파일의 의미기반분할을 위한 효율적인 점진형 Modulo-K 알고리즘

반종명 ○, 원유집, 민재홍

한양대학교 전자컴퓨터통신공학과 분산멀티미디어 연구실

{bankun, yjwon, milkiway} @ece.hanyang.ac.kr

Incremental Modulo-K algorithm for Content Based File Partitioning

Jongmyeong Ban○, Youjip Won, Jaehong Min

DMC Lab. Dept. of Electronics and Computer Engineering, Hanyang University

요 약

현재의 스토리지 시스템 및 백업 시스템에서 중복된 데이터의 증가로 인한 문제가 점점 대두되고 있다. 이러한 중복 데이터를 검사하거나 파일의 유사성을 검사하는 데에는 BSW알고리즘이 많이 사용되고 있으며 이 BSW알고리즘 내에서 해시 값을 구하는데에는 Rabin's Fingerprint알고리즘이 일반적으로 사용되고 있다.

본 논문에서는 Rabin's Fingerprint알고리즘에 비해 보다 빠르고 간단한 Modulo-K알고리즘을 제안하며 BSW알고리즘에서 Rabin's Fingerprint를 대체함으로써 최종적으로 BSW알고리즘의 속도를 향상시킬 수 있는 Modulo-K알고리즘을 제안한다.

1. 서 론

오늘날 디지털 데이터의 양은 빠른 속도로 증가하고 있으며 이러한 디지털 데이터를 저장 및 관리하는 스토리지 시스템의 중요성은 점점 커져가고 있다[1]. 사용자들은 네트워크를 통하여 많은 양의 디지털 데이터를 전송하고 있으며 이러한 데이터들에는 많은 부분에서 동일한 내용을 포함하고 있다. 일반적으로 이러한 중복 데이터 문제는 P2P서비스나 웹하드, 이메일의 첨부파일 등에서 쉽게 찾아볼 수 있으며 이로인하여 많은 양의 스토리지 공간이 소모되고 있다.

과거 이러한 중복 데이터문제는 다루어지는 데이터의 크기가 작았기 때문에 크게 문제되지 않았다. 하지만, 현재의 고용량 시스템에서는 사용되는 데이터의 크기가 엄청난 속도로 증가하고 있기 때문에 이 문제가 크게 대두되었고 해결방법이 필요하게 되었다.

이미 중복데이터를 감지하여 관리하는 파일시스템[2, 3]이나 중복감지 백업[4, 5] 및 중복데이터를 관리 및 탐지하는 다양한 방법[6-8]등이 연구되었다. 중복데이터를 관리하는데는 일반적으로 파일을 청크(chunk) 혹은 블록(block)이라는 단위 조각으로 나누어서 그 조각을 비교하는 방법이 사용되고 있다.

현재는 보다 정확한 중복데이터를 탐지하기 위하여 컨텐츠 기반의 가변크기 청크로 나누는 방법이 많이 사용되고 있으며 이때 사용되는 알고리즘은 BSW(basic sliding window)알고리즘이다. 따라서 이 BSW알고리즘의 속도를 높인다면 중복감지로 인한 오버헤드를 감소시킬 수 있으며 전체적인 중복감지 백업 및 관리 시스템의 성능을 향상시킬 수 있다.

2. 배경

2.1. 중복 감지 방법

2.2.

가장 확실하게 중복을 확인하는 방법으로는 비트 대 비트 비교방법이다. 이것은 비교하려는 모든 데이터에 대해서 1:1로 비교하는 방법으로서 비교하려는 대상이 작을 때는 문제가 없었지만 현재의 대용량 시스템에서 이러한 방식으로 비교를 할 경우 아무리 시스템 성능이 높아지더라도 사용할 수 없을 정도로 느린 방법이다.

다른 방법으로는 해시 값 비교가 있다. 이것은 비교하려는 파일들에 대해서 해시 값을 구하고 각 파일에 대한 해시 값을 비교함으로써 파일의 중복여부를 탐지하는 방법이다. 이 방법은 비트 비교에 비해서 빠른 속도를 보장하지만 파일의 내부에 존재하는 부분 중복은 찾지 못하는 문제가 있다.

이러한 파일의 내부에 존재하는 중복을 탐지하는 방법으로는 파일을 청크라 불리는 조각으로 나누어 이 청크에 대한 해시 값을 비교하는 방법이 있다. 이 방법은 크게 두 가지로 나눌 수 있는데, 하나는 고정크기 청크

이고 다른 하나는 가변크기 청크이다. 고정크기 청크의 경우 1KB, 2KB, 4KB등 하나의 블록사이즈단위로 파일을 나누어 그 해시 값만을 비교하는 방법으로 아주 빠른 속도를 가지고 있다. 하지만 원본파일과 약간의 데이터가 수정된 파일간에 중복을 찾는 것에는 취약한 문제가 있다[그림 1].

두번째 방법으로는 컨텐츠에 기반해 청크를 나누는 방법으로 이때 나누어진 청크의 크기가 데이터의 내용에 따라 가변적으로 정해짐으로써 고정크기 청크의 문제점을 해결한 방법이다[그림 2].

[그림 2]에서는 파일이 (a)에서 (b),(c),(d)로 변했을 때의 나누어진 청크의 변화를 보이고 있다. (b)는 새로운 데이터가 추가되었을 때 청크$_1$이 청크$_6$으로 변하지만 청크$_1$을 제외한 다른 청크$_2$~청크$_4$는 변함이 없어 중복 데이터라는 것을 알 수 있다. (c)는 청크$_2$의 내용이 수정됨에 따라 청크$_2$가 청크$_7$과 청크$_8$로 나뉘었지만 나머지 청크 들에는 변화가 없음을 알 수 있다. 역시 (d)에서도 청크$_3$과 청크$_4$의 내용이 삭제됨에 따라 청크$_3$과 청크$_4$가 합쳐져 청크$_9$가 되었지만 나머지 청크$_6$~청크$_8$,청크$_4$는 변화가 없음을 알 수 있다. 이 그림에서 같은 번호의 청크들은 해시 값이 동일하므로 전체 파일의 일부가 수정되더라도 수정이 안된 부분에 대해서 중복을 찾을 수 있다.

이러한 컨텐츠 기반 청킹을 하는데는 BSW(Basic Sliding Window)라 불리는 알고리즘이 사용된다.

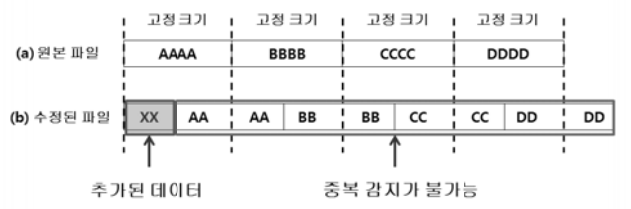

그림 1. 고정크기 청크의 문제점

그림 2. 가변크기 청크의 예

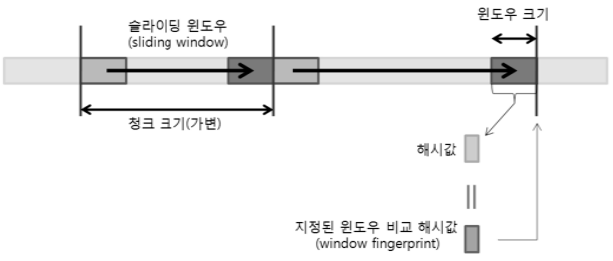

그림 3. Basic Sliding Window

2.3. Basic Sliding Window

[그림 3]은 BSW알고리즘의 동작을 설명하고 있다. BSW에서는 일정한 윈도우 크기가 이동해 가면서 이 윈도우의 데이터에 대한 해시 값을 계산한다. 이렇게 구해진 해시 값은 미리 지정되어있는 '비교 해시 값(window fingerprint)'과 동일한지 확인하는데, 이때 동일하지 않다면 다시 1바이트를 이동하여 다시 비교하고 동일하다면 그 위치에서 청크를 나누게 된다.

이러한 방식으로 BSW알고리즘은 파일의 처음부터 끝까지 파일의 데이터에 의존하여 청크를 나누게 되며 이 때문에 컨텐츠 기반 청킹(Contents based chunking)이라고도 한다.

BSW알고리즘에서 계속해서 윈도우를 이동해 가면서 해시 값을 구하기 위해서는 빠르고 가벼운 해시함수를 사용해야 하는데 암호화에 사용되는 MD5[9]나 SHA-1[10]등과 같은 해시함수는 이런 용도로 사용되기에 너무 느리며 출력되는 해시 값도 128~160비트로 필요에 비해 너무 크기 때문에 일반적으로 보다 빠르고 출력되는 해시 값이 크지 않은 Rabin's Fingerprint[11]라는 해시함수가 사용된다.

BSW알고리즘의 성능은 해시 값을 얼마나 빨리 구하고 비교하느냐에 좌우되므로 본 논문에서는 Rabin's Fingerprint보다 빠르고 가벼운 Modulo-K알고리즘을 도입하여 BSW알고리즘을 보다 빠르게 구현하고자 한다.

3. Modulo-K 구현

그림 4. Modulo-K알고리즘의 기본 컨셉

[그림 4]는 윈도우 크기가 48바이트일 때의 모습으로, $S_1S_2...S_6$및 $S_7S_8...S_{12}$는 각각 8바이트(총 48바이트)로 이전 윈도우와 1바이트 이동한 윈도우의 변화를 보이고 있다.

Modulo-K에서는 이전의 윈도우(W_1)에서 구한 해시 값을 다음 윈도우(W_2)의 해시값을 구하는데 사용함으로써 계산을 줄여 속도를 높이는 방식을 취하고 있다.

처음 해시값을 구하는데는 (식1)을 사용하며 (식2)는 이전값을 이용하여 해시 값을 구하는데 사용한다. 이때 K값은 $2^{31}-1$의 소수를 사용한다.

$$W_1 \% K = (S_1S_2S_3S_4S_5S_6) \% K = (2^{10} \times S_1 \% K + 2^8 \times S_2 \% K + 2^6 \times S_3 \% K + 2^4 \times S_4 \% K + 2^2 \times S_5 \% K + S_6 \% K) \% K \quad (1)$$

$$W_2 \% K = (S_7S_8S_9S_{10}S_{11}S_{12}) \% K = (b' \% K + b \% K) \% K = ((a' \times 2^8) \% K + b \% K) \% K = (((K + (W_1 \% K) - a \times 2^4) \times 2^8) \% K + b \% K) \% K \quad (2)$$

[식2]는 [그림 4]에서 W_1과 W_2를 비교했을 때 a'와 b'는 동일한 데이터를 가지고 있고 차이점은 a와 b밖에 없다는 점을 이용한다. 따라서 각각에 대한 모듈러 연산(식1) 대신 이전에 계산한 값을 이용하고 이전 값의 a와 새로 들어온 값 b만을 더해 계산하는 방법으로 모듈러 연산횟수를 줄일 수 있다(식2). 모듈러 연산은 다른 연산에 비해 매우 느린 연산이므로 이 연산횟수를 줄이는 것으로 커다란 속도향상을 얻을 수 있다.

4. Modulo-K의 엔디안 문제

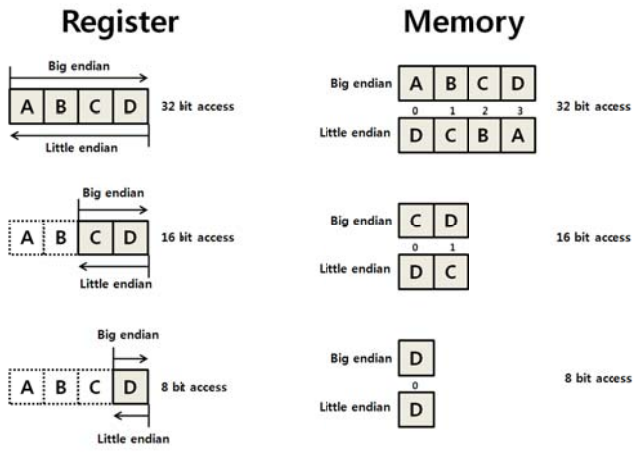

그림 5. 리틀-엔디안 과 빅-엔디안의 차이

현재 존재하는 프로세서에는 일반적으로 x86, 6502, Z80등에 사용되는 리틀-엔디안과 PDP-11, Motorola 6800, 68000등 에서 사용되는 빅-엔디안방식의 두가지로 나눌 수 있다.

[그림 5]는 이 두 엔디안의 차이를 보이고 있다. Modulo-K에서는 빅-엔디안을 기준으로 계산되기 때문에 일반적으로 사용되는 리틀-엔디안 머신에서는 정상적으로 계산되지 않는 문제가 발생하게 된다. 따라서 우리는 네트워크 어플리케이션에서 바이트 순서를 정렬하는데 사용되는 *'htonl'*함수를 사용하여 이 문제를 해결한다. 물론 이 함수를 사용하는 것은 성능상 오버헤드가 되지만 Modulo-K는 (식1)을 계산할 때만 바이트 순서 정렬을 수행하며 (식2)를 계산할 때는 바이트 정렬을 할 필요가 없다. (식1)은 (식2)에 비해서 아주 적은 횟수만 수행되며 청크의 평균 크기가 8KB일 때 약 1:6000의 비율로 수행되게 된다. 따라서 바이트 순서 정렬로 발생하는 오버헤드는 극히 미미한 수준이다.

비록 리틀-엔디안 머신에서 Modulo-K는 약간의 오버헤드가 있지만 여전히 Rabin's fingerprint보다 훨씬 빠르고 간단한 해시함수로서 BSW의 성능을 높일 수 있으며 그 Rabin's Fingerprint와 Modulo-K와의 성능 비교는 다음 성능평가에서 보이고 있다.

5. 성능평가

표 1. 실험환경

하드웨어	사양
CPU	Intel Core 2 Duo 6750 (2.66GHZ)
Memory	2GB (DDR2)
Harddisk	SATA 320GB (Samsung)
OS	Linux 2.6.22-14 (Ubuntu Gutsy)

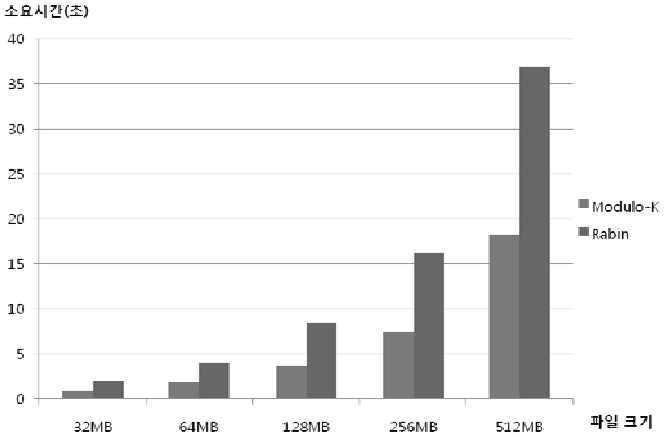

그림 6. Modulo-K와 Rabin의 비교

본 실험에서 사용된 데이터는 32MB, 64MB, 128MB, 256MB, 512MB의 동영상 파일이며 [그림 6]은 Modulo-K와 Rabin's Fingerprint의 속도를 비교하고 있다. 실험은 각각의 파일을 읽어 각각 Modulo-k와 Rabin's Fingerprint를 사용하여 BSW알고리즘을 수행, 청크를 나누었을 때 걸리는 시간을 측정하였으며 총 10번의 실험을 하여 평균을 구하였다.

[그림 6]의 그래프에서 보이듯이 Modulo-K는 Rabin's Fingerprint에 비해서 약 2배의 속도를 보이고 있다.

6. 결론

오늘날의 컴퓨터 시스템은 기술의 발달로 인하여 저렴해진 저장장치의 가격과 빠른 네트워크 속도로 인하여 과거에 비해 엄청난 양의 데이터가 사용되고 있다.

이는 무분별한 중복데이터를 발생시켰고 이를 탐지하여 관리하는 기술이 필요해지게 되었다.

이미 기존의 여러 연구에서 백업할 때 이러한 중복 데이터를 탐지하여 백업의 양을 줄이거나 네트워크로 전송할 때 중복된 데이터를 전송하지 않음으로 네트워크 대역폭을 절약하는 등의 중복 데이터를 관리하는 방법에 대한 논의가 있었지만 이 중복데이터를 판별하기 위해 데이터를 청크와 같은 조각으로 나누는 오버헤드나 속도의 향상에 대한 연구는 미흡한 실정이었다.

본 논문에서는 가장 많이 사용되는 BSW알고리즘의 Rabin's Fingerprint알고리즘을 대체하여 BSW의 성능을 향상시킬 수 있는 Modulo-K알고리즘을 제시하였다. Modulo-K는 기존 Rabin's Fingerprint를 사용하는 시스템에 쉽게 적용하여 전체적인 성능을 향상시킬 수 있으므로 중복감지 시스템의 성능향상에 도움이 될 것이다.

7. 참고문헌

[1] J. McKnight, T. Asaro, and B. Babineau, "Digital Archiving: End-User Survey and Market Forecast 2006-2010," The Enterprise Strategy Group Jan. 2006.

[2] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles* Banff, Alberta, Canada: ACM Press, 2001.

[3] B. Hong, D. Plantenberg, D. D. E. Long, and Sivan-Zimet, "Duplicate data elimination in a san file system," in *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*. 2004, pp. 301--314.

[4] Q. Sean and D. Sean, "Venti: A New Approach to Archival Storage," in *Proceedings of the Conference on File and Storage Technologies*: USENIX Association, 2002.

[5] L. L. You, K. T. Pollack, and D. D. E. Long, "Deep Store: An Archival Storage System Architecture," in *21st International Conference on Data Engineering (ICDE'05)*, 2005, pp. 804-8015.

[6] N. Jain, M. Dahlin, and R. Tewari, "TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization," in *In Proceedings of the 4th USENIX Conference on File and Storage Technologies*, 2005.

[7] F. George, E. Kave, and C. Stephane, "Finding similar files in large document repositories," in *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* Chicago, Illinois, USA: ACM Press, 2005.

[8] U. Manber, "Finding similar files in a large file system," in *Proceedings of the Winter 1994 USENIX Technical Conference*, San Francisco, CA, Jan. 1994.

[9] R. L. Rivest, "The MD5 Message Digest Algorithm," Request for Comments(RFC) 1321, Internet Activities Board, 1992.

[10] F. 180-1, "Secure hash standard," U.S. Department of Commerce/N.I.S.T.: Springfield, Apr. 1995.

[11] M. O. Rabin, "Fingerprinting by Random Polynomials," Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.