파일 지문으로 중복 파일을 제거한 클러스터링 백업 시스템 설계

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

Design of Deduplication Supported Clustering Backup System using File Finger Printing

Ho-Min Jeong*, Young-Woong Ko
Dept. of Computer Science, Hallym University

요 약

기존의 백업시스템에서는 데이터의 중복을 고려하지 않고 백업 데이터 전부를 저장하기 때문에 저장 용량 공간을 많이 차지하는 문제점이 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 백업 데이터의 각 파일에 대해서 일정한 크기의 블록 단위로 파일지문을 부여하고 파일지문이 동일할 경우 하나의 사본만을 공유하는 방법으로 백업 데이터의 용량을 감소시키는 방법을 도입하였다. 제안하는 백업 시스템은 중복되어 발생하는 데이터에 대해 하나의 사본만 백업함으로써 백업되는 데이터의 양을 효과적으로 감소시켰다. 또한 백업되는 파일 블록에 대해서 클러스터링 기술을 사용함으로서 입출력 성능 향상을 고려하였다.

1. 서론

UCC(User Created Content)의 발전, Web 2.0으로 확장, 고용량 동영상 데이터의 증가로 인해 정보생산량이 급격히 증가하고 있다. IDC보고서에 의하면 2010년 생산되는 디지털 정보량이 1조 기가바이트에 육박할 것이라고 보고하고 있다[1]. 이는 정보생산량이 사용할 수 있는 스토리지 용량을 능가하고 있는 것이다.

특히 디지털 정보의 백업 데이터가 스토리지 용량의 대부분을 차지하고 있으며 데이터의 백업은 대형 컴퓨터를 운영하는 대규모 사업체에서는 물론 개인 컴퓨터에서도 필수적인 사항이다[2][3]. 그러나 대부분의 백업은 데이터의 사본을 다른 공간에 저장하는 형태라서 저장 공간 사용의 효율이 많이 떨어진다. 이에 저장 공간을 효율적으로 사용하는 백업이 필요하다. 빠른 백업과 복구를 제공하는 백업방식이 필요하다.

본 논문은 파일들의 중복을 제거할 뿐만 아니라 파일을 블록 단위로 나눈 객체의 중복까지 제거해 기존의 백업 시스템에 비해 적은 저장 공간으로 방대한 양의 백업 데이터를 관리할 수 있는 방법을 제안하였으며 빠른 백업과 복구 성능을 위해 클러스터링 구조의 백업 시스템을 설계하였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 데이터 저장기술과 백업 기술에 대한 관련연구를 알아보고, 3장에서는 제안하는 파일 백업 시스템에 대해 시스템 구조에 대해서 4장에서는 백업 및 복구 처리 방법을 설명한다. 마지막으로 5장 에서는 결론을 맺고 향후 연구방향을 제시한다.

This research was supported by the Program for the Training of Graduate Students in Regional Innovation which was conducted by the Ministry of Commerce Industry and Energy of the Korean Government,

2. 관련연구

2.1. backup 방식

지금까지 구현된 상용 백업 소프트웨어는 각각의 소프트웨어마다 다양한 형태의 백업 기능을 제공하고 있다[8]. 백업은 형태, 기능, 대상에 따라서 백업 방식을 구분하고 있다. 모든 파일을 백업할 것인지 또는 변경된 부분만을 백업할 지에 따라 전체 백업과 부분 백업으로 나뉜다. 전체 백업은 수행할 파일시스템이나 장치 내에 존재하는 모든 파일에 대하여 백업을 수행하는 것이고 부분 백업은 이전 백업에 대해서 변경된 파일만 백업을 수행한다. 부분 백업은 이전의 마지막 전체 백업 이후에 변경된 데이터를 백업하는 차등 백업과 바로 이전의 백업 이후에 변경된 데이터를 백업하는 점진 백업으로 나뉜다.

백업할 대상이 파일 시스템 기반인지 장치 기반인지로 구분할 수 있는데 파일 시스템 기반의 백업은 파일 구조를 파악하여 백업장치에 파일과 디렉터리 구조를 저장하기 때문에 백업은 오래 걸리지만 복구은 빠르게 수행할 수 있다. 이에 반해 장치 기반의 백업(device-based back up)은 파일구조를 무시하고 디스크 블록을 백업장치에 기

록한다. 따라서 백업 성능은 향상되지만 복구시간이 오래 걸리는 단점이 있다.

[표 1] 데이터 저장 기술 비교

	SAN	NAS	CAS	OSD
메타데이타 관리	응용 프로그램	응용 프로그램	메타데이터 서버	메타데이터 서버
데이터 유형	가변	가변	고정	고정
저장 데이터	블록	파일	객체	객체
데이터 중복	발생	발생	발생하지 않음	발생
데이터 주소	Location	Location	ID	ID

[표 2]는 데이터 저장 기술의 특징을 비교한 표이다. 데이터 저장 기술로 객체 기반 스토리지 장치(Object-based Storage Device : OSD)와 콘텐츠 기반 스토리지 장치(Content Addressed Storage : CAS), NAS(Network Attached Storage)와 SAN(Storage Area Network)이 있다[1][4][5]. NAS와 SAN은 스토리지의 확장성과 데이터의 빠른 전송이 가능하다. 최근의 기술인 CAS와 OSD는 NAS와 SAN이 구현 하는 기능 외에 기존의 파일 기반 저장장치와 달리 데이터를 일정한 입출력 크기로 나눈 후, 나눈 데이터의 개개별로 고유 ID를 부여하여 객체 단위로 저장장치에 저장하고, 접근하는 기술을 지녔다. 데이터를 객체 단위로 저장하면 스토리지로부터 객체에 직접 접근함으로써 시스템 성능을 향상시킬 수 있으며, 시스템 성능의 저하 없는 확장을 제공할 수 있다. 또한 객체에 대한 메타데이터에 독립적인 객체 접근 인터페이스를 사용함으로써, 이 기종 플랫폼에서 객체를 안전하게 공유할 수 있는 특성을 제공할 수 있다[9].

2.2 Finger Printing

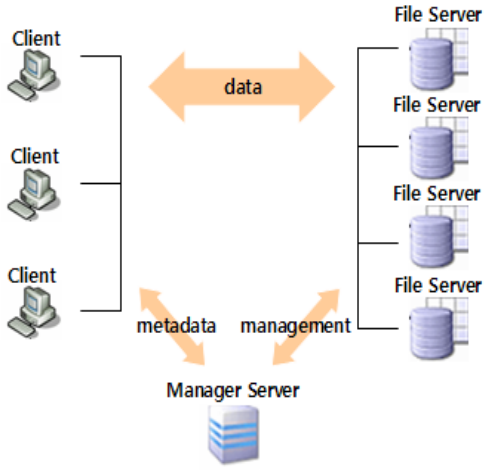
파일이나 객체를 파일지문으로 만들어 중복을 방지하는 효과를 나타낼 수 있다[6][7]. 일련의 메시지를 축약된 비트로 만드는 MD5, SHA-1 같은 널리 알려진 해시 알고리즘이 개발되어 있다. 서로 다른 두 개의 파일의 지문이 일치할 확률이 얼마인지 계산해서 특정 시스템에 맞는 해시 함수를 사용해야 할 것이다. 생일 파라독스 문제의 해법은 최소 2개의 서로 다른 파일의 지문이 일치할 확률을 계산할 수 근거가 되다.

$$\epsilon \approx 1 - e^{-\frac{1}{M}\frac{N(N-1)}{2}}$$

다음의 식은 해시 함수가 고르게 분포한다는 가정에서 입력데이터의 수와 표현할 수 있는 해시의 수를 통해서 지문이 일치할 확률을 나타내는 식이다. M은 해시의 크기 N은 객체의 수로 표현된다. 예를 들어 시스템의 전체 용량이 1Exa Byte ($=2^{60}$)이고 각 객체들은 8Kbyte ($=2^{13}$)라고 가정하면 시스템 전체가 저장할 수 있는 객체의 수는 ($=2^{47}$) 이다. M의 크기에 MD5, SHA-1 해시 함수의 축약 데이터 수(2^{128}, 2^{160})를 대입하고 N의 크기에는 객체의 수(2^{47})를 대입하여 계산하면 확률 ϵ는 10^{-10}, 10^{-17}의 이하의 값을 갖는다. 이는 충돌이 일어날 확률을 무시해도 될 정도의 값이다. 만약 객체의 수가 많아 파일 지문의 충돌이 일어날 때에는 해시함수의 축약 데이터가 큰 SHA-256(2^{256}), SHA-384(2^{384})등의 확장된 해시 함수의 이용방안을 모색할 수 있다. 축약 데이터가 큰 해시 함수를 쓸 경우에는 추가적인 오버헤드가 생길 수 있다. 본 논문에서는 파일 지문을 얻어낼 함수로 SHA-1를 사용한다.

3. 백업 시스템 구조 설계

본 논문에서 제안하는 시스템은 여러 사용자가 사용하는 데이터의 중복이 매우 높음을 염두 하였다. 기존의 백업시스템에서는 데이터의 중복을 고려하지 않고 백업 데이터 전부를 저장하기 때문에 저장용량 공간의 증가가 많은 문제점이 있다. 이러한 문제점을 해결하기 위해 백업 데이터의 각 파일들에 파일지문을 부여하고 파일지문이 동일할 경우 하나의 사본만을 공유한다. 따라서 중복되어 저장되는 데이터의 양을 효과적으로 감소 시켜 저장용량의 효율을 향상시킨다. 시스템은 크게 세 부분으로 나뉘는데 [그림 1]과 같이 매니저 서버에서 클라이언트의 작업 요청을 처리하고 파일서버의 관리를 함으로서 관리자 역할을 맡는다. 클라이언트와 파일서버에서 오버헤드가 큰 데이터송수신을 처리함으로서 매니저 서버에 부하를 줄일 수 있다.

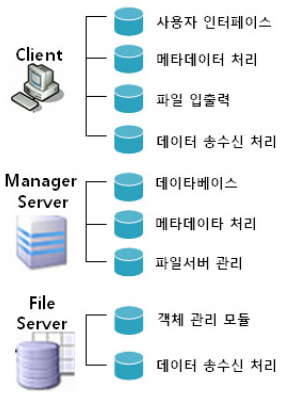


[그림 1] 전체 시스템 흐름도

3.1. 구성 모듈

백업 시스템의 모듈 구성은 [그림 3]과 같다. 크게 클라이언트, 매니저 서버, 파일 서버의 세부분으로 나뉘고 각 부분의 모듈들은 서로 연관되어 있다.

클라이언트 부분에서는 사용자 인터페이스, 메타데이터 처리, 파일 입출력, 데이터 송수신 처리를 담당하는 모듈로 구성되어 있다. 사용자 인터페이스 모듈에서는 사용자가 요청하는 백업과 복구 작업을 받아들이며 진행 사항 알람과 부가적인 기능을 수행하는 역할을 한다.

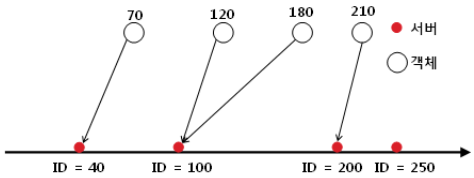


[그림 2] 구성 모듈

클라이언트의 메타데이타 처리 모듈은 파일시스템의 전반적인 정보를 저장한 메타데이타를 생성하고 매니저 서버에 요청한 메타데이타를 파싱하며 파일지문, 객체지문의 데이터들을 매니저 서버와 송수신을 처리한다. 파일 입출력 모듈에서는 파일을 객체단위로 나누고 파일지문과 객체지문을 만드는 작업과 복구 작업 시 수신한 객체들을 파일로 조합한다. 클라이언트의 데이타 송수신 처리 모듈은 여러 파일서버와 동시에 송수신을 하며 파일 입출력 버퍼에 저장된 객체를 처리한다.

매니저 서버는 데이터베이스, 메타데이타 처리, 파일서버 관리 모듈로 구성되어 있다. 데이터베이스 모듈은 백업된 파일시스템의 정보를 설계한 스키마에 맞게 저장시키는 저장소 역할이다. 매니저 서버의 메타데이타 처리 모듈은 클라이언트에서 보내온 파일시스템 정보가 담겨있는 메타데이타를 데이터베이스에 저장하거나 파일지문, 객체지문들이 들어있는 메타데이타의 중복여부 체크한다. 파일서버 관리 모듈은 파일서버들의 용량정보, IP주소 등의 정보를 업데이트하여 관리를 돕는 역할을 한다.

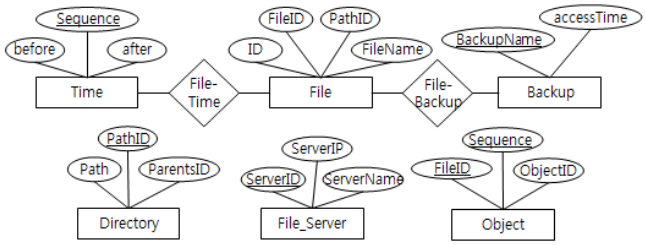
파일 서버는 객체 관리 모듈, 데이터 송수신 모듈로 이루어져 있으며 각각의 서버는 망을 구성할 때 임의의 ID를 가지게 된다. [그림 3]과 같이 객체는 자신의 지문 서버의 ID가 일치하거나 지문보다 작은 서버 ID중에 제일 큰 ID의 파일서버에 블록을 저장한다. 이는 파일서버의 추가나 제거로 인한 데이터 이동이 생길 경우에도 혼란이 없이 매니저 서버에 저장된 메타데이타를 그대로 사용이 가능하다.



[그림 3] ID의 할당에 따른 객체의 분배

3.2. 메타 데이타 저장소

본 논문에서는 백업 데이터의 메타데이타 관리를 위해 데이터베이스 사용을 고려하였다. 데이터베이스 사용으로 중복 파일 지문의 검색이 용이하며 여러 가지 기능의 설계가 가능해 지는 이점이 있다. [그림 4]는 E-R 다이어그램이다. 핵심적인 메타데이타 값을 표현할 수 있고 기능의 확장이 용이하다.



[그림 4] MetaData Server DB 스키마

저장소는 몇 가지 테이블로 구성되어 있으며 각 테이블의 역할과 속성은 다음과 같다.

File(ID, FileID, PathID, FileName)

File 테이블은 세 가지 속성을 나타내고 있다. File_ID값은 파일지문을 나타내며 SHA-1 함수를 이용하여 나타낸 값이고 시스템에서 사용하는 모든 지문들은 SHA-1 해시함수를 이용하였다. Path_ID 값은 파일의 부모 디렉토리의 절대경로를 해시한 값으로 디렉토리 테이블의 시스템의 경로를 참조할 때 사용한다. File_Name은 파일이름 값이며 이는 파일의 기본적인 속성이다. File_ID는 같지만 경로가 다를 경우가 생길수가 있는데 이는 Path_ID를 키로 넣어 파일 구조를 저장가능하게 한다. File 테이블에는 파일의 여러 가지 의미 있는 속성들이 추가로 삽입가능하다.

Backup(BackupName, accessTime)

백업이름은 사용자의 백업 데이터를 구분하며 접속날짜 등을 추가로 정의하였다.

Directory(PathID, ParentID, Path)

디렉토리 테이블은 사용자들이 사용하는 모든 경로를 저장하고 있다. PathID는 절대경로를 해시한 값이어서 단일값을 갖는다. ParentID의 값은 PathID와 1:1 대응하면서도 PathID 값이기 때문에 상위 PathID를 추적하면 RootPath에 도달하여 전체경로를 표현할 수 있다.

Time(Sequence, before, after)

사용자가 데이터를 복원 시 날짜를 매개로 요청을 하기 때문에 파일 하나하나의 존속기간을 인식해야 하며 존속형태는 [표2]처럼 기간의 묶음으로 볼 수 있으며 이를 테이블로 표현할 수 있는 것이다.

[표 2] Time Table

Sequence	before	after	File_hash
1	2007.6.23	2007.6.30	1084214
2	2007.7.20	2007.8.30	1084214
3	2007.9.10	2007.9.30	1084214

File_Server(ServerID, ServerIP, ServerName)

ServerID는 파일서버수를 고려하여 160bit 값을 균등하게

분포한 값으로 파일서버의 ID이다. 기타의 값은 파일 서버에 접속정보를 기술한다.

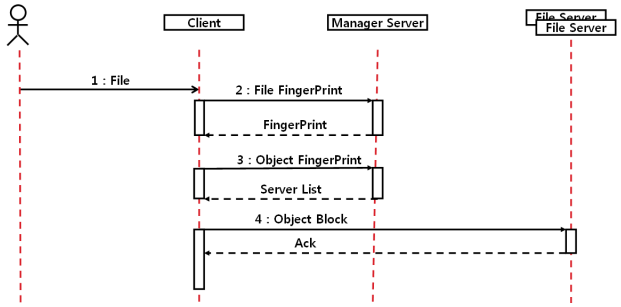
Object(FileID, Squence, ObjectID)

파일들은 8Kbyte, 16Kbyte등의 객체로 나뉘어 있으며 이들을 지문으로 만들어 보관한다. File_ID에 해당하는 파일 블록의 재현을 위해 지문 값과 지문의 순서를 저장한다.

4. 백업, 복구 처리과정

4.1. 백업 처리

[그림 5]는 파일의 백업 처리 과정이다. 사용자는 매니저 서버에 디렉토리를 저장한 상태이고 파일서버가 다수 존재하고 매니저 서버에서는 이를 온전히 관리하는 상태라고 가정한다.



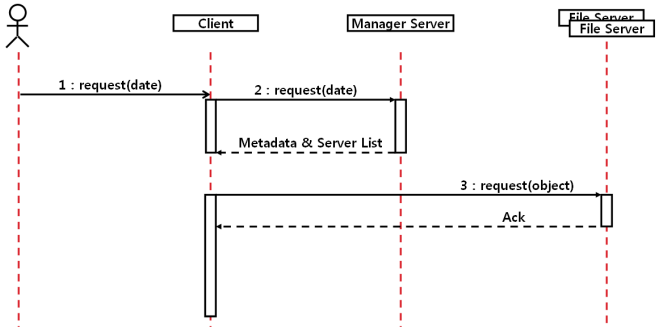
[그림 5] 파일의 백업 처리

1. 사용자는 백업을 할 파일을 선택하였으며 클라이언트는 선택한 파일을 SHA-1 해시 함수를 이용해 파일지문을 생성한다.
2. 생성한 파일지문을 클라이언트에서 매니저 서버에 전송을 하고 매니저 서버는 파일지문의 중복여부를 체크하여 중복된 파일 지문정보를 클라이언트에 전송한다.
3. 클라이언트는 중복이 아닌 파일을 객체 단위로 나눈다. 나누어진 객체를 SHA-1 해시 함수를 이용해 객체 지문을 만들고 지문 정보들을 매니저 서버에서 수신하여 저장한다. 매니저 서버에서 저장을 완료하면 클라이언트에 파일서버 목록을 전송한다.
4. 클라이언트는 파일서버의 ID에 따라 객체 블록을 객체 지문과 연관 있는 파일서버에 전송한다. 파일 서버는 객체 블록의 객체지문을 매개로 중복된 객체가 있는 지 체크하고 중복 값이 존재하면 저장을 피한다. 모든 객체 블록을 파일 서버에 전송을 하면 백업 처리를 마치게 된다.

4.2. 복구 처리

[그림 6]는 파일의 복구 처리 과정이다. 사용자는 백업 처리를 여러 번 수행한 상태이고 파일 서버에 객체가 온전히 저장된 상태라고 가정한다.

1. 사용자는 복원하고 싶은 날짜를 선택하여 클라이언트에 복원 요청을 한다.
2. 클라이언트는 매니저 서버에 날짜를 매개로 해당 메타데이타의 정보를 요청한다. 서버는 날짜를 포함하는 기간에 저장한 파일의 리스트를 만들고 해당하는 객체 지문과 서버리스트를 클라이언트에 전송한다.
3. 클라이언트는 객체지문의 블록이 저장된 서버ID들에 블록의 전송요청을 하고 서버에서 송신한 블록의 전송이 끝나면 객체들을 조합하여 올바른 파일로 복구를 한다. 파일들의 조합이 끝나면 복구 처리를 마치게 된다.



[그림 6] 파일의 복구 처리

5. 결론 및 향후 연구

본 논문에서는 파일과 객체의 지문을 생성해 지문이 같은 데이터는 하나의 사본만을 유지해 데이터 중복을 제거하는 방법을 제안하였다. 여러 파일서버를 두어 객체를 저장하는 방법, 메타데이타 저장소 유지에 필요한 스키마와 백업과 복구에 대한 처리과정을 설계 하였다.

향후 연구방향으로 저장 공간을 보다 효율적으로 이용하기 위해 백업 데이터를 압축하여 저장하는 방법과 입출력 성능 향상을 위해 여러 대의 매니저 서버를 구성해보고 어느 특정 파일에 접근이 많아 특정 파일서버에 부하가 걸릴 경우 부하를 분배하는 연구를 진행할 예정이다.

참고문헌

[1] http://korea.emc.com/
[2] Storage Networking Industry Association, Backup/Recovery Tutorial, 2001
[3] http://www.microsoft.com/korea/technet/security/guidance/secmod201.asp
[4] M. Mesnier, G. R. Ganger, and E. Riedel, "Object-Based Storage," IEEE Communications Magazine, pp.84-90, 2003.
[5] W. C. Preston, Using SANs and NAS, O'REILLY, 2002.
[6] R. L. Rivest, "The MD5 Message Digest Algorithm, " Request for Comments(RFC) 1321, Internet Activities Board, 1992.
[7] National Institute of Standards and Technology. Secure hash standard. FIPS Publication #180-1, April 1997.
[8] W. Curtis Preston and Gigi Estabrook, "UNIX Backup and Recovery," O'REILLY, 730 pages, 1st Edition November 1999.
[9] Alain Azagury, Vladimir Dreizin, “Towards an Object Store ” Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies, pp.165-178, 2003.