히, 빅데이터 시스템이나 텍스트 기반 클라우드 컴퓨팅 시스템에서는 데이터의 엔트로피가 매우 낮기 때문에 반복된 데이터 패턴을 찾기 쉽고, 따라서 결과적으로 무손실압축만으로 높은 압축률을 달성할 수도 있다. 하지만 데이터 인코딩의 특성 상 매우 높은 컴퓨팅 능력을 요구할 수 있으며, 많은 압축 알고리즘은 압축률을 최대로 높이기 위해 많은 자원을 필요로 하는 경우가 대부분이다. 또한, 기법의 적용범위는 넓지만 성공했을 때의 효과는 나머지 두 기법들보다 높은 압축률을 보이지는 못한다는 특성이 있다. 이에 더해 낸드 플래시 기반 저장장치에의 적용에 있어, 한 페이지를 압축하여 더 작은 페이지로 압축하여도 낸드 플래시 메모리에 기록하는 페이지 크기는 미리 정해져 있으므로, 단순히 데이터의 크기를 줄이는 것만으로는 실질적인 쓰기 횟수를 줄이지 못할 수 있을 가능성이 있다. BlueZip[1]은 임베디드 시스템에 적합한 하드웨어 모듈을 설계하여 압축으로 인한 성능 오버헤드를 최소화하였으며, 한 페이지가 아닌 여러 페이지를 버퍼에 모아 압축하는 방식으로 실제 기록되는 페이지 수를 줄이도록 제안된 기법이다. 이 기법은 실제 저장되는 데이터를 최대 40% 가깝게 줄일 수 있었지만, 엔트로피가 높은 데이터의 경우 거의 이득을 얻을 수 없다는 단점이 있다.

2.3 델타압축(Delta-compression)

델타압축기법은 다른 두 기법인 중복제거기법과 무손실압축의 중간적인 위치에 존재하는 기법이다. 이 기법은 유사한 데이터를 참조데이터로 요구한다는 점과 성공했을 때 매우 큰 압축률을 보인다는 점에서는 중복제거기법과, 반복되는 패턴을 줄여 압축한다는 점에서는 무손실압축 기법과 유사점을 가진다. 또한 델타압축기법은 완벽히 동일하지는 않지만 유사한 데이터를 참조데이터로 삼아 쓰기 데이터양을 줄일 수 있기 때문에 중복제거기법보다 적용범위가 넓으며, 참조 데이터와의 차이(delta)만을 주 데이터로 하여 압축하기 때문에 성공했을 때 무손실압축보다 월등히 높은 압축률을 보여준다. 하지만, 유사한 참조 데이터를 찾기 위한 방법은 중복제거기법에서의 동일한 데이터를 찾는 것보다 복잡하며 별도의 압축 알고리즘을 필요로 하기 때문에 높은 오버헤드를 가진다. ΔFTL[3]은 같은 논리 주소로 요청되는 쓰기 명령은 업데이트에 관련됐을 가능성이 높다는 가정 하에 이미 저장되어있는 같은 논리주소의 데이터를 참조데이터로 삼아 델타 압축을 진행한다. 이 기법은 복잡한 인코딩 방식을 적용하는 대신 임베디드 시스템에 적합한 LZ계열 압축 알고리즘을 사용하며, 쓰기 요청된 데이터와 참조 데이터와의 XOR연산을 통해 엔트로피를 크게 낮춰 델타 압축의 효과를 내도록 설계되었다. ΔFTL은 델타 압축에 요구되는 참조 데이터 검색 및 압축 알고리즘 적용으로 인한 오버헤드를 줄이는 데는 성공적이었지만 데이터가 업데이트 될 때 같은 논리주소에 이를 저장하는 파일시스템과 함께여야만 잘 동작할 수 있다는 단점을 가진다. 예를 들어, 낸드 플래시 기반 저장장치를 위한 여러 log-structure 기반 파일시스템과는 기법의 이득을 보기 어려울 수 있다는 한계를 지닌다.

2.4 기법의 통합효과

쓰기 데이터양 감소를 위한 기법들은 개별적으로 낸드 플래시 기반 저장장치에의 적용을 위해 상당한 수준으로 연구되어 왔지만, 추가 개선의 가능성이 있음에도 불구하고, 세 기법을 통합하는 연구는 상대적으로 이뤄지지 않아왔다. 각 기법들은 각각 적용범위와 효과가 다르기에 통합적용 되었을 경우에 한 가지 기법을 적용했을 때보다 성능향상이 있을 것으로 기대된다. 이에 본격적인 연구에 앞서 잠재적인 성능향상의 정도를 탐색하기 위해 각 기법 각각의 성능과 이들이 통합됐을 때의 효과를 시뮬레이션을 통해 알아보는 작업을 수행하였다.

그림 1은 시스템에서 추출한 여러 데이터 워크로드에 대해, 중복제거기법과 무손실압축기법 적용을 달리하여 실제 저장장치에 저장되는 데이터양 변화를 보여주는 그래프이다. Original(baseline)은 아무런 기법을 적용하

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

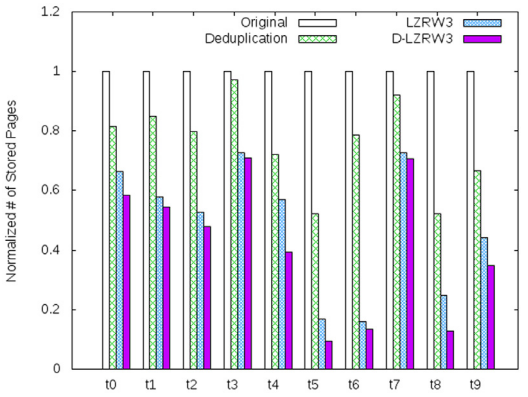

그림 1 기법을 통합시의 쓰기 데이터양 감소
Fig. 1 Reduction of write traffic on flash

지 않았을 경우의 쓰인 페이지 수를 의미하며, 모든 데이터 값들은 이 baseline 값을 기준으로 정규화하여 나타내었다. 무손실압축의 알고리즘으로는 임베디드 시스템에 적합한 LZ계열 알고리즘 중의 하나인 LZRW3[4]를 사용하였다.

그림 1에서 LZRW3으로 명명된 기법은 무손실압축(LZRW3)만을 적용한 경우이며 deduplication은 MD5를 기반으로 하는 중복제거기법만을 적용한 경우이다. 그리고 D-LZRW3은 중복제거기법을 적용한 후에 동일 데이터를 찾지 못했을 경우 LZRW3를 적용하여 무손실압축을 행한 경우를 의미한다. 시뮬레이션 결과, 하나의 기법을 적용한 경우보다 두 개의 기법을 동시에 적용한 경우 더욱 더 많은 쓰기 데이터양 감소를 얻을 수 있다는 것을 알 수 있으며, 여기에 델타압축까지 통합된다면, 더 많은 쓰기 데이터양 감소를 보일 것으로 기대하였다.

하지만 세 가지 기법을 통합하여 적용하려면, 요구되는 자원의 양과 오버헤드가 크게 증가하게 된다. 우선 세 가지 기법을 순차적으로 적용할 경우에는 각 기법의 시간적인 오버헤드가 합쳐지게 된다. 최악의 경우 중복제거기법을 위해 참조 데이터를 해시함수를 통해 검색했지만 찾지 못하고, 이어서 델타 압축을 위해 참조 데이터를 검색했으나 유사한 데이터를 찾지 못하고, 결과적으로 무손실압축을 적용하는 경우가 발생 할 수 있다. 이 경우 최종적인 저장용량 감소효과 대비 쓰기속도의 저하가 커지게 되어 높은 성능을 보장해야하는 환경에의 적용이 부적합할 수 있다. 이와 별개로 필요자원도 문제가 되는데, 세 기법을 단순히 조합한 경우 플래시 저장 공간 외에도 각 기법에 필요한 임시 저장용량도 모두 개별적으로 요구되며 중복제거기법에서 요구하는 동일한 참조 데이터 검색 방법, 델타 압축을 위한 유사한 참조 데이터 검색 방법, 그리고 델타 압축의 압축 알고리즘, 마지막으로 무손실압축 알고리즘까지 지나치게 많은 기법 적용으로 인한 오버헤드가 발생하게 된다. 따라서 이러한 기법들의 단순한 결합은 쓰기 데이터양을 효과적으로 줄일 수 있지만 사용할 수 있는 시간적, 공간적 자원이 제한적인 내장형 시스템 환경에 바로 적용하는 데 어려움이 있다. 따라서 각 쓰기 데이터양 기법들을 통합하기 위해서는, 발생하는 오버헤드를 효과적으로 최적화하는 것이 필수적이다.

백업용 네트워크 저장장치 분야에서는 저장용량을 효율적으로 사용하기 위해 앞서 언급한 기법을 통합하여 쓰기 데이터를 줄이려는 시도가 있었다. Shilane은 백업용 저장장치에 중복제거기법과 델타압축을 순차적으로 적용한 기법[5]를 제안하였다. 이 기법은 쓰기 데이터를 상당부분 줄이는 데에 성공하였으나, 오버헤드에 대한 고려가 충분치 않아 내장형 시스템 환경에서 사용하기에는 무리가 따른다.

3. MADE의 설계

3.1 MADE의 구성

그림 2는 본 연구에서 제안하고자 하는 델타 압축 모듈인 MADE 모듈의 동작 알고리즘을 도식화한 것이다. MADE 모듈은 델타압축 과정을 최적화하여 중복제거기법과 무손실압축의 적용 범위 및 효과의 일부를 가지도록 설계되었으며, 실시간 저장장치에 활용하기 위하여 기법 적용에 따른 오버헤드를 최소화하는데 중점을 두어 개발되었다.

MADE 모듈에서는, 페이지의 쓰기요청이 들어오게 되면 우선적으로 해당 페이지의 유사도 힌트(signature)를 Rabin-Karp fingerprint를 이용한 min-hash 기법을 이용하여 계산한다. 계산되는 유사도 힌트는 참조 데이터를 찾기 위한 값으로 사용되며, 데이터의 패턴이 반영되어 유사도힌트가 완벽히 일치하지 않더라도 유사한 페이지를 찾을 수 있도록 설계되었다. 이렇게 계산된 유사도 힌트는 기존에 저장된 다른 페이지들의 유사도 힌트 값들과의 일치여부를 이용하여 유사한 데이터를 찾아낸다. 그리고 해당 유사도 힌트를 가지는 페이지를 읽어내 참조 페이지로 사용한다. 참조로 삼을 만큼 유사한 페이지를 찾지 못했을 경우, 데이터가 모두 '0'인 페이지를 참조 데이터로 사용하여 자체참조를 통한 압축이 이뤄질 수 있도록 한다. 설정된 참조 페이지를 VCDIFF[6] 기반의 개선된 델타압축 알고리즘에 적용하여 대상 페이지를 압축한다. 유사한 참조 페이지를 찾지 못한 경우에는 해당 데이터가 앞으로 다른 페이지의 참조 데이터로서 사용될 수 있기 때문에 유사도 힌트를 전용 저장소(fingerprint table)에 저장한다. 유사한 데이터를 찾은 경우에는 다단계의 참조가 발생하지 않도록 저장하지

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

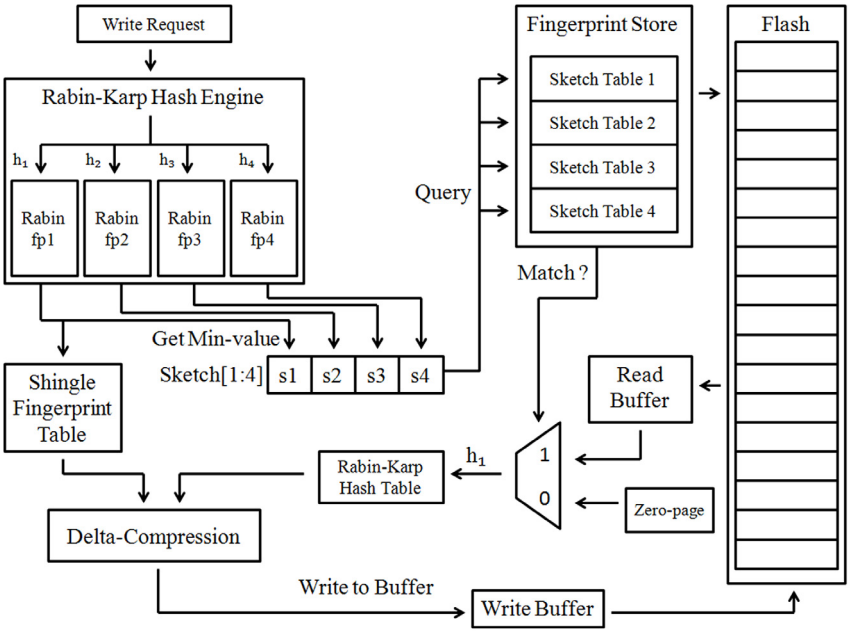

그림 2 MADE 모듈의 구조

Fig. 2 Overview of the MADE module

않는다. MADE 모듈이 갖는 특징과 장점은 아래와 같이 요약할 수 있다.

• 효율적인 유사 페이지 탐색

: MADE는 유사한 페이지를 탐색하기 위해 페이지 전체를 비교하는 것이 아니라 해시함수를 통해 계산된 유사도 힌트(signature)를 사용한다. 유사도 힌트는 20~40 바이트 정도의 작은 크기를 갖기 때문에 같은 크기의 메모리를 사용할 경우 페이지의 내부 데이터를 바이트 단위로 비교하는 기법에 비해 저장하고 탐색할 수 있는 페이지의 범위가 넓고 빠른 검색이 가능하다는 장점이 있다. 또한 유사도 힌트 저장소(fingerprint table)는 LRU 정책으로 관리되어 시간적인 지역성(temporal locality)이 반영되도록 하여 적은 추가 메모리 하에서도 더욱 효율적으로 유사 페이지를 찾을 수 있도록 하였다.

• 중복제거기법, 델타 압축, 무손실압축의 통합 효과

: MADE 모듈은 기본적 동작은 델타 압축을 기반으로 하지만, 완전히 동일한 페이지가 들어올 경우에 이를 수 바이트 수준으로 압축하여 마치 중복제거기법을 적용한 것과 같은 효과를 보인다. 뿐만 아니라 데이터의 일부분만 다르거나 동일한 데이터가 일정한 오프셋만큼 밀려있는 경우에도 중복제거기법만큼의 압축 효율을 보인다는 강점이 있다.

비슷한 페이지를 찾아내지 못했을 경우에도 데이터가 모두 '0'인 페이지를 참조데이터로 해서 자체참조를 통한 압축을 시행하도록 설계되어있어 무손실압축과 비슷한 효과를 낼 수 있도록 설계되었다. 이는 적용된 델타 압축 알고리즘인 VCDIFF가 쓰기 요청된 페이지의 데이터만을 이용해서도 작동할 수 있다는 특성에 기인한다. 이 경우에는 일반적인 무손실압축과 비슷하게 요청된 페이지 내부에 있는 중복데이터를 제거하여 압축효과를 얻는다.

• 오버헤드의 최적화

: 유사도 힌트(signature)를 만들기 위한 min-hash를 계산하는 과정과 델타압축의 데이터 인코딩 과정은 모두 내부적으로 해시함수 계산을 필요로 한다. 따라서 두 모듈에 사용될 해시함수를 동일한 설정 값의 Rabin-Karp fingerprint로 하여 한 번의 데이터 탐색으로 min-hash 및 데이터 인코딩 과정에서 사용한 해시 값들을 공유할 수 있도록 하였다. 결과적으로, 압축과정에서 발생하는 추가적인 데이터 스캔을 막음으로써, 시간적인 오버헤드를 크게 감소시킬 수 있었다.

3.2 유사한 페이지 탐색기법

MADE 모듈에서는, 델타압축(delta-compression) 시 필요한 유사 데이터를 갖는 참조 페이지 탐색을 위해 Rabin-Karp fingerprint 기반의 min-hash를 사용한다. MADE에 사용된 Min-hash 기법은 비슷한 데이터를 가질수록 해시 값이 일치할 확률이 높아지는 LSH(Locality Sensitive Hash)의 특성을 이용한 것으로, 참조

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

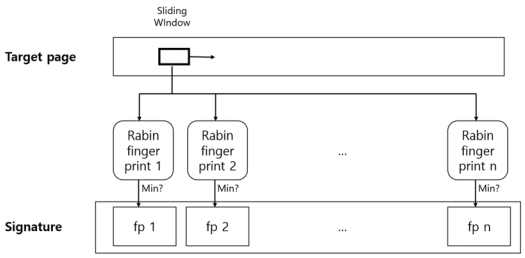

그림 3 유사도 힌트(signature) 계산
Fig. 3 Page signature calculation

페이지로 선정할 데이터의 유사도 기준을 세밀하게 조절할 수 있다는 특징을 갖는다. 그림 3은 요청된 페이지에 대해 min-hash를 이용하여 유사도 힌트(signature)를 구하는 예를 보여준다. 우선 요청된 페이지 상에 sliding window를 끝까지 이동하면서 window 내부의 Rabin fingerprint를 구한다. Rabin fingerprint는 서로 다른 n개의 해시 함수를 이용하여 계산되는데, 각각의 해시함수별로 계산된 Rabin fingerprint 중 최솟값을 취한다. 이렇게 계산된 n개의 min-hash fingerprint fp_n 중 적어도 한 개가 일치할 경우 유사한 페이지로 판정한다.

결과 값이 고루 분포되는 특성을 갖는 임의의 해시함수(evenly distributed hash function)를 이용하면 두 데이터로부터 계산한 min-hash가 일치할 확률은 Jaccard 유사도(similarity)와 같다는 사실이 알려져 있다[7].

Jaccard 유사도는 비교하는 두 데이터에 중복되는 부분의 비율에 관련된 척도로, 이 값이 커질수록 두 데이터 사이에 중복되는 패턴의 비율이 높다는 것을 의미한다. 따라서 이러한 패턴을 활용하는 델타압축 기법인 VCDIFF의 알고리즘은 Jaccard 유사도가 높은 데이터를 참조데이터로 제공할 경우 더 높은 압축률을 보일 것이다.

앞서 서술한 것처럼 여러 개(n)의 해시함수를 이용하여 min-hash를 계산할 경우 n개의 min-hash 중 적어도 하나가 일치할 확률 p는 다음과 같이 주어진다.

$$p = 1 - s^n \quad (1)$$

위 식을 이용하면 어떤 데이터를 유사하다고 판정할 확률(p)과 비교하는 두 데이터 간의 Jaccard 유사도(s) 사이의 관계를 사용하는 해시함수의 개수(n)을 이용하여 조절할 수 있다. n이 커질수록 낮은 Jaccard 유사도에서 비슷하다고 판정하는 경향이 생기며 그림 4와 같이 유사하다고 판정하는 기준을 완화할 수 있다.

MADE에서는 위의 기법보다 한 단계 발전된 형태를 사용하여 유사도와 비슷하다고 판정할 확률을 더욱 미세하게 조정할 수 있도록 하였다[7]. 그림 5에 묘사된 바와 같이 앞서 사용한 여러 개의 min-hash값을 일정한 개수로 나눠서 그룹을 만들고 이들 그룹 중 적어도 하나가 일치하는지 확인하는 방식을 채택한다. 만약 두 개

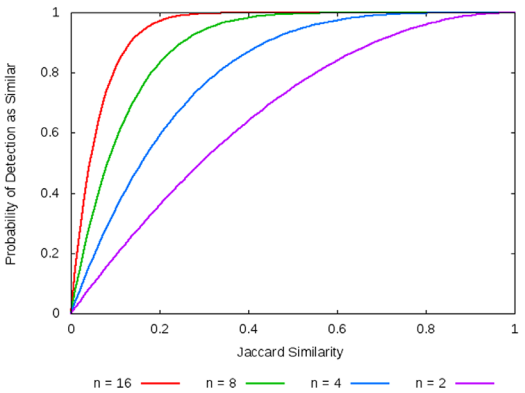

그림 4 n 값에 따른 LSH 특성 변화
Fig. 4 LSH characteristic curve with respect to number of signatures (n)

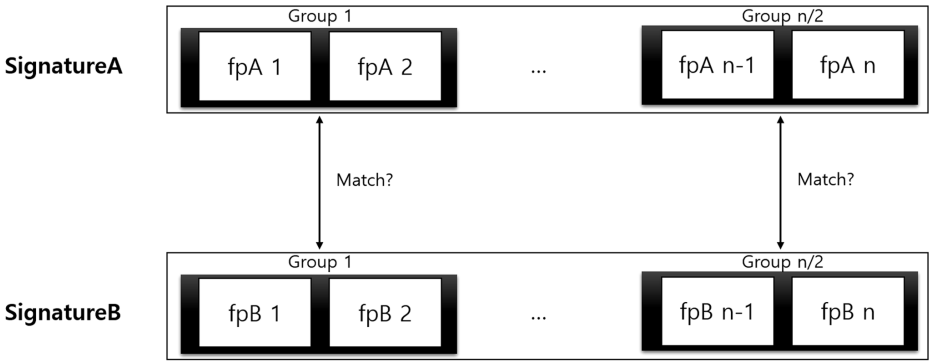

그림 5 유사도 힌트(Signature) 비교방식
Fig. 5 Comparison method for page signatures

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

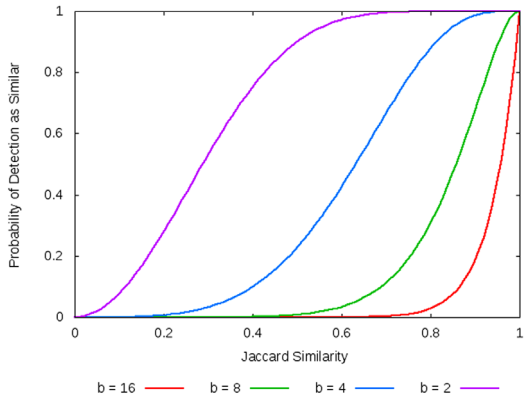

그림 6 b값에 따른 LSH의 특성 변화

Fig. 6 LSH characteristic curve with respect to number of bands (b)

의 페이지에 대해 n개의 min-hash 함수를 계산하고, 그 결과 값들을 순서대로 b개씩 묶어서 group으로 만들 때, 적어도 하나의 그룹이 완전히 일치할 확률은 다음과 같다.

$$p = 1-(1-(1-s)^b)^{n/b} \quad (2)$$

n을 고정하고 b의 값을 조정하면 앞의 경우와는 다르게 비슷하다고 판정할 확률이 급격히 증가하는 구간을 높은 유사도에서 나타나도록 할 수 있다. 즉, 그림 6과 같이 유사하다고 판정하는 기준 값을 엄격하게 할 수 있다. 이 연구에서는 델타압축 모듈과 연계한 실험을 통하여 얻은 압축 성능과 메모리 오버헤드를 고려하여 16 바이트 크기를 갖도록 n=4, b=2인 세팅을 사용하였다.

3.3 델타압축 기법의 최적화

VCDIFF는 복사기반 델타 압축(copy-based delta compression)의 한 종류로, 두 데이터 내에 존재하는 동일한 부속 문자열(substring)들을 이용하여 압축을 수행하는 기법이다[6]. 이 기법은 반복되는 부속 문자열을 그 부속문자열이 나타나는 위치와 반복되는 횟수로 기록하는 방식(run-length encoding)과 일정 길이 이상으로 나타나는 같은 부속문자열을 해당 부분 문자열이 나타나는 위치와 그 길이로 기록하는 방식의 조합이다. 따라서 VCDIFF는 동일한 부속 문자열이 많을수록 좋은 압축 성능을 발휘하며, MADE의 유사 페이지 판정 기준으로 삼는 Jaccard 유사도가 높을수록 좋은 성능을 발휘할 수 있는 기법이다. 뿐만 아니라 참조데이터가 없더라도 압축 대상 데이터 내부에 존재하는 부속 문자열들의 패턴만으로도 압축이 가능하므로, 델타압축을 알고리즘이지만 별도의 알고리즘 적용 없이도 무손실압축의 효과를 함께 볼 수 있는 기법이다. 하지만 내장형 시스템에의 적용을 염두하고 설계된 기법이 아니기 때문에 계산하는 데에 필요한 오버헤드가 크며, 일반적인 파일에 비해 상대적으로 작은 용량인 페이지를 처리하는 데에는 불필요한 요소를 포함하고 있다. 따라서 MADE 모듈에서는 VCDIFF의 인코딩 방식을 본 연구가 대상으로 하고 있는 제한적인 환경에 적합하도록 최적화 하여 사용하였다.

표준 VCDIFF 인코딩에서는 COPY나 RUN에서 사용할 반복횟수와 문자열 길이의 최댓값을 18로 제한하고 있다[8]. 표준에서 채택하고 있는 이러한 방식은 반복횟수와 문자열의 길이를 따로 기록할 필요가 없이 미리 정의된 명령어코드만을 사용하여 더 작은 크기로 압축할 수 있다는 장점이 있지만 대상과 참조 두 페이지에 동일한 데이터가 길게 나타날 경우 하나의 COPY명령어로 한 번에 표현할 수도 있는 데이터를 여러 번 쪼개서 표현해야한다는 문제점이 있다. 일반적으로 저장장치에 쓰기 요청을 통해 들어오는 데이터의 상당 부분이

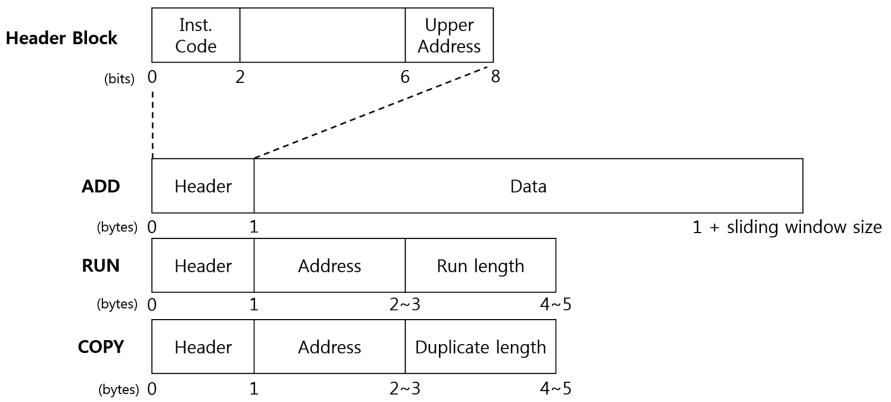

그림 7 MADE에 적용된 델타압축 인코딩 방식

Fig. 7 MADE delta encoding

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

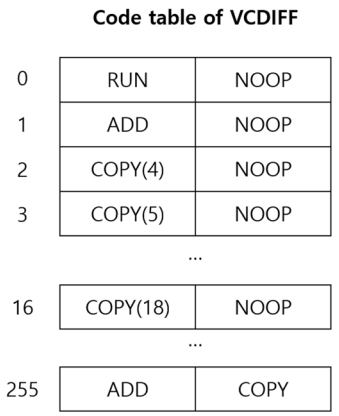

그림 8 VCDIFF의 코드 테이블
Fig. 8 VCDIFF code table

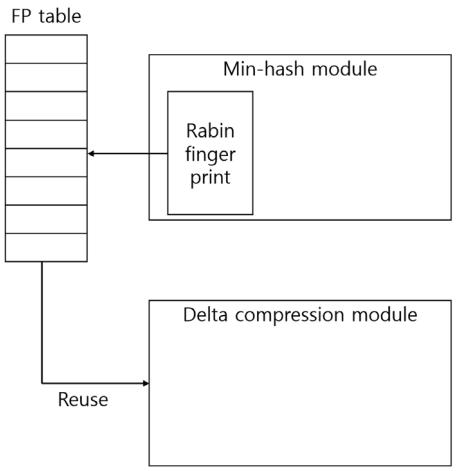

그림 9 중복된 계산 통합을 이용한 오버헤드 최적화
Fig. 9 Overhead optimization by sharing the calculated fingerprints between two modules

동일한 데이터 패턴을 갖고 있으므로[3] 위와 같이 제한된 명령어를 사용하는 방식은 비효율적으로 작동할 가능성이 높다. 따라서 MADE에서는 길이가 긴 데이터에도 더 효율적으로 대처할 수 있도록 인코딩 방식을 개선하였다.

MADE에서 사용한 VCDIFF 인코딩의 첫 블록(첫 번째 바이트)에는 압축명령어의 종류와 참조한 데이터의 주소 값 상위 2비트가 기록되어있다. 주소 값 상위 2비트의 첫 번째 비트는 외부 참조 데이터 사용 여부를 나타낸다. 두 번째 블록부터는 명령어에 따라 서로 다른 의미를 갖는 데이터가 저장된다. ADD의 경우 두 번째 블록은 sliding window의 크기만큼 해당 위치에 존재하는 문자열을 기록한 값이다. 따라서 두 번째 블록의 크기는 sliding window의 길이와 동일하다. MADE에서는 실험을 통하여 64바이트의 크기를 사용할 때 가장 높은 효율을 갖는 것을 확인하여 그 값을 사용하였다. COPY와 RUN의 경우 두 번째 블록에는 참조한 데이터의 주소가 기록된다. 상위 2비트는 첫 번째 블록에 기록되었으므로 나머지 값을 기록한다. 주소를 기록할 때에는 가변길이(variable length)기록 방식을 사용한다. 흔히 사용하는 페이지의 크기는 4KB이지만 최근 페이지의 크기가 증가하고 있기 때문에 사용할 수 있는 페이지 크기의 최댓값을 32KB로 두었다. 이 세팅 하에서 VCDIFF 표준에 정의된 가변길이 기록 방식을 사용하면 주소는 1바이트나 2바이트의 길이로 표현이 가능하다. COPY와 RUN의 세 번째 블록에는 참조한 데이터의 길이와 Run-length encoding에서 패턴이 반복된 횟수가 각각 기록된다. 여기서도 역시 가변 길이 기록 방식을 사용하며, 위와 동일한 가정 하에 1바이트 내지 2바이트로 표현이 가능하다.

이와 같은 인코딩 방식을 사용하면 압축하는 두 페이지가 동일한 경우 COPY명령어 하나로 표현이 가능하다. 이 경우 페이지 하나가 4바이트로 표현이 되기 때문에 실질적으로 중복제거기법과 동등한 효과를 보일 수 있다.

3.4 오버헤드 최적화

MADE의 델타압축은 중복되는 데이터 탐색의 효율성을 위해서 데이터 전체에 대해 sliding window를 이동해가면서 계산한 Rabin fingerprint 값을 이용하도록 되어있다. 유사한 페이지 탐색 모듈에서도 동일한 연산을 수행하며 최솟값을 찾아내는 min-hash를 계산하기 때문에 효율적인 연산을 위해 유사한 페이지 탐색 모듈에서 계산한 Rabin finger print값들을 델타 압축에서 활용할 수 있도록 하였다

델타 압축에서 압축 대상이 되는 페이지는 가장 앞에서부터 한 칸씩 이동하며 Rabin fingerprint 값을 구해야하기 때문에 min-hash를 구하는 과정 중 계산된 Rabin fingerprint 값을 테이블에 순차적으로 저장하고 이를

LBA	PPA	Offset	Length	Reference page number
32 bit	32 bit	16 bit	16 bit	32 bit

그림 10 Mapping table entry 구조
Fig. 10 Mapping table entry design

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

델타 압축 모듈에서 활용하였다.

참조 페이지의 경우 델타 압축 전에 동일한 해시함수를 이용하여 페이지 전체의 Rabin finger print값이 계산된다. 이때 압축 대상 페이지와는 다르게 계산한 Rabin finger print값을 해시 테이블에 저장한다. 해시 테이블을 사용하는 것으로 압축 과정 중 발생하는 수많은 검색 작업의 효율성을 높일 수 있었다. 해시 테이블은 open-addressing 방식을 택하였으며 32KB의 공간을 사용한다. 이는 open-addressing 해시 테이블이 최적의 검색 성능을 가지기 위해서는 저장된 데이터의 크기만큼의 빈 공간을 유지해야 하며, 하나의 페이지를 처리할 때에 약 16 KB의 용량이 소모되기 때문이다.

전체적인 구조는 압축 대상 페이지의 Rabin fingerprint값이 저장된 배열의 가장 처음부터 순차적으로 진행하면서 배열의 현재 위치에 저장되어있는 값과 같은 값을 해시 테이블에서 탐색하는 것으로 되어있다. 이처럼 요구되는 연산에 맞추어 알맞은 자료구조를 채택함으로써 효율적인 모듈이 되도록 하였다.

4. FTL 설계

4.1 FTL 설계 개요

그림 2는 MADE 모듈이 적용된 FTL의 대략적인 구조를 나타낸다. 쓰기 데이터가 입력되면 우선 네 가지의 Rabin 해시함수를 이용하여 Rabin-Karp hash engine에서 해시 값을 계산한다. 네 가지의 해시 값들 중 하나는 이후 델타압축의 인코딩 과정에서 재사용하기 위해 Rabin fingerprint table에 저장된다. 이 네 가지의 fingerprint 중에서 각각에 대해 최솟값을 조합하여 유사도 힌트(signature)를 구성한다. 유사도 힌트는 네 바이트 크기의 Rabin fingerprint 네 개로 구성되어 있다. Fingerprint store는 네 가지의 해시 값들에 해당하는 유사도 힌트 저장소(signature table) 네 개로 이루어져 있으며, 각각의 해시함수로 추출해낸 fingerprint를 이용하여 fingerprint store에서 탐색을 시도한다. 탐색에 성공했을 경우 플래시에서 해당 물리페이지를 읽기 버퍼에 로드한 뒤 이를 읽어와 Rabin fingerprint를 구해 Rabin-Karp hash table에 저장한다. 그리고 이를 참조 데이터로 하여 델타압축한다. 만약 탐색에 실패했을 경우 전체 데이터가 '0'으로 채워진 zero page를 참조 데이터로 삼아 델타압축한다.

데이터를 읽어야 할 경우에는 우선 page mapping table의 정보를 활용하여 참조 페이지의 주소를 알아낸 뒤, 해당 페이지를 읽어온다. 그리고 읽어온 페이지를 이용하여 델타 인코딩되어 있는 명령어를 순차적으로 실행하는 방식으로 읽기 요청이 들어온 데이터를 디코딩하여 읽기 요청에 대한 처리를 마무리 한다.

4.2 Mapping Table 설계

위의 FTL에서 사용하는 mapping table의 경우 아래의 그림 11과 같은 구조를 가진다. Mapping table은 32비트의 논리블록주소(Logical Block Address: LBA), 해당 데이터블록이 포함되어있는 32 비트의 물리페이지 주소(Physical Page Address: PPA), 해당 블록이 페이지의 어느 부분에 들어있는지 나타내는 16비트의 offset과 압축 결과의 길이를 나타내는 16비트의 length 비트, 그리고 참조한 페이지의 물리번호인 32 비트의 참조 페이지 번호(reference page number)로 이루어져 있다. 이중 참조 페이지 번호는 데이터가 델타압축을 이용하여 압축됐을 때만 기록하게 되며, 데이터의 델타압축 여부를 판명하는 flag로써도 사용된다.

낸드 플래시 메모리는 쓰기 명령의 단위가 페이지로 미리 정의돼있으므로, 한 페이지를 압축하여 쓰더라도 실질적 쓰기 명령 횟수는 줄어들지 않는다. 따라서 MADE 모듈을 적용하여 실제 낸드 플래시 칩에 인가되는 쓰기 명령 횟수를 줄이기 위해서는 쓰기 버퍼를 두어 여러 논리 페이지들의 압축 결과가 한 페이지 이상이 됐을 때 비로소 낸드 플래시 칩에 기록하는 방법과 같은 기법의 적용이 필수적이다. 이 경우, 데이터의 단편화(fragmentation)을 막으려면, 한 논리 페이지의 압축결과가 두 물리페이지에 걸쳐 저장되는 경우가 발생하며, FTL에서는 해당 경우의 페이지 주소 사상을 적절히 처리할 수 있어야한다. 우리가 개발한 MADE 모듈이 적용된 FTL에서는 해당 정보를 낸드 플래시 메모리 페이지의 OOB (Out Of Band) 영역에 기록하도록 한다. 예를 들어, 논리 페이지의 압축결과가 두 물리 페이지에 걸쳐 저장된 경우, offset + length는 한 페이지

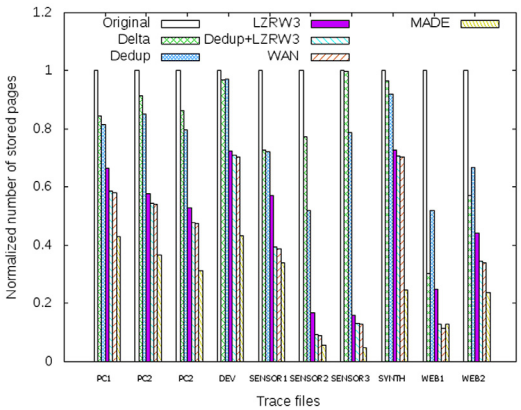

그림 11 MADE와 기존기법들이 시뮬레이션에서 기록한 쓰기 데이터양

Fig. 11 Normalized number of stored pages: comparing previous methods and MADE

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

의 크기를 넘어서므로, 두 물리 페이지에 걸쳐 저장됐다는 여부를 알 수 있고, 해당 PPA의 페이지를 읽을 때 OOB를 함께 읽음으로써, 압축결과의 나머지 부분이 어디에 저장돼있는지를 알 수 있다. 또한, 낸드 플래시의 OOB영역은 ECC (Error Correction Codes)를 위해 사용되는 것이 일반적이나, 네 바이트 수준의 여유용량은 충분히 가지고 있으므로, 기법의 실질적인 구현에는 문제가 되지 않는다.

4.3 읽기 오버헤드 최적화: 읽기 버퍼

MADE는 델타압축에 기반을 둔 기법이기 때문에, 참조 데이터를 불러들이는 추가적인 읽기 작업이 반복적으로 발생한다는 특성이 있다. 읽기 작업을 수행할 때 발생하는 최악의 경우로는 다음과 같은 예를 생각해볼 수 있다.

어떤 페이지에 대한 읽기 요청이 들어올 경우, FTL은 이를 수행하기 위해 page table을 탐색한다. 하지만 이 페이지가 두 페이지에 걸쳐 존재할 경우, 두 번 플래시로부터 페이지를 읽어 압축된 데이터를 불러와야한다. 이 후, 이들 페이지를 조합한 압축된 데이터를 얻게 된다. 델타압축을 사용했기 때문에 압축을 풀기 위해서는 추가적으로 참조된 페이지를 불러들어야 한다. 만약 참조 페이지 역시 두 물리 페이지에 걸쳐 존재할 경우 다시 두 번 페이지를 불러들이고 압축을 해제하는 과정이 필요하다. 따라서 한 번의 읽기 요청을 처리하기 위해, 최악의 경우 네 번 플래시로부터 페이지를 읽어야하는 경우가 생긴다. 마찬가지로, 쓰기 작업의 경우에도 참조 데이터를 사용하기 위해 추가적으로 여러 번 읽기 연산을 수행할 가능성이 있다.

이러한 추가적 반복 읽기는 일반적인 상황(common case)에 반복적으로 발생하므로, 이 과정을 최적화 하지 않으면 저장장치의 성능에 심각한 영향을 미칠 수 있다. 읽기로 인한 오버헤드를 줄이기 위해 우리가 설계한 MADE 모듈이 적용된 FTL에서는 참조 데이터만을 위한 읽기 버퍼가 설계에 반영하였다. 이는 참조 데이터로 사용되는 페이지가 일정한 지역성을 띠고 존재한다는 관찰결과를 바탕으로 접근한 방법으로, 자주 쓰이는 LBA에 해당하는 페이지들을 버퍼에서 관리하여 서비스함으로써 추가적인 낸드 플래시 읽기 명령을 줄이는 방식이다. 이 읽기버퍼 내부의 페이지들은 LRU 방식으로 관리되며, 페이지 테이블에서 LBA를 탐색할 때 이 buffer 내부를 먼저 확인해봄으로써 불필요한 플래시로부터 읽기 작업을 줄일 수 있다.

4.4 유사도 힌트 저장소(Signature table) 구조

MADE의 특징적인 구조인 유사성을 찾아내기 위한 fingerprint store는 각각의 최소 Rabin-Karp fingerprint 값을 저장하는 네 개의 유사도 힌트 저장소(signature table)로 이루어져 있다. 이 유사도 힌트 저장소의 경우에는 네 바이트의 유사도 힌트 값과 네 바이트의 논리 블록 주소로 이루어진 여덟 바이트의 entry로 이루어져 있다. 유사성 탐색을 할 때에는 유사도 힌트 저장소에서 현재 쓰려는 데이터블록의 min-hash 유사도 힌트 값을 각 entry의 유사도 힌트 값과 비교하여 탐색하게 되며, 각 entry가 가지고 있는 논리 블록 주소로 페이지 테이블에서 탐색한다. 이 논리 블록 주소에 해당하는 데이터를 플래시로 읽어 들여 참조 데이터로 삼기 위함이다. 물리페이지를 불러오지 않고 논리 블록주소로 불러들이는 이유는 우선 각 페이지는 다양한 페이지들을 저장하고 있고, 이 논리 블록주소에 들어있는 데이터를 압축해제하면 4 KB의 페이지가 되기 때문이다.

유사도 힌트의 크기는 물리 페이지의 크기에 피해 매우 작지만, 저장장치의 공간이 넓어지면서 모든 유사도 힌트를 저장하기에는 현실적으로 어려움이 따른다. 8-KB의 페이지 크기를 갖는 256-GB의 낸드 플래시 기반 저장장치의 경우, 유사도 힌트 저장을 위해서 2-GB 정도의 저장 공간을 필요로 하게 되는데, 저장장치에 적용되는 RAM이 갈수록 늘고 있는 추세를 고려하더라도 지나치게 많은 양이다. 따라서 유사도 힌트 저장소(signature table) 또한 크기를 제한하여 관리할 필요성이 있고, 본 연구에서 설계한 MADE 모듈이 적용된 FTL에서는 sketch table을 제한된 크기로 LRU방식으로 운용한다. 앞 절에서 언급했듯, 특정 참조 데이터가 높은 비율로 압축에 관여하므로, LRU 방식의 sketch table은 자주 참조되는 데이터의 sketch를 효율적으로 관리할 수 있다.

5. 실험결과 및 토의

5.1 쓰기 데이터양 감소

실험으로는 FTL작동환경을 모사한 프로그램과 실제적인 컴퓨터 사용 환경에서 수집된 I/O 트레이스(trace)를 이용한 시뮬레이션을 수행하였다.

실험에는 총 10개의 I/O 트레이스가 사용되었다. I/O 트레이스로는 일반적인 PC환경에서 추출한 데이터인 PC, 반도체 생산 공정에서 나오는 신호처리 데이터인 SENSOR1-3, 하드웨어 합성과정에서 발생하는 데이터 SYNTH, 그리고 웹브라우저 사용 시에 발생하는 데이터들을 취합한 WEB 트레이스, 리눅스 운영체제 설치 시 발생하는 DEV 트레이스가 사용되었다. 개별적인 특성은 표 1에 나타냈다.

MADE에 대한 시뮬레이션을 평가하기 위해 사용한 대조 기법들로는 MD5해시에 기반을 두어 작동하는 중복제거기법(dedup), LZ77 무손실압축 기법의 변형인 LZRW3 무손실압축[4]를 이용한 기법(LZRW3), 그리고 앞서 언급한 중복제거기법(dedup)과 무손실압축기법

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

표 1 각 트레이스 별 I/O 특성 비교
Table 1 I/O characteristics of ten I/O traces

Trace	총 I/O 요청	총 I/O 요청 중		총 쓰기 I/O 요청 중	
		읽기 요청 비율(%)	쓰기 요청 비율(%)	중복 LBA 대상 쓰기 요청 비율(%)	중복 데이터 쓰기 요청 비율(%)
PC1	207827	23.9	76.1	29.2	19.2
PC2	492782	0.0	100.0	9.4	16.1
PC3	385795	9.9	90.1	15.9	22.6
DEV	452149	12.5	87.5	23.8	72.6
SENSOR1	27128	0.0	100.0	29.5	27.8
SENSOR2	26186	0.0	100.0	27.5	47.8
SENSOR3	84102	73.5	26.5	0.2	21.2
SYNTH	168036	5.1	94.9	6.1	8.1
WEB1	234055	0.0	100.0	83.8	47.3
WEB2	221800	0.0	100.0	64.4	33.4

(LZRW3)을 통합하여 순차적으로 실행한 복합기법(D-LZRW3, 그리고 백업 저장장치에서 중복제거와 델타압축을 연속적으로 적용한 기법인 Shilane의 기법(WAN)[5]가 있다. MADE는 8-KB의 fingerprint store을 사용하였으며, 중복제거기법(dedup)은 20KB의 해시 저장소를 활용하였다.

그림 11은 앞서 서술한 실험 환경에서 각 기법별로 실제 낸드 플래시에 인가된 페이지 쓰기 데이터양을 아무 기법도 적용하지 않았을 때(Original; Baseline)와 비교하여 상대적으로 나타낸 것이다.

실험 결과에서 볼 수 있듯이, MADE는 쓰기 데이터 저감 측면에서 대부분의 I/O 트레이스에 대해 중복제거 기법과 무손실압축, 그리고 그 둘의 조합을 능가하는 성능을 보여주었다. 특히 하드웨어 합성과정 중 발생한 데이터 SYNTH의 경우 엔트로피가 높으면서도 거의 동일하지만 일부분만 다른 페이지가 많이 존재한다는 특성을 가진다. 표 1에서 보면, SYNTH 는 다른 여타 트레이스들에 비해 적은 동일 LBA 쓰기 요청 비중(6.1%)과 중복 데이터 비중(8.1%)을 가지는데, 이러한 트레이스 고유의 특성에 의해 기존의 기법들인 중복제거기법과 단순 델타 압축 기반의 기법이 낮은 성능을 보였다. 이에 비해 MADE의 경우 능동적으로 유사도가 높은 페이지를 탐색하여 델타압축을 수행하여 주목할 만한 쓰기 데이터양 감소효과(90%)를 얻을 수 있었다.

5.2 참조되는 페이지의 지역성

앞서 4.3장에서 논의된 것과 같이 참조할 데이터를 읽어 들여오는 것이 추가적인 오버헤드가 되는데, 이를 최적화하기 위해 MADE의 FTL 설계에는 읽기 버퍼가 포함되어있다. 이러한 최적화의 효과를 검증하기 위해 특정한 값을 가지는 참조 데이터가 최초에 플래시에 기록된 이후, 각각의 데이터가 총 몇 번 참조되는지를 기록하였다.

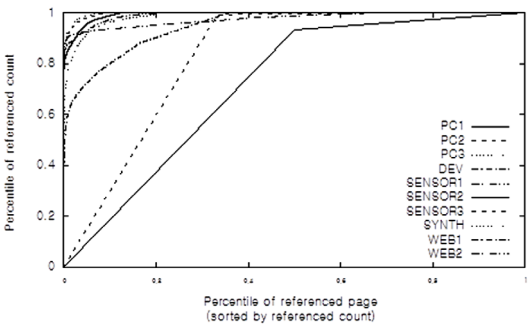

그림 12 참조된 페이지 횟수의 지역성
Fig. 12 Locality of the referenced pages

그림 12는 델타압축을 위한 데이터 참조가 소수의 페이지들에 집중되는 경향을 보여준다. 가로축은 전체 페이지를 참조된 횟수가 많은 순으로 정렬했을 때의 순위를 백분위수 p로 나타낸 것이다. 세로축은 참조횟수가 가장 많았던 페이지부터 가로축 백분위수 값에 해당하는 페이지까지의 총 참조횟수를 전체 참소횟수에 대한 비율로 나타낸 것이다.

그림 12에 나타난 경향을 관찰하면, 참조로 사용된 페이지들 중 참조횟수 상위 10%의 페이지들이 전체 참조횟수의 80% 이상을 차지하는 것을 알 수 있다. 특히, 데이터의 내부적인 유사성이 높은 SENSOR2와 SENSOR3 트레이스의 경우에는 한 개의 참조 페이지가 전체 참조횟수의 90% 이상을 차지하는 것을 관찰할 수 있었다.

이처럼 델타압축 시 사용되는 참조 페이지는 특정 페이지들에 집중되는 경향이 있으므로 실제 하드웨어 구현에서의 읽기버퍼 도입은 큰 성능향상을 가져올 것으로 예상된다.

6. 결론 및 향후작업

본 연구에서는 낸드 플래시 기반 저장장치의 내구성

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)

향상을 위한 쓰기 데이터양 감소 기법에 대해 연구하였다. 본 연구에서 제안한 MADE 모듈은 쓰기 데이터양을 줄이기 위해 주로 사용되는 중복제거기법(deduplication), 델타압축기법(delta-compression), 그리고 무손실압축기법(lossless compression)이 조합된 효과를 내며, 오버헤드적인 면에서 기존의 기법과 큰 차이가 없도록 설계되었다. 거기에 더해 기존 기법들에 비해 상대적으로 더 많은 쓰기 데이터양을 절감할 수 있었다.

MADE는 시뮬레이션에서 하나의 I/O 트레이스만을 제외한 모든 트레이스에서 중복제거기법과 무손실압축을 함께 적용한 경우보다도 우수한 성능을 보였다. MADE가 압축 성능 면에서 앞서지 못한 하나의 트레이스에서는 뒤지지 않는 결과를 보여주었다.

본 연구에서는 실제 하드웨어로 FTL을 구현해보지는 못하고 시뮬레이션을 통한 쓰기데이터 감소효과만을 관찰하여 기존 기법들에 비하여 오버헤드의 절대적 감소량이 얼마나 되는지 확인해보지 못하였다. 이런 시간적 오버헤드의 관점에서의 MADE에 대한 평가는 향후 연구에서 진행할 예정이다.

References

[1] Lee, Sungjin, et al., "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *Consumer Electronics, IEEE Transactions on*, Vol. 57, issue 4, pp. 1732-1739, 2011.

[2] Chen, Feng, et al., "CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives," *Proc. of the 9th Conference on File and Storage Technologies (FAST)*, 2011.

[3] Wu, Guanying, and He, Xubin, "Delta-FTL: improving SSD lifetime via exploiting content locality," *Proc. of the 7th ACM European Conference on Computer Systems (EuroSys)*, pp. 253-266, 2012.

[4] R. N. Williams, "An extremely fast ziv-lempel data compression algorithm," *Proc. of the Data Compression Conference*, pp. 362-371, 1991.

[5] Shilane, Phlip, et al., "WAN-optimizzed replication of backup datasets using stream-informed delta compression," *ACM Transactions on Storage (TOS), 2012*, Vol. 8, issue 4, No. 13.

[6] Bentley, Jon, and Douglas, McIlroy, "Data compression using long common strings," *Proc. of Data Compression Conference. IEEE*, pp. 287-295, 1999.

[7] Rajaraman, Anand, and Ullman, "Mining of massive datasets," *Cambridge University Press*, pp. 71-126, 2011.

[8] Korn, D, Macdonald, J, and Mogul, J, "RFC 3284: The vcdiff generic differencing and compression data format," *Internet Engineering Task Force (IETF)*, 2002.

권 혁 준

2015년 서울대학교 산림과학부/컴퓨터공학부 학사. 관심분야는 컴퓨터 구조, 프로세서 설계, 임베디드 소프트웨어

김 도 현

2009년~2015년 현재 서울대학교 컴퓨터공학부 학사과정. 관심분야는 임베디드 시스템, 컴퓨터 구조

박 지 성

2011년 서울대학교 컴퓨터공학과 학사 동 대학교 석박통합과정. 관심분야는 임베디드 시스템, 컴퓨터 구조, 메모리 및 저장장치 최적화

김 지 홍

정보과학회논문지
제 42 권 제 5 호 참조

서울대학교 | IP: 147.46.240.126 | Accessed 2015/12/30 15:16(KST)