파일유사도 정보를 이용한 고정 분할 기반 중복 제거 기법

문영찬, 정호민, 고영웅
한림대학교 컴퓨터공학과
e-mail: {vzmyc, hmjung, yuko}@hallym.ac.kr

Efficient Deduplication Scheme on Fixed-length Chunking System Using File Similarity Information

Young Chan Moon, Ho Min Jung, Young Woong Ko
Dept. of Computer Engineering, Hallym University

요 약

기존의 고정 길이 분할 (FLC: Fixed Length Chunking) 중복 제거 기법은 파일이 조금이라도 수정이 되면 수정된 블록에 대한 해시 정보가 달라져 중복 데이터 임에도 불구하고 중복 블록으로 검색이 되지 않는 문제점이 있다. 본 연구에서는 FLC 기반의 중복 제거 기법에 데이터 위치(offset) 정보를 활용하여 중복 블록을 효율적으로 찾아냄으로써 기존의 FLC 기반의 중복 제거 기법보다 더 좋은 성능을 발휘하는 유사도 정보를 활용하는 중복 제거 기법(FS_FLC: File Similarity based Fixed Length Chunking)을 설계하고 구현했다. 실험 결과 제안한 알고리즘은 낮은 오버헤드로 가변 분할 기법(VLC: Variable Length Chunking)만큼의 높은 중복 데이터 탐색 성능을 보여주었다.

1. 서론

IT산업의 발달로 PC, 스마트폰, 디지털TV등 하드웨어가 나날이 발전하고 더불어 그 안에 포함되는 컨텐츠들 또한 방대하게 증가하고 있다. 따라서 이에 따른 안전한 데이터 저장문제도 굉장히 중요시 되고 있는 분야 중 하나이다. 빅-데이터 시대로 도래하면서 데이터 저장 공간도 그만큼 기하급수적으로 많이 필요하게 되었다. 따라서 데이터가 거의 동일하고 조금만 수정된 파일의 경우 새로 저장하면 데이터 공간이 새로 생성 된 파일만큼 더 필요하게 된다. 하지만 새로 추가된 부분이나 수정된 부분만을 저장하게 된다면 그 만큼 데이터 저장 공간을 절약할 수 있게 됨으로써 중복제거가 절실히 필요하게 되었다.

기존에 널리 사용되고 있는 중복제거 기법으로는 고정 길이 분할(FLC: Fixed Length Chunking)을 이용한 중복제거기법과 가변길이 분할(VLC: Variable Length Chunking)을 이용한 중복제거기법이 사용되고 있다. FLC 기법은 파일을 고정된 길이로 블록을 나누기 때문에 수행시간 면에서 이점이 많다. 하지만 파일이 조금이라도 수정이 된다면 해당 블록의 해시 값이 달라져 중복된 부분을 검색하기 어렵다. VLC 방식은 중복데이터 검색능력은 뛰어나지만 시간이 오래 걸리고, 블록 길이가 일정치 않다는 단점이 있다.

본 연구에서는 기존의 FLC 방식에 파일 유사도 기법을 적용하는 중복 제거 기법(FS_FLC: File Similarity based Fixed Length Chunking)을 제안한다. 특히 파일 유사도 정보에 데이터의 위치 정보(offset)를 기록하고 이를 활용하여 효율적으로 중복 데이터 탐색을 수행하여 전반적으로 중복 데이터 검색 오버헤드를 줄일 수 있다. 이러한 방식을 이용하면 기존의 FLC 방식보다 훨씬 더 많은 중복 데이터를 검색할 수 있고, 시간적인 면에서 VLC 방식보다는 빠른 수행시간을 보여주었다.

본 논문의 구성은 다음과 같다. 2장에서는 중복제거 관련 연구에 대해 살펴보고, 3장에서는 파일유사도의 개념을 설명하고 본 연구에서 새롭게 제안하는 중복제거 시스템에 대한 설계 및 구현에 대하여 기술한다. 4장에서는 개선된 알고리즘에 대한 성능평가를 기술하고 마지막으로 5장에서 결론을 맺는다.

2. 관련연구

데이터 중복 제거에 관한 연구는 백업 시스템이나 네트워크를 중심으로 데이터의 공통점을 파악하고, 저장 및 문자열과 중복된 데이터 전송으로 인한 네트워크 자원 소비를 줄이고 디스크 기반 백업 스토리지 시스템의 전력 소비 효율 향상[1]을 위해 제안되어 연구가 이루어져 왔다.

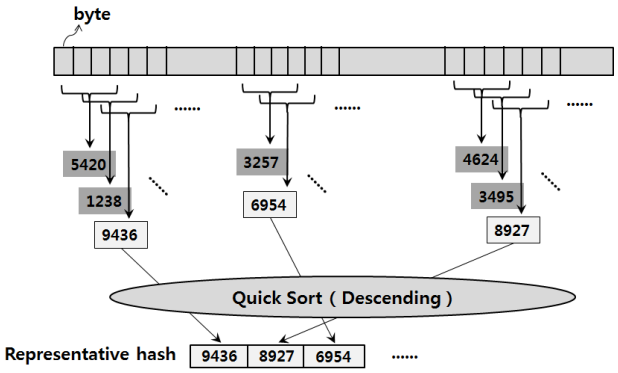
본 연구는 교육과학기술부와 한국연구재단의 지역혁신인력양성사업과 기초연구사업(No.2009-0076520)의 지원을 받은 결과물임을 밝힙니다.

Chord[2], Pastiche[3]는 라우팅을 위한 P2P 오버레이 네트워크와 파일의 중복을 막기 위해 MD5, SHA1 해시 함수를 사용하여 해시를 만들고 이를 비교하여 처리한 후 중복된 데이터를 제외하고 저장한다. Rsync[4]는 네트워크로 연결된 디렉토리의 데이터를 동기화 시켜주는 프로그램으로 롤링 체크섬(Rolling Checksum)이라는 중복 데이터를 검색하는 알고리즘을 사용해 새로운 데이터의 복사만 일어나게 한다. Venti[5]는 네트워크 스토리지 시스템에서 중복 데이터를 제거하여 저장하는 스토리지 시스템이다. Venti는 데이터를 저장할 경우 파일을 고정된 크기(8Kbyte)의 블록으로 나누고 각 블록에 SHA1 해시를 적용하여 160bit 크기의 해시를 만들고 전송한다. LBFS[6]는 자주 끊기거나 품질이 좋지 않은 네트워크 환경을 위해 설계된 네트워크 파일 시스템이다. LBFS에서는 파일 전송 효율을 높이기 위해 CDC(Content-defined Chunks) 방식을 사용한다. CDC는 Rabin Fingerprint로 해시하여 특별한 값을 반환하는 앵커(Anchor) 블록을 파일에 삽입하고 데이터 전송 전에 앵커 사이의 블록을 SHA1, MD5 같은 해시 함수를 사용해 해시를 만들고 해시 테이블과 비교하여 중복을 검색하는 방법이다. 최근에는 중복 탐지 알고리즘을 개선하여 중복제거 시스템을 향상 시키는 연구[7]도 이루어졌다.

3.파일 유사도 정보를 활용한 중복 제거 시스템

3.1 파일 유사도 개념

(그림 1)은 논문에서 제안하는 알고리즘에서 파일 유사도 개념도이다. 파일 유사도를 검사하기 위해서는 바이트(byte)단위로 해시가 가능한 함수를 사용해야 한다.

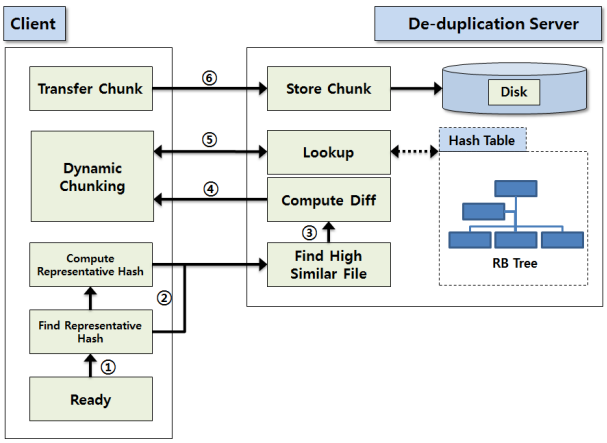


(그림 1) 파일 유사도 개념도

본 연구에서는 라빈(Rabin)해시 함수를 사용하여 해시 값을 구하였다. 그림에서 보이는 바와 같이 첫 번째 단계로, 파일에서 일정단위로 연속된 바이트들의 구간을 해시 함수를 이용하여 해시 값(5420)을 구한다. 그리고 다시 1 바이트를 옮겨 일정 단위로 연속된 바이트들의 구간을 해시 함수로 해시 값(1238)을 구한다. 이러한 방법으로 파일에 대해 해시 값을 모두 구한 뒤, 계산된 해시 값들을 내림차순으로 정렬을 하여 값이 큰 순서로 대푯값을 선택하고 발견된 위치(offset)를 함께 table에 저장한다. 파일 유사도의 결과 값은 해시 함수로 계산된 대푯값 10개에 대해 비교하여 나타낼 수 있다. 서버와 클라이언트 양측에서 모두 파일에 대해 해시 값을 구하고, 대푯값 10를 오름차순 정렬에 의해서 구한다. 서버에서 구한 대푯값 10개와 클라이언트에서 구한 대푯값 10개를 비교하는데, 이때 해시 값이 같으면 동일 블록으로 생각할 수 있다. 10개 대푯값 중에서 1개가 같으면 파일 유사도 10%, 2개가 같으면 파일 유사도 20%로 표현 할 수 있다.

3.2 시스템 설계 및 구현

(그림 2)는 본 연구에서 제안한 시스템의 동작 과정을 설명하는 그림이다.



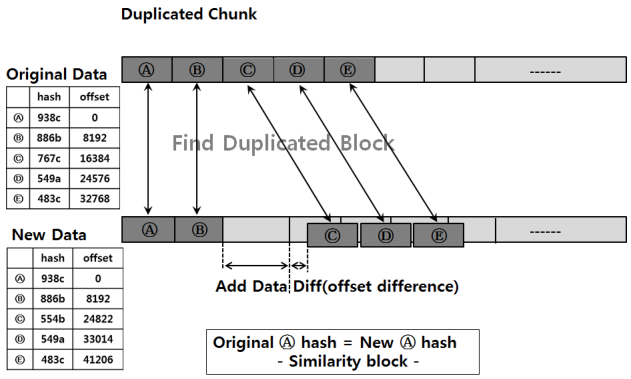
(그림 2) 시스템 설계도

시스템은 클라이언트와 중복제거 서버로 구성이 되어있다. 시스템의 흐름은 첫 번째로 클라이언트에서 파일의 유사도를 검사라기 위해 라빈 해시함수로 파일을 해시한다. 해시한 블록들에 대해서 값이 큰 해시 값들을 뽑기 위해 오름차순으로 정렬을 한다(Ready). 이 정렬한 값들 중에서 값이 큰 10개를 찾는다(Find Representative Hash). 대표 해시 값 10개를 서버로 전송한다. 서버도 클라이언트와 마찬가지로 라빈 해시함수를 이용하여 파일을 해시하고 그 중에서 값이 큰 해시 값을 클라이언트로부터 수신한 대표 해시 값들을 비교하여 유사도가 높은 블록을 찾아낸다(Find High Similar File). 유사도가 높은 블록이 발견되면 클라이언트에서 해시 값이 일치 하는 블록의 실제 위치를 찾기 위해 서버 측 블록 시작 위치와 유사 해시가 발견된 위치의 차이를 계산해야 한다(Compute Diff). 클라이언트는 전송 받은 diff를 가지고 블록 시작 위치에서부터 떨어진 위치만큼 이동하여 일정 블록을 나누어 읽어들이고 해시하여 중복인지 아닌지를 판별(Dynamin Chuncking)하고, 해당 블록이 유사하면 중복된 데이터로 처리하여 룩업 테이블에 해시 값과 위치 값(offset)을 저장한다. 그리고 나머지 블록에 대해서는 새로운 데이터로

구분하여 서버 측으로 블록들을 전송한 뒤, 디스크에 저장한다.

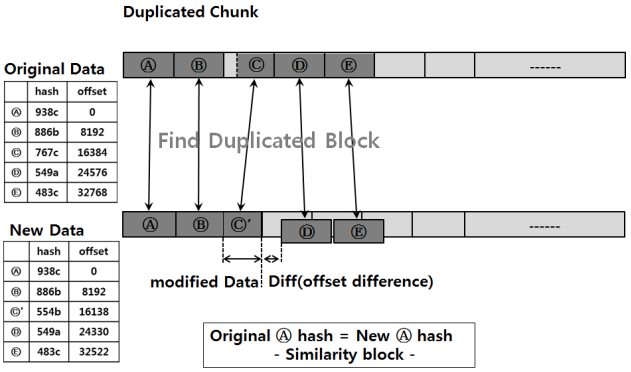
3.3 FS_FLC의 중복데이터 검색방법

다음 (그림 3)는 일반적으로 파일 중간에 새로운 데이터가 추가 되었을 경우에 FS_FLC의 중복데이터 검색 과정에 대한 설명 그림이다.



(그림 3) FS_FLC 중복데이터 검색 과정

중복 데이터를 검색하는 과정은 첫 번째 단계로, 서버와 클라이언트에서 파일 유사도 측정을 기반으로 라빈 해시 함수를 이용하여 해시 값을 계산하고, 그 중 대푯값 10개와 각 위치(offset)를 테이블에 작성한다. 두 번째 단계로 대표 해시 값의 위치를 기준으로 일정크기 만큼씩 동일하게 블록을 나눈다(FLC). 세 번째 단계는 서버의 대표 해시 값과 offset, 클라이언트의 대표 해시 값과 offset을 비교하는 과정이다. (그림 2)에서 보는바와 같이 Ⓐ, Ⓑ 블록은 해시 값과 offset이 같다. 해시 값과 offset이 같다면 파일유사도가 유사하므로 동일한 데이터를 가진 블록이 서버와 클라이언트 모두 있다고 판단 할 수 있다. 실제 파일에서 같은 위치(서버의 원본 파일에서의 데이터 블록 위치와 클라이언트의 수정된 파일에서의 데이터 블록 위치)에 존재하므로 Ⓐ, Ⓑ 블록은 중복 데이터로 처리하여 그대로 백업을 하면 된다. 문제점은 Ⓒ이후의 블록들처럼 해시 값은 같지만, offset이 다른 경우는 중복데이터 블록의 정확한 위치를 모르기 때문에 찾아야 한다. 해시 값은 같고 offset이 다른 데이터 블록의 위치를 찾는 방법은 그림에서 보는 바와 같이 Ⓒ를 예를 들어 설명하겠다. Ⓒ블록은 서버와 클라이언트 모두 해시 값이 같기 때문에 유사한 블록이 맞다. 서버가 Ⓒ의 정확한 위치를 알기 위해서는 클라이언트에서 Ⓒ의 해시 값이 발견된 위치에서 클라이언트에서 새로운 데이터가 추가된 부분 뒤의 위치만큼의 차이를 빼주어 위치의 차이(diff)를 구하고, 이 diff를 서버로 전송해줘야 한다. diff는 고정길이 중복제거 방식을 사용한다는 점에서 각 블록의 시작 위치로부터 알 수 있다. 블록크기가 8192일 경우, 파일의 첫 블록의 시작위치는 0이고, 두 번째 블록의 위치는 8192이다. 따라서 Ⓒ의 offset을 8192로 나누어서 떨어진 나머지가 위치 차이인 diff가 된다. 서버는 이 diff를 전송 받아 해당 블록으로부터 diff만큼 이동하여 해당 블록 사이즈만큼 데이터를 다시 읽고, 해시하여 이 데이터 블록이 유사한지 아닌지를 판별한고, 유사하면 중복데이터 블록으로 처리하고 나머지 블록에 대해서 순차적으로 작업을 진행한다.



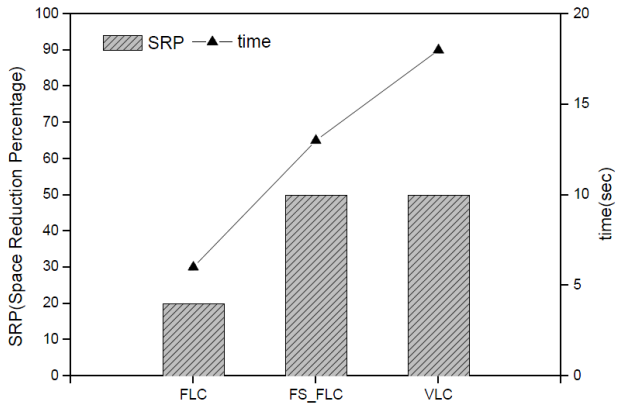
(그림 4) 파일 중간이 수정된 경우

(그림 4)은 클라이언트 측면에서 기존에 동일했던 블록에 대해서 중간에 삭제가 되었을 경우이다. Ⓐ, Ⓑ 블록은 파일 유사도가 같은 블록이다. 따라서 중복데이터로 처리하면 된다. 문제는 클라이언트 측면의 Ⓒ′블록이다. 이 블록은 서버의 원본 파일의 Ⓒ와 비교 하였을 때, 데이터가 수정되어 완전히 다른 블록이다. 하지만, 파일유사도 측정을 통해 Ⓒ블록과 Ⓒ‘블록의 유사도를 비교해보면 유사한 블록이 존재하는 것을 찾을 수 있다. 데이터 블록의 일부만 수정 되었을 뿐 나머지 부분은 중복된 부분이 있을 수 있기 때문이다. Ⓓ,Ⓔ 블록에 대한 중복 데이터 검색은 (그림 3) 설명한 방식대로 diif를 계산하여 위치를 찾고, 중복을 처리할 수 있다.

4.성능평가

본 연구의 실험은 서버/클라이언트 네트워크 환경을 기반으로 진행되었다. 운영체제는 양측모두 윈도우기반이고 프로그램은 자바로 작성되었으며, 데이터는 실험목적에 의해 중복률 각 50%로 고정된 데이터파일을 사용하였다.

(그림 5)에서 보는 바와 같이 50% 중복데이터를 가지고 알고리즘별 검색률과 검색시간을 측정한 결과이다. FLC방식이 중복제거를 하는데 걸리는 시간이 가장 짧게 걸리는 것을 알 수 있다. Fixed하게 Chunk를 검색하여 유사도를 측정하기 때문에 다른 방법보다 블록비교 횟수가 상대적으로 적다. 그리고 시간적인 면에서는 가장 좋지만, 중복제거를 얼마나 하는가에 대한 문제에 있어서는 세 방법 중 가장 성능이 나쁘다는 사실도 알 수 있다. 실험결과 FS_FLC가 시간적인 면에서는 기존의 FLC보다 더 걸리지만, 중복데이터를 비교적 많이 찾아내는 VLC방식과 비교



(그림 5) 알고리즘별 중복데이터 검색률 및 시간

해 보았을 때 시간도 훨씬 덜 걸렸고, 중복데이터도 걸린 시간에 비해 VLC 만큼 찾을 수 있었다. 또 FS_FLC의 큰 장점은 고정길이 블록을 사용한다는데 있다. 고정길이 블록을 사용하면 가변길이 일 때에 비해 데이터 검색 오버헤드를 줄일 수 있다. FS_FLC는 시간당 중복 데이터 검색 성능이 뛰어나기 때문에 시간적인 부분에서 메리트를 갖는 스토리지 시스템에서 사용된다면 도움이 될 것으로 예상된다.

5.결론 및 향후연구

본 연구에서는 VLC의 중복제거 성능과 비슷한 성능을 보이는 FS_FLC를 설계하고 구현하였다. 주 아이디어는 파일 블록의 offset을 이용하여 파일이 중간에 수정되거나 데이터가 추가 되었을 시, 동일 해시에 대해 offset 차이만큼 이동하고 고정된 길이의 블록을 다시 읽어 들여 중복제거를 하는데 있다. 실험결과 본 연구에서 FLC보다 좀 더 중복된 데이터를 더 찾았고, VLC와 거의 비슷한 중복데이터 검색 성능을 보였다. 향후 연구로는 제안한 알고리즘을 스마트폰 백업시스템에 적용하여 에너지효율성을 높일 수 있도록 하고, 서버의 디스크 스토리지 사용량을 줄이는데 적용할 것이다.

참고문헌

[1] Zhichao Li, Stony Brook University; Kevin M. Greenan and Andrew W. Leung, EMC Corporation; Erez Zadok, Stony Brook University. : Power Consumption in Enterprise-Scale Backup Storage Systems: FAST 2012 Conference on Back it Up.(2012)

[2] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. Networking, IEEE/ACM Transactions on 11(1) (2003) 17-32

[3] Cox, L., Murray, C., Noble, B.: Pastiche: Making backup cheap and easy. ACM SIGOPS Operating Systems Review 36(SI) (2002) 285-298

[4] Tridgell, A.: Ecient algorithms for sorting and synchronization. PhD thesis, PhD thesis, The Australian National University (1999)

[5] Quinlan, S., Dorward, S.: Venti: a new approach to archival storage. In: Proceedings of the FAST 2002 Conference on File and Storage Technologies. Volume 4. (2002)

[6] Muthitacharoen, A., Chen, B., Mazieres, D.: A low-bandwidth network file system. ACM SIGOPS Operating System Review 35(5) (2001) 174-187

[7] Fanglu Guo and Petros Efstathopoulos, Symantec Research Labs. : Building a High-performance Deduplication System: 2011 USENIX Conference on USENIX Annual Technical Conferenc.(2011)