중복 데이터 관리 기법을 통한 저장 시스템 성능 개선

정호민*, 김병기, 김진, 고영웅
한림대학교 컴퓨터공학과
e-mail:{chorogyi, bkkim, jinkim, yuko}@hallym.ac.kr

Performance Improvement of Storage System Using De-duplication

Ho-Min Jeong*, Byung-Ki Kim, Jin Kim, Young-Woong Ko
Dept. of Computer Engineering, Hallym University

요 약

기존의 저장 방식은 대용량의 데이터를 비효율적으로 처리해 왔지만 데이터 중복 제거 기법을 이용하면서 저장 공간과 네트워크 대역폭을 효율적으로 사용할 수 있게 되었다. 그러나 기존의 데이터 중복 제거 알고리즘들은 수행시간이 길고 중복 데이터를 효율적으로 제거하지 못하는 문제가 있다.
본 논문에서는 개선된 중복 검색 및 제거 메커니즘을 제공하는 저장 시스템을 제안하고 있다. 제안하는 알고리즘은 저장 공간을 최소화하고 네트워크 대역폭을 줄일 수 있다. 주요 아이디어는 스트라이드 방식의 중복 검색 메커니즘이며 중복된 데이터 블록을 발견하는데 있어서 계산 시간을 줄여주고 있다. 제안하는 시스템의 성능을 검증하기 위하여 리눅스 배포 데이터를 저장하는 실험을 수행하였으며 실험 결과 스트라이드 방식이 저장 공간을 줄이고 중복된 데이터 블록을 효율적으로 관리할 수 있음을 보였다.

1. 서론

컴퓨터를 이용한 문서 작업이 증가하고, 인터넷을 이용하여 데이터를 교환하는 것이 일반화되면서 저장 시스템(storage system)에 대한 요구 사항이 지속적으로 중요해지고 있다. 대부분의 저장 시스템은 사용자의 요구에 의해서 다양한 종류의 파일을 일정한 크기의 블록으로 변환하여 저장을 하게 되는데, 최근 들어 중복된 데이터에 대한 효율적인 관리 기법이 주요한 관심이 되고 있다. 특히 동등 계층 통신(P2P: peer to peer) 시스템, 백업(backup) 시스템, ftp 밀러(mirror), 가상화(virtualization)[1] 시스템 등에서 높은 비율의 중복 블록이 발생되고 있다. 예를 들면 리눅스 ftp 밀러의 저장 서버에서 동일한 파일을 다양한 종류의 미디어 포맷으로 변환(CD이미지, DVD이미지, RPM 파일 등)하여 서비스하고 있으며, 중복 데이터 블록이 50% 이상 발생하고 있다. 또한 P2P 시스템에서는 동일한 멀티미디어 파일이 이름만 변경된 상태로 다수가 존재하는 경우도 비일비재하다.

본 연구는 산업자원부와 한국산업기술재단의 지역혁신인력양성사업의 지원을 받아 연구되었음

이와 같이 저장 시스템에 존재하는 중복된 파일 및 블록에 대한 효율적인 데이터 관리를 위하여 스토리지 시스템을 설계하는 단계에서 중복 제거(duplication elimination)를 위한 방안이 제공되어야 할 필요성이 있다.

본 논문에서는 기존에 사용되는 중복 블록 제거 기법을 개선할 수 있는 새로운 알고리즘을 제시하고 있다. 제안하는 기법은 고정 블록 중복 제거 방식에 스트라이드(stride) 기법과 롤링 체크섬(Rolling Checksum)을 적용하는 것이다. 실험 결과에서 볼 수 있듯이 중복 제거 성능을 극대화하면서 수행 시간을 최소화시키고 있음을 알 수 있다. 제안하는 중복 제거 알고리즘은 리눅스 CD이미지와 DVD 이미지의 중복을 50% 정도 찾고 있으며 수행속도는 기존의 롤링 체크섬 기법보다 5배 이상 빠른 성능을 보이고 있다. 본 논문의 구성은 다음과 같다. 먼저 2장에서는 중복 제거와 관련된 최근 연구 동향에 대해서 알아보고, 3장에서는 제안하는 스토리지 시스템의 설계 원리와 중복 제거 알고리즘의 동작에 대해 설명한다. 4장에서는 제안하는 시스템의 구현 및 성능평가에 대해 기술하였으며 마지막으로 5장에서는 결론 및 향후 연구방향을 제시한다.

2. 관련 연구

Rsync[2]는 네트워크로 연결된 디렉토리의 데이터를 동기화 시켜주는 프로그램인데 롤링 체크섬(Rolling Checksum)이라는 중복 데이터를 검색하는 알고리즘을 사용해 새로운 데이터의 복사만 일어나게 하는 프로그램이다. 롤링 체크섬은 원본 파일을 일정하게 나누어 블록단위의 해시 값을 생성하고 비교하려는 파일은 슬라이딩(Sliding) 기법을 이용하여 바이트 단위별로 모든 블록을 중첩(overlapping) 시켜 해시를 만들고 원본파일의 해시 리스트와 비교하여 중복된 부분을 찾는다.

Plan9[3]의 Venti[4]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. 만약 블록의 해시가 스토리지에 있을 경우 중복으로 간주하여 블록의 저장을 피한다. Venti는 이전 저장 정보가 없이도 델타(delta) 백업의 효과를 낼 수 있으며 실험 결과에서는 다른 스냅샷(snapshot) 시스템과는 달리 30%의 저장 공간을 줄이는 것으로 나타났다. Venti의 단점으로는 중복 데이터를 놓치는 경우가 발생하는 것을 들 수 있다. 파일의 수정이 일어나게 되면 수정이 일어난 뒷부분의 블록들의 위치가 달라지며 Venti는 수정이 일어난 뒷부분의 블록을 다른 해시 값으로 계산하여 중복을 인식하지 못한다.

LBFS[5]는 자주 끊기거나 품질이 좋지 않은 네트워크 환경을 위해 설계된 네트워크 파일 시스템이다. LBFS에서는 파일 전송 효율을 높이기 위해CDC(Content-defined Chunks) 방식을 사용한다. CDC는 Rabin Fingerprint[10]로 해시하여 특별한 값을 반환하는 앵커(Anchor) 블록을 파일에 삽입하고 데이터 전송 전에 앵커 사이의 블록을 SHA1, MD5같은 해시 함수를 사용해 해시를 만들고 그 해시들을 전송하여 서버에서 캐싱된 해시 룩업(Lookup) 테이블과 비교하여 중복을 검색하는 방법이다.

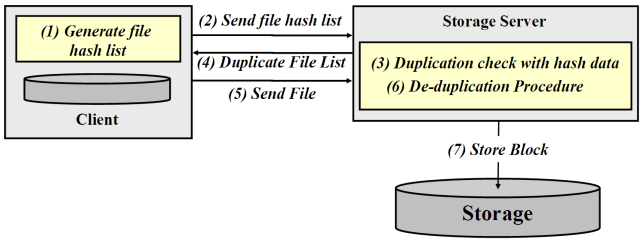
IDE[6]에서는 CDC 방법의 문제를 수정하였는데 빈번한 수정에 의해 앵커 사이의 블록 크기가 너무 작거나 커지는 문제를 해결하기 위해 계층적 해시데이터 구조와 희소행렬(sparse matrix)을 설계하여 블록 크기를 고르게 하였다. 이외에 HP에서 TTTD(Two Thresholds, Two Divisors)알고리즘[7]을 설계하여 앵커 사이의 블록 크기의 편차를 최소화 했으며 이를 통해 클라이언트와 서버간 통신에 필요한 오버헤드가 줄어든 것을 확인하였다. DRED[8] 시스템은 델타 인코딩 기법을 사용해 웹 페이지, 전자메일과 같은 데이터의 중복을 효율적으로 제거하였다. 델타 인코딩은 시간적인 순서로 연관된 두 개의 데이터 집합의 파일 이름, 크기 등의 데이터를 통해 압축하는 방식이다.

3. 중복 제거 스토리지 시스템 설계

중복 제거 저장 시스템의 목표는 클라이언트에서 네트워크를 통해 데이터를 스토리지 서버에 저장할 때 디스크 공간을 효율적으로 사용하고 수행 시간을 줄이는 것이다. 본 논문에서는 주요 중복 제거 알고리즘을 구현하여 중복 제거 효율과 수행성능을 비교할 수 있도록 하였다. 네트워크 스토리지 시스템에서 효율적인 수행을 위해 동일한 파일에 대해서는 재전송을 막고 저장된 파일에 효율적인 접근이 가능하도록 저장구조를 설계하였다.

3.1. 시스템 구조

본 연구에서 제안하는 시스템의 형태는 (그림 1) 에서 보이는 것과 같이 클라이언트와 중복 제거 서버로 구성되어 있다. 클라이언트는 사용자가 저장할 파일을 선택할 수 있도록 인터페이스를 제공하고 파일을 전송할 수 있도록 설계하였으며 중복 제거 파일 서버는 클라이언트로부터 받은 파일들의 중복을 제거하고 스토리지에 저장하는 일을 한다.



(그림 1) 저장 시스템 설계 개념도

클라이언트에서는 미리 저장할 파일들의 해시 리스트를 만들고 파일을 보내기 전에 해쉬 리스트를 서버에 전송한다. 파일 해시는 SHA1 해시 함수에 파일 스트림을 입력해 얻은 결과로 파일을 대표한다. 서버에서는 기존의 저장된 파일에 대한 해시 데이터를 미리 보관하여 클라이언트에서 파일 해시를 전송할 때 해시들을 서로 비교하여 파일의 중복을 판단할 수 있는 것이다. 중복된 파일을 탐색하는 작업은 계산 비용만 소모되고 특히 중복된 파일이 존재하면 재전송을 피할 수 있어 저장비용과 전송비용을 줄일 수 있다.

3.2 기존의 중복 제거 알고리즘

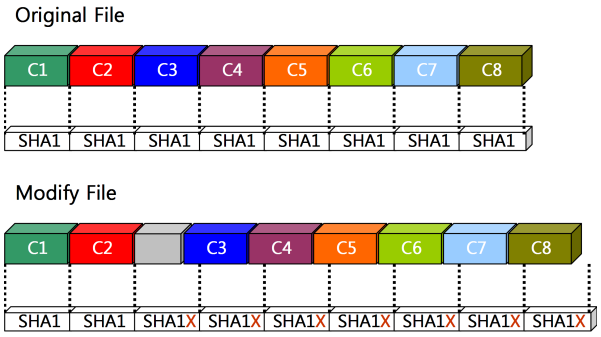
중복 제거 알고리즘의 범주를 대표적으로 Venti의 Duplicate Block Check with Fixed-size Blocking(DBC_FS), Rsync의 Duplicate Block Check with Byte Shift(DBC_BS), LBFS의 (CDC:Contents-Define Chunking)로 나눌 수 있다. LBFS의 알고리즘을 스토리지 시스템에 적용하기 어려우며 알고리즘 특성상 앵커(Anchor)가 필요하기 때문에 클라이언트에서 파일을 투명하게 처리 할 수 없다.

DBC_FS, DBC_BS 기법은 중복제거에 필요한 데이터가 해시 데이터 외에는 없고 어떠한 파일 조작도 일어나지 않기 때문에 스토리지 시스템에서 중복제거용도로 사용하기 적합하다. 본 연구에서는 DBC_FS, DBC_BS기법을 구현하였으며 몇 가지 아이디어를 추가하여 스토리지

시스템에 맞는 기법을 설계하였다.

■ DBC_FS 알고리즘

DBC_FS 알고리즘은 파일을 일정한 크기의 블록으로 나누어 SHA1, MD5와 같은 해시함수를 적용하여 블록을 대표하는 해시를 생성한다. 생성된 해시가 기존의 스토리지에 저장되었던 해시 리스트와 비교를 통해 중복된 해시가 발견되면 연관된 블록도 중복 처리하는 것이다. DBC_FS 기법은 다른 종류의 중복제거 알고리즘보다 블록을 나누는 기법이 간단하기 때문에 빠른 수행 성능을 보여준다.



(그림 2) 고정 블록 방식 중복 제거 기법

■ DBC_BS 알고리즘

DBC_BS 방식은 모든 중복데이터를 찾을 수 있는 방법이다. 먼저 연속 구간에 대해서 해시를 하고 동일한 해시가 있는지 찾는다. SHA1, MD5 같은 해시의 경우 연속 구간의 해시 성능이 좋지 않기 때문에 RabinFinger Print를 사용한다. 그러나 Rabin Fingerprint는 해시 충돌의 위험이 있기 때문에 동일한 해쉬가 발견되면 SHA1, MD5 같은 해쉬 함수를 사용해서 중복된 블록인지를 확인한다.

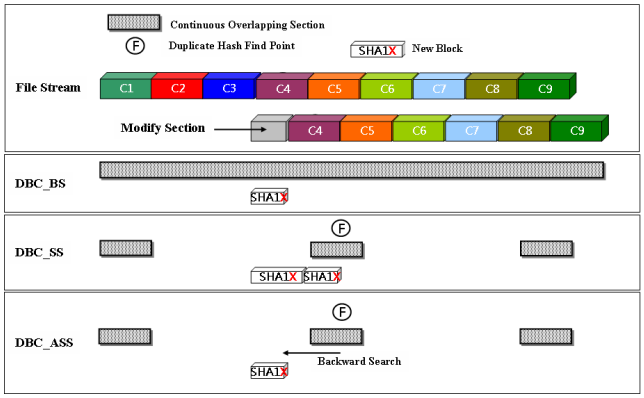
3.3. 개선된 중복 제거 알고리즘

■ DBC_SS 알고리즘

DBC_BS의 수행시간이 너무나 오래 걸리기 때문에 실제 대용량의 파일을 처리하는 스토리지 시스템에 적용하기 어렵다. 본 논문에서는 Duplicate Block Check with Stride Scheme(이하 DBC_SS)을 제안하고 있다. DBC_SS는 전체 구간의 블록을 중첩하는 대신 일정크기의 구간만을 중첩하여 줄어든 구간만큼 수행 성능을 향상 시키는 알고리즘이다. DBC_BS에 비해 중복제거 성능이 줄어들 수가 있지만 시스템에 맞게 슬라이드 크기를 조절한다면 성능을 DBC_BS 만큼 올릴 수 있다. 또한 연속적으로 중복이 있는 블록에 대해서 경우 중복 제거에 매우 효과적이며 중복이 없는 파일일 경우에도 DBC_BS 보다 빠른 수행속도를 보장 받을 수 있다.

■ DBC_ASS 알고리즘

Duplicate Block Check with Advance Stride Scheme (이하 DBC_ASS)은 DBC_SS에서 중복 블록을 찾았을 경우 스트라이드 블록들 중에 중복 블록이 존재할 수 있기 때문에 알고리즘 수행 도중 중복 블록을 만난다면 현재 블록의 위치에서 뒷부분으로 스트라이드 크기만큼 구간을 재탐색하는 것이다. 그러므로 스트라이드만 했을 때 보다 더 많은 중복을 발견할 가능성이 있으며 재탐색 비용 때문에 수행 시간이 늦어지게 된다.



(그림 3) Byte Shift 방법과 개선된 중복 제거 알고리즘

4. 구현 및 성능 평가

본 연구에서는 다음과 같은 환경에서 개발 및 실험하였다. 실험에 사용된 데이터는 리눅스 배포 데이터(Linux Distribution Data)가 사용되었다.

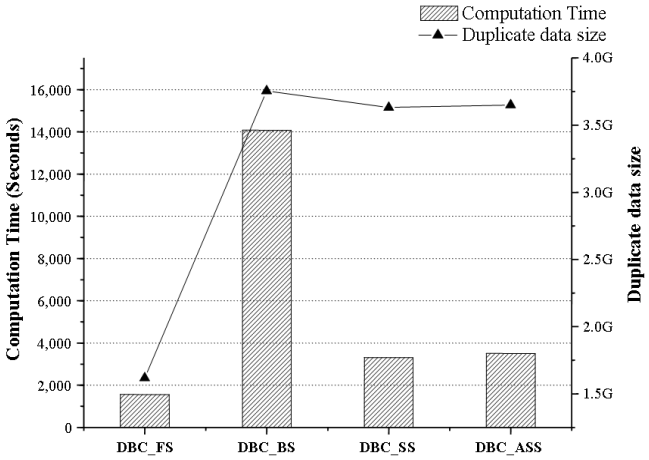
<표 1> 개발 및 실험 플랫폼

S/W 플랫폼	Client	운영체제	Window XP
		개발도구	Visual Studio 2008
	De-Duple. Server	운영체제	Fedora Core 9
		Kernel Version	2.6.18
		개발도구	gcc-4.0.2, mysql-4.1.2
H/W 플랫폼	CPU	Pentium 4 3.0 GHz	
	Memory	512 MB	
	Hard Disk	웨스턴 디지털 WD-1600JS(7200/8MB)	
	Network	LAN 100.0 Mbps	

리눅스 배포 데이터는 CentOS 5.2 i386 패키지[9] 데이터를 사용하였으며 CD 이미지 6장, DVD 이미지 1장으로 구성되었으며 총용량은 7,052,120,064 byte이다.

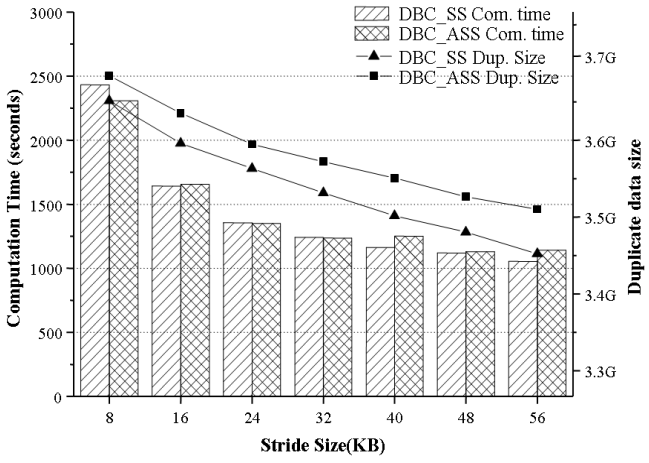
(그림 4)는 리눅스 배포 데이터를 중복제거 서버에 저장하였을 때 중복 제거 알고리즘별로 수행한 결과 그래프이다. 수행 속도에서는 DBC_FS 기법이 좋은 성능을 보여주고 있으며 중복 제거 성능에서는 DBC_BS 기법이 많은 중복을 발견하였다. 그러나 DBC_FS은 다른 기법보다 중복 제거 성능이 절반에도 못 미치는 결과를 보이고 있고 DBC_BS은 DBC_SS나 DBC_ASS (Stride Size : 8KB)의 기법보다 수행시간이 5배 이상 걸린다. DBC_SS와 DB

C_ASS 기법은 중복 데이터를 제거하는 성능이 DBC-BS와 거의 차이가 나지 않으면서 수행 시간은 DBC_FS에 근접하는 결과를 보여주고 있다. 따라서 리눅스 배포 데이터 같은 데이터를 중복 제거하여 저장할 때에는 DBC_SS, DBC_ASS의 기법을 사용하는 것이 효율적임을 알 수 있다.



(그림 4) 알고리즘별 리눅스 배포 데이터 중복제거 결과

(그림 5)는 DBC_SS, DBC_ASS의 기법에서 스트라이드 크기를 증가시키면서 관찰한 결과이다. 두 기법 모두 스트라이드 크기가 커질수록 중복 제거 성능이 떨어지지만 수행 성능이 향상되는 것을 볼 수 있다. 또한 중복 제거 성능도 급격히 감소하는 것이 아니라 선형적으로 감소하는 것으로 나타나고 있다. 수치적으로 따지자면 중복 제거 용량은 스트라이드 크기가 8일 때 와 56일 때 100MB 정도의 차이를 보이고 있으며 DBC_SS가 DBC_ASS 보다 수행시간이 조금 더 걸리고 50~70MB 크기만큼 중복을 더 찾아내고 있다.



(그림 5) 스트라이드 크기 변화에 대한 리눅스 배포 데이터 중복 제거 결과

5. 결론 및 향후연구

본 논문에서는 스토리지 시스템에서 효율적으로 이용할 수 있는 중복제거 기법에 대한 설계 및 구현을 기술하고 있다. 주요 쟁점은 해시를 사용하여 데이터 중복을 찾는 기법과 중복제거 시스템에서 해시 충돌을 견딜 수 있는지를 살펴보았으며 기존의 연구에서 제안하였던 중복제거 알고리즘과 비교/분석하였다.

본 논문에서는 DBC_BS 기법에 스트라이드 방식을 넣어 수행속도의 성능을 대폭 향상 시킬 수 있었으며 중복제거 성능에서도 다른 기법보다 높은 중복탐색을 할 수 있었다. 중복 제거 알고리즘을 비교하기 위해 여러 중복제거 기법을 수행할 수 있도록 스토리지 시스템을 설계하였다. 제안된 시스템의 유용성과 중복제거 알고리즘의 성능의 비교를 위하여 리눅스 배포 데이터의 저장 실험을 수행하였으며 실험 결과 DBC_SS, DBC_ASS가 효율적인 기법임을 확인하였다.

향후 연구로는 제안된 기술의 성능을 개선하기 위해 다양한 압축 기법을 적용하고 알고리즘을 개선하여 수행속도를 더욱 향상시키며 다양한 데이터 집합에 대한 실험을 수행하여 적용할 수 있는 응용에 대한 연구를 수행할 계획이다.

참고문헌

[1] J.S. Robin and C.E. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In Proceedings of the 9th USENIX Security Symposium, Denver, CO, August 2000.

[2] A. Tridgell. Efficient algorithms for sorting and synchronization. PhD thesis, The Austrailian National University, 1999.

[3] plan9 home page, http://plan9.bell-labs.com/plan9/

[4] QUINLAN, S., AND DORWARD, S. "Venti: a new approach to archival storage," In Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST), 2002.

[5] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A Low-Bandwidth Network File System. In Proceedings of the Symposium on Operating Systems Principles (SOSP'01), pages 174 - 187, 2001.

[6] D. Bobbarjung, Suresh Jagannathan, C. Dubnicki. Improving Duplicate Elimination in Storage Systems, ACM Transactions on Storage, November 2006.

[7] K. Eshghi and H.K. Tang . A Framework for Analyzing and Improving Content-Based Chunking Algorithms. Hewlett-Packard Labs Technical Report TR 2005-30

[8] Fred Douglis and Arun Iyengar. Application-specific Delta-encoding via Resemblance Detection. In Proceedings of 2003 USENIX Technical Conference, pages 113 - 126, San Antonio, Texas, USA, 2003

[9] centos home page, http://www.centos.org/