SAN 논리볼륨 관리자를 위한 매핑 및 자유공간 관리기법의 설계 및 구현

남상수°, 홍현택, 피준일, 송석일, 유재수
충북대학교 정보통신공학과 및 컴퓨터정보통신연구소
{ssnam°, hongry, pji, prince}@netdb.chungbuk.ac.kr, yjs@cbucc.chungbuk.ac.kr

Design and Implementation of Mapping and Freespace Management for SAN Logical Volume Manager

Sang Su Nam°, Hyun Taek Hong, Jun Il Pee, Seok Il Song, Jae Soo Yoo
Dept. of Computer and Communication Engineering, Chungbuk National University

요 약

최근 폭발적으로 증가하는 대량의 정보를 효율적으로 관리하기 위해서 서버에 개별적으로 연결되던 저장장치를 고속의 전용 네트워크에 직접 연결하고 이를 여러 서버가 공유하는 SAN(Storage Area Network)이 등장했다. SAN을 보다 효과적으로 활용할 수 있도록 SAN 운영체제들은 여러 저장장치를 하나의 커다란 저장장치로 보이게 하는 저장장치 가상화를 지원한다. 저장장치 가상화의 핵심적인 역할을 하는 것이 바로 논리볼륨 관리자이다. 논리볼륨 관리자는 논리주소 블록을 실제 디스크의 물리블록으로 매핑시키는 매핑 관리자를 통해 가상화를 실현한다. 이 논문에서는 효율적인 논리볼륨 관리자를 위해 매핑 관리자와 자유공간 관리자의 기능 향상에 대한 방법을 찾고 이를 반영한 매핑 관리자와 자유공간 관리자를 설계하고 구현한다. 더불어 이 논문의 매핑 관리기법은 특정 시점의 볼륨이미지를 유지할 수 있는 스냅샷과 시스템을 정지시키지 않고 SAN에 디스크를 추가할 수 있는 온라인 재구성 기능을 지원한다.

1. 서론

폭발적으로 증가하는 대량의 정보를 효율적으로 공유하고 고속으로 서비스하기 위하여 데이터 화일을 중심으로 하는 새로운 개념의 컴퓨터 시스템 환경이 도래하였다. 이에 서버에 개별적으로 연결되던 저장 시스템을 파이버 채널과 같은 고속의 전용 네트워크에 직접 연결하여 중앙 집중적인 저장 시스템의 관리가 가능하게 하고, 서버를 거치지 않고 네트워크에 연결된 저장장치를 직접 접근할 수 있는 SAN이 등장했다. SAN은 서버 중심의 시스템 환경이 가지는 문제점을 해결해 줄 수 있는 최선의 방책으로 인식되고 있다. SAN 환경을 보다 효과적으로 활용할 수 있기 위해서는 SAN에 대한 사용자 관점의 뷰를 제공하는 것이 필요하다. 이를 위해 대부분의 SAN 운영 S/W는 여러 물리적 디스크들을 사용자 관점에서 바라볼 수 있는 가상 디스크 형태로 추상화 시켜주는 저장장치 가상화 개념을 지원한다. 이러한 저장장치 가상화의 핵심적인 역할을 하는 것이 바로 논리볼륨 관리자이다.

논리볼륨 관리자는 사용자들이 특정 논리주소 블록에 대한 I/O를 요구하면 적절히 요구한 논리블록을 실제 디스크 내의 물리블록으로 매핑해서 I/O를 수행한다. 매핑수행 과정에서 필요에 따라 논리볼륨내의 실제 디스크의 각 블록의 사용유무를 관리하는 자유공간 관리자가 논리블록에 적절히 물리블록을 할당해서 매핑을 돕게된다. 또한 논리 볼륨 관리자는 특정 시점의 볼륨이미지를 유지할 수 있는 스냅샷과 디스크를 추가할 때 시스템을 정지시키지 않고 SAN에 디스크를 추가할 수 있는 온라인 재구성 기능을 제공해야 한다. 이 논문에서는 Linux상의 SAN 볼륨관리자인 pool[1]을 기반으로, 효율적인 논리볼륨 관리자를 위한 매핑 및 자유공간 관리기법을 설계하고 구현한다. 또한 이 논문의 매핑 관리 기법은 스냅샷과 온라인 재구성 기능을 지원한다.

이 논문의 구성은 다음과 같다. 2장에서는 볼륨 관리자의 매핑 관리와 자유공간 관리에 대한 기존 연구에 대해 기술한다. 3장에서는 매핑 관리자와 자유공간 관리자를 설계할 때 고려해야할 사항을 제시하고 이에 따른 설계 내용을 기술한다. 4장에서는 3장에서 설계한 매핑 및 자유공간 관리자의 구현에 대해서 설명하고 5장에서 결론을 맺는다.

※ 본 연구는 한국전자통신연구원 2001년도 위탁과제 연구비 지원으로 수행되었음

2. 관련연구

기존의 논리볼륨 관리자의 매핑 방법을 살펴보면 크게 수식 기반의 매핑 방법과 매핑테이블 기반의 매핑 방법으로 나누어진다. 수식 기반의 매핑 방법은 단순하여 매핑 요구를 빠르게 처리할 수 있지만, 논리볼륨 재구성과 같이 매핑 관계가 변하는 상황에 유연하게 대처하지 못한다. 반면에, 매핑테이블 기반의 매핑 방법은 수식 기반보다 매핑 요구 처리가 느리고 SAN과 같은 대용량 환경에선 매핑 테이블의 사이즈가 커서 관리하기가 어렵지만, 위와 같이 매핑 관계가 변하는 상황에 유연하게 대처할 수 있는 장점이 있다. 이 장에서는 기존의 논리볼륨 관리자에서 지원하고 있는 매핑 방법과, 기존의 화일 시스템에서 이용했던 자유공간 관리 기법에 대해 조사/분석한 내용을 기술한다.

2.1. 매핑 기법

미네소타 대학에서 개발된 SAN 볼륨관리자인 pool은 별도의 매핑 테이블을 두지 않고 수식 기반의 매핑을 수행한다. 수식 기반의 매핑 방법에서는 매핑 관계가 변하는 볼륨 재구성에 유연하게 대처하지 못하므로 pool의 재구성은 디스크의 추가시 논리볼륨의 용량이 추가될 뿐, 디스크의 부하 균등을 위한 디스크간 데이터 이동이 전혀 고려되지 않았다.

Petal[2]은 네트워크로 연결된 여러 호스트에 부착된 디스크를 가상화하여 하나의 논리 디스크로 클라이언트에게 제공하는 시스템이다. Petal에서는 가상화를 매핑테이블 기반의 매핑 방법을 이용하고 있으며, 온라인 백업을 위한 스냅샷과 볼륨 재구성을 지원하고 있다. 하지만 Petal은 네트워크를 통해 연결된 여러 서버가 가지고 있는 저장장치들을 가상화 해주는 시스템이며 SAN과 같은 환경에 적합한 볼륨관리자는 아니다.

2.2. 자유공간 할당 정책

조사에 의하면 기존에 논리볼륨 관리자를 위한 자유공간 할당 기법에 대한 설명을 하고 있는 참고문헌은 없다. 차이는 있지만 기존의 파일 시스템에서 행해지는 자유공간 관리 기법을 분석해 논리볼륨 관리자에 도입할 수 있는 방법을 찾아 보았다. 자유공간 관리 정책은 효율적으로 공간을 할당/해제하여 화일 시스템의 단편화를 최소화하는 것이 목적이다. 기존 화일 시스템의 자유공간 할당 정책으로 여러 가지가

소개되었다. 대표적인 것으로 선형 리스트 구조를 이용한 순차적 적합, 트리 구조를 활용한 색인 적합, 비트맵 구조를 사용한 비트맵 적합등이 있다[3]. 이러한 정책중에 GFS[1]나 EXT2[4]와 같은 파일 시스템은 비트맵 기반의 자유공간 관리를 하고 있다. XFS[5]나 Fujitsu사의 SafeFILE 제품의 파일 시스템인 HAMFS[6, 7]의 경우는 B-tree를 활용한 색인 적합 방법을 사용하고 있다.

3. 매핑 관리자 및 자유공간 관리자 설계

3.1. 매핑 관리자의 설계

이 논문에서는 매핑 관리자를 설계하기 위해 매핑테이블의 관리 방법과 매핑테이블의 표현 방법을 주로 고려한다. 매핑테이블 관리 방법이란 SAN 환경에서 다중호스트가 매핑테이블을 접근해서 변경 및 읽기를 할 때 어떻게 이를 효과적으로 수행할 것인가 관한 것이다. 매핑테이블의 표현 방법이란 매핑테이블의 구조를 어떻게 할 것인가를 결정하는 것이다. 일반적으로 성능향상을 위해서 매핑테이블을 하나의 디스크에 모두 저장하지 않고 분할저장해서 부하를 분산하게 된다. 이때 분할 방법과 테이블의 엔트리의 표현 방법에 따라서 전체 볼륨 관리자의 성능에 영향을 미치게 된다. 다음부터 이 두가지 고려사항에 대해 기술하고 이 논문에서는 어떤 방법을 취했는지 설명한다. 스냅샷과 볼륨 재구성 기능에 대한 설계 내용도 같이 설명한다.

3.1.1. 매핑테이블의 관리 방법

이 논문에서 설계하는 매핑 관리자는 테이블 기반 방법을 기본으로 한다. 테이블 기반의 매핑 관리자는 매핑테이블을 디스크에 저장하고 매핑을 수행하거나 매핑테이블의 내용을 변경할 때 디스크의 해당 블록을 접근한다. 여기서 생각해 보아야 할 점은 SAN에 부착된 다중 호스트들이 매핑테이블을 어떻게 접근해서 변경 및 읽기를 수행하는 가이다. I/O를 처리하기 위해서는 반드시 매핑테이블을 접근해야 하므로 매핑테이블의 변경 및 읽기를 어떻게 하느냐에 따라서 I/O 성능에 중요한 영향을 미치게 된다. 다중 호스트가 볼륨을 공유하는 상황에서 매핑테이블을 저장/관리하는 방법을 분류해보면 다음과 같이 세가지 정도로 분류해 볼 수 있다.

- 매핑테이블을 포함한 볼륨 관리에 필요한 메타데이터만 관리하는 메타 서버를 둔다.
- 볼륨 관리에 참여하는 모든 호스트가 매핑테이블 전체를 중복해서 관리한다.
- 볼륨 관리에 참여하는 호스트들이 매핑테이블을 분할 관리한다.

이 논문에서는 마지막 방법을 택한다. 일반적으로 SAN 환경에서 볼륨 관리에 참여하는 호스트들 사이에는 범용 네트워크인 LAN을 이용하지 않고 전용 네트워크를 이용하므로 통신으로 인한 매핑 지연이 크지 않을 것이고 매핑을 위해 주고받는 데이터의 양이 20Bytes 내외로 매우 작다. 또한 마지막 방법은 다른 방법에 비해 훨씬 구현 및 관리가 간단하다.

3.1.2. 매핑테이블 표현 방법

일반적으로 성능향상을 위해서 매핑테이블을 하나의 디스크에 모두 저장하지 않고 분할저장해서 부하 분산을 시도한다. 이때 분할 방법과 테이블의 엔트리의 표현 방법에 따라서 전체 볼륨 관리자의 성능에 영향을 미치게 된다. 이 논문에서는 매핑테이블의 표현 방법으로 네가지를 고려해 보았다.

첫 번째로 고려해 볼 수 있는 것은 그림 3-1과 같은 매핑테이블의 가장 기본적인 표현 방법이다. 즉, (device, sector)로 이루어지는 물리주소들이 대응되는 논리주소의 순서에 따라서 배열형태를 이루는 것이다.

두 번째 방법은 매핑테이블을 분할할 때 물리주소를 기준으로 분할한다. 즉, 특정 디스크에 저장되는 매핑 정보들은 그 디스크 내의 물리 블록들에 대한 매핑 정보만을 저장한다.

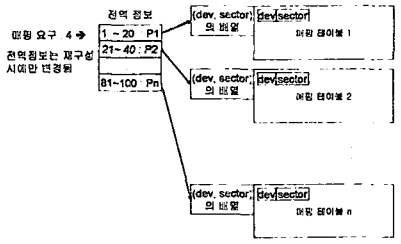

그림 3-1 매핑테이블의 분할 관리

세 번째 방법은 두 번째 방법과 논리주소의 희소분할 방법을 적절히 혼합한 형태를 고려할 수 있다.

마지막 네 번째 방법으로 논리주소를 기준으로 매핑테이블을 분할하되 매핑 정보를 모든 논리블록에 대해서 테이블에 기록하는 것이 아니고 범위를 이용해서 기록하는 것을 생각해 볼 수 있다.

이 논문에서는 첫 번째 방법과 마지막 방법을 구현한다. 마지막 방법의 경우는 연속적인 논리주소의 매핑요구가 발생할 때 효과적이다.

3.1.3. 스냅샷의 설계

스냅샷 기능은 사용자가 원하는 시점의 볼륨이미지를 유지하여 추후에도 스냅샷 당시의 데이터를 참조하고자 할 경우 사용된다. 일반적으로 스냅샷을 생성할 때 스냅샷이 생성되는 원래 볼륨이 가지고 있는 매핑테이블을 그대로 스냅샷 목적으로 사용하기 위해 복사한다. 그리고 이를 유지하기 위해서 COW(copy on write)를 이용한다. 앞에서 언급한 것처럼 SAN과 같은 대용량 환경에선 매핑테이블의 사이즈가 매우 크므로 이를 복사하는데 상당한 시간이 소요된다. 스냅샷을 생성하는 동안에는 다른 I/O 연산을 허용하지 않으므로 스냅샷 생성시간이 길어지는 것은 문제가 있다.

이에 대한 대안으로 스냅샷 생성중에 매핑테이블을 복사하지 않고 원래 볼륨의 매핑테이블을 공유하면서 COW에 의해서 변경되는 부분만 해쉬테이블에 저장하는 방법을 생각해 볼 수 있다. 이 방법을 이용하면 스냅샷을 생성하는 시간이 상대적으로 매우 짧고 초기의 저장공간 사용량이 작다. 하지만 스냅샷 이후에 볼륨의 모든 부분이 변경이 되면 오히려 매핑테이블을 복사하는 것 보다 저장공간을 더 소요하게 될 수 있다. 이 외에도 스냅샷 데이터를 접근할 때 매핑테이블과 해쉬테이블을 모두 접근해 보아야 하므로 스냅샷 데이터를 접근하는데 상대적으로 시간이 더 소요될 수 있다.

3.1.4. 볼륨 재구성 방법 설계

재구성 방법의 설계 내용에 대해서 설명하기 전에 이후의 재구성방법은 모두 스트라이핑 볼륨을 기준으로 설명한다. 다른 레이드 레벨의 볼륨들은 스트라이핑 볼륨의 재구성 방법을 그대로 또는 조금 수정해서 바로 적용할 수 있다. 그리고 디스크의 단편화가 일어나지 않도록 재구성 한다는 전제하에 수행한다.

온라인 볼륨 재구성이란 디스크를 추가하거나 제거할 때 시스템을 정지시키지 않고 변경된 내용을 포함하도록 논리볼륨을 재구성하는 것을 말한다. 이 논문에서 설계하는 볼륨 재구성 방법은 세가지로 나눠볼 수 있다.

첫 번째 방법은 기존 디바이스의 모든 데이터를 완전히 스트라이핑 되도록 이동시키는 방법으로, 이동하는 블록수가 많아 재구성 속도가 다소 느리지만 재구성 후의 I/O 성능 향상이 높은 방법이다. 또한 이 방법은 재구성 중 다른 I/O 요구를 수용하기 위해 해당 블록과 연관된 블록들을 처리해야 하는데, 연관된 블록의 수가 많다는 단점을 가지고 있다.

두 번째 방법은 기존 디바이스의 일부 데이터를 가능하면 연속된 데이터가 인접하지 않도록 이동시키는 방법으로, 재구성 후의 I/O 성능 향상은 앞의 방법에 비해 못하지만 재구성 속도가 빠르고 서로 연관된

블록의 수가 적어 재구성 중 다른 I/O 요구를 수용하기가 비교적 용이한 방법이다.

세 번째 방법은 새 디바이스에 이동할 데이터를 기존 디바이스에서 나누어 이동시키며, 앞 디바이스의 데이터부터 순차적으로 이동한다. 따라서 이동할 데이터가 적어 재구성 속도는 빠르고 서로 연관된 블록이 없기 때문에 재구성 중 I/O를 수용하기 가장 쉬운 방법이다. 하지만 스트라이핑을 전혀 고려하지 않았기 때문에 재구성 후 I/O 성능향상의 정도가 매우 떨어진다.

3.2. 자유공간 관리자의 설계

대용량의 저장 공간을 다수의 서버가 접근 가능한 SAN 환경을 고려한 자유공간 관리의 요구 사항은 아래와 같다.

- 대용량의 저장 공간에 적합해야 한다.
- 여러 호스트가 SAN의 저장 공간을 공유한다.
- 스냅샷과 재구성을 지원해야 한다.

자유공간 관리의 요구 사항에 기초하여 설계한 자유공간 관리 기법의 특징을 정리하면 다음과 같다. 먼저 자유공간 관리를 위해 비트맵 구조를 사용하며 자유공간은 최초 적합으로 할당한다. 또한 대용량의 저장 공간을 갖는 SAN을 고려하여 다수의 서버가 자유공간 정보를 분할 관리하도록 한다. 이때 다른 서버가 관리하는 영역에 대한 자유공간 할당 요청은 서버간의 통신을 이용하여 할당한다. 스냅샷과 재구성을 위한 공간 할당의 경우에는 추가의 디바이스를 사용하지 않고 같은 디바이스에 할당한다. 이때 정상 공간 요청에 대해서는 디바이스의 앞부분부터, 스냅샷이나 재구성을 위한 공간 할당 요청의 경우에는 디바이스의 뒷부분부터 할당함으로써 두 영역간에 간섭이 없도록 한다. 제안하는 기법에는 현재 디바이스에 있는 총 블록의 수, 정상 할당의 경우 다음 할당될 위치, 스냅샷이나 재구성을 위한 공간 할당의 경우 다음 할당될 위치에 대한 정보를 추가하여 비트맵 탐색시간을 줄인다.

4. 매핑 및 자유공간 관리자 구현

이 장에서는 3장에서 설계한 매핑 관리자 및 자유공간 관리자를 구현한 내용에 대해서 기술한다. 3장에서는 매핑테이블, 자유공간 관리, 스냅샷, 재구성 각각에 대해서 여러 가지 방법을 제안하였다. 매핑테이블에 대한 방법으로 네가지 방법을 제안했었지만, 논리주소를 기준으로 매핑테이블을 각 디스크에 분할에서 저장하고 분할된 매핑테이블을 전담 관리하는 호스트를 두는 형태를 구현하였다. 네 번째 매핑 엔트리를 범위로 표현하는 방법도 구현해 보았지만 파일 시스템에 따라 효율성이 다르므로 범용의 논리볼륨 관리자로서는 문제가 있다. 자유공간 관리자는 비트맵만을 이용하여 스냅샷과 일반적인 공간할당을 구분없이 처리하는 방법과 비트맵과 다음 할당할 블록의 위치를 가리키는 포인터를 같이 사용한 방법을 모두 구현하였다.

스냅샷에 대해서는 매핑테이블을 복사하는 방법과 해쉬 테이블을 유지하는 방법을 제시하였었는데 두가지 방법을 모두 구현하였다. 재구성방법에 대해서도 세가지 방법을 제시하였다. 이 논문에서는 세가지 방법 모두 구현하였으며 어떤 방법을 이용할 지는 재구성 유틸리티에서 옵션으로 줄 수 있도록 하였다.

이 논문에서 구현한 매핑 및 자유공간 관리자의 주요 함수들의 호출 관계를 그림으로 표현하면 그림 4-1과 같다. 구현한 볼륨 관리자를 구성하는 주요 함수들로는 논리볼륨 관리자의 유틸리티를 작성할 수 있게 해주는 pool_ioctl()과 상위 파일 시스템이나 데이터베이스 관리시스템과의 인터페이스를 위한 pool_make_request_fn()이 있다. 그리고, 실제로 매핑을 수행하는 map()과 스냅샷 볼륨일 경우 COW를 수행하기 위한 copy_on_write()가 존재한다. 또한, 매핑을 처리하기 위해서 물리블록을 할당하거나 스냅샷과 재구성을 처리하기 위해서 물리블록을 할당하기 위한 자유공간 관리자 함수들로 alloc_block(), alloc_snapshot_blocks(), release_snapshot_blocks(), release_block()이 있다. 마지막으로 사용자의 스냅샷 요구나 볼륨 재구성 요구를 처리하기 위한 create_snapshot(), remove_snapshot(), resize_volume() 이 그림 4-1과 같은 호출관계를 유지하고 있다.

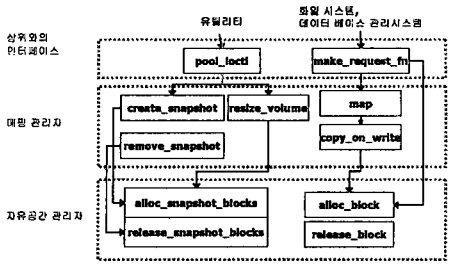

그림 4-1 구현한 자유공간 관리자 및 매핑 관리자의 함수 호출 관계

5. 결론

이 논문에서는 논리볼륨 관리자중에서 특히 매핑 관리자와 자유공간 관리자의 기능향상에 대한 방법을 찾고 이를 반영한 매핑 관리자와 자유공간 관리자를 설계하고 구현하였다. 이 논문에서 설계한 매핑 관리자는 매핑테이블을 이용하되 여러 가지의 매핑테이블의 표현 방법에 대해 제시하였다. 또한 매핑 관리자의 스냅샷 및 재구성에 대해서 다양한 방법을 제시하고 구현하여 비교해 보았다. 자유공간 관리자는 비트맵을 기반으로 하여 정상적인 공간할당과 스냅샷을 위한 공간할당을 모두 효과적으로 지원할 수 있도록 하였다.

6. 참고문헌

[1] Steven R. Soltis, Thomas M. Ruwart and Matthew T. O'Keefe, "The Global File System," In Proceedings of International Conference on Mass Storage Systems and Technologies, 1996.

[2] Edward K.Lee and Chandramohan A.Thekkath, "Petal : Distributed Virtual Disks," In Proceedings of International Conference on ASPLOS, 1996.

[3] P. R. Wilson et al, "Dynamic storage allocation: A survey and critical review," In Proceedings of International Workshop on Memory Management, 1995.

[4] Daniel Pierre Bovet and Marco Cesati, "Understanding the Linux Kernel," O'Reilly, 2000.

[5] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto and G. Peck, "Scalability in the XFS file system," In USENIX Technical Conference, 1996.

[6] Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami and Jim Williams, "HAMFS File System," In Proceedings of 18th IEEE Symposium on Reliable Distributed Systems, 1999.

[7] Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami and Jim Williams, "Alternatives of Implementing a Cluster File Systems," In Proceedings of the 17th IEEE Symposium on Mass Storage Systems 2000.