대용량 공유 스토리지 시스템을 위한 효율적인 스냅샷 기법

(An Efficient Snapshot Technique for Shared Storage Systems supporting Large Capacity)

김 영 호 † 강 동 재 †† 박 유 현 †† 김 창 수 †† 김 명 준 †††
(Young-Ho Kim) (Dong-Jae Kang) (Yu-Hyeon Bak) (Chang-Soo Kim) (Myung-Joon Kim)

요 약 본 논문에서는 대용량 스토리지를 공유하는 스토리지 클러스터 시스템에서 스냅샷 생성 이후 발생하는 쓰기 연산의 성능 저하를 해결하는 매핑 테이블 기반의 스냅샷 기법을 제안한다. 대용량 공유 스토리지 클러스터 시스템의 스냅샷 기법은 몇 가지 심각한 성능상의 문제점을 갖는다. 첫째 스냅샷 생성 시 스냅샷 매핑 테이블을 복사하는 기간동안 대상 저장 장치에 대해 모든 호스트의 접근 및 서비스가 중지된다. 둘째 스냅샷 시점의 데이타의 유지를 위해 수행되는 Copy-on-Write(COW) 이후에 발생하는 데이타 블록의 변경은 COW의 수행 여부의 판단을 위해 스냅샷 매핑 블록에 대한 추가적인 디스크 I/O의 요구로 쓰기 연산의 성능이 저하된다. 셋째 스냅샷 삭제 수행 시에도 COW가 수행 되었는지 판단하기 위한 매핑 블럭에 대한 추가적인 디스크 I/O가 요구되어 동시 수행되는 I/O 연산의 성능 저하를 가져온다.

제안한 스냅샷 기법에서는 최초 할당 비트(FAB:First Allocation Bit)와 스냅샷 상태 비트(SSB: Snapshot Stautus Bit)를 매핑 엔트리에 도입하여 기존 스냅샷 기법이 갖는 문제점들을 해결하였다. 스냅샷 생성 시 대상 저장 장치에 대한 I/O의 중단없이 데이타의 일관성을 보장한다. 또한 쓰기 연산 수행 시 COW의 수행 여부 판단을 원본 매핑 엔트리의 FAB와 SSB를 이용하여 스냅샷 매핑 블록에 대한 추가적인 I/O를 없앤다. 동일한 방법으로 삭제 시의 COW 수행 여부 판단을 처리하여 성능을 향상시킨다. 원본 매핑 엔트리의 SSB를 통해 할당을 해제하는 방식으로 성능을 향상시키는 스냅샷 수행 기법에 대해 설계하고 구현한다.

키워드 : 스냅샷, 매핑 테이블, SAN, 공유 스토리지

Abstract In this paper, we propose an enhanced snapshot technique that solves performance degradation when snapshot is initiated for the storage cluster system. However, traditional snapshot technique has some limits adapted to large amount storage shared by multi-hosts in the following aspects. As volume size grows, (1) it deteriorates crucially the performance of write operations due to additional disk access to verify COW is performed. (2) Also it increases excessively the blocking time of write operation performed during the snapshot creation time. (3)Finally, it deteriorates the performance of write operations due to additional disk I/O for mapping block caused by the verification of COW.

In this paper, we propose an efficient snapshot technique for large amount storage shared by multi-hosts in SAN Environments. We eliminate the blocking time of write operation caused by freezing while a snapshot creation is performing. Also to improve the performance of write operation when snapshot is taken, we introduce First Allocation Bit(FAB) and Snapshot Status Bit(SSB).It improves performance of write operation by reducing an additional disk access to volume disk for getting snapshot mapping block. We design and implement an efficient snapshot technique, while the snapshot deletion time, improve performance by deallocation of COW data block using SSB of original mapping entry without snapshot mapping entry obtained mapping block read from the shared disk.

Key words : Snapshot, Mapping Table, Storage Area Network, Shared Storage

† 정 회 원 : 한국전자통신연구원 디지털홈연구단 연구원
kyh05@etri.re.kr
†† 비 회 원 : 한국전자통신연구원 디지털홈연구단 연구원
djkang@etri.re.kr
bakyh@etri.re.kr
cskim7@etri.re.kr
††† 종신회원 : 한국전자통신연구원 디지털홈연구단 연구원
joonkim@etri.re.kr
논문접수 : 2003년 5월 19일
심사완료 : 2003년 12월 6일

1. 서 론

기업 규모의 시스템에서는 데이타 웨어하우스(Data Warehouse)의 구축과 ERP(Enterprise Resource Planning), CRM(Customer Relationship Management) 시스템 등의 도입으로 사용자에게 공유되고 서비스되는 데이타의 양이 기하급수적으로 증가하고 있다. 대용량 스토리지에 대한 요구가 계속 높아지고 있으며, 인터넷의 폭발적인 성장, 온라인 상에서의 정보유지 필요성, 의사결정 지원정보의 수집/추적에 대한 필요성, 서버 통합, 비즈니스 중심 애플리케이션으로의 PC 서버 움직임, 그리고 애플리케이션의 복잡성 증가 등 다양한 요인들도 이러한 요구를 부채질하고 있는 실정이다. 이처럼 높은 신뢰성과 성능, 내장애성(fault tolerance), 그리고 통합 관리와 고속 대용량 데이타 처리라는 요구에 대한 솔루션으로 등장한 것이 바로 SAN(Storage Area Network)이다. SAN은 분산 네트워킹에서 주류가 되고 있으며, 이미 스토리지를 구성하는 데 있어서 기본적인 조건으로 자리잡고 있다.

SAN은 새로운 컴퓨터 시스템 환경으로의 패러다임을 이동시키는 주된 개념으로 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 액세스할 수 있는 데이타 파일 중심의 컴퓨터 시스템 환경을 말한다. SAN의 기본적인 아이디어는 스토리지 전용망을 구축하는 것으로, 스토리지에 대한 고속 액세스와 일원적인 관리, 그리고 확장성과 내장애성을 높이는 것이 목적이다[1-3]. 사실상 SAN은 FC(Fibre Channel) 하나에 의해 구축된 망이 되기 때문에 이전의 SCSI에 의한 접속과 비교해서 고속이며, 긴 케이블로 인한 거리상 이점이 있고, 접속 대수도 증가시킬 수 있다. 일반적인 SAN 환경의 시스템 구성은 그림 1과 같이 각 호스트 및 저장 장치가 연결된 파이버 채널을 통해 스토리지 풀을 공유하는 형태를 갖는다.

SAN 환경에서 제공되는 고속의 대용량 저장 장치를 효율적으로 사용하기 위해 여러 개의 물리적 디스크 장치를 하나로 연결하고 이에 대해 소프트웨어 레이드를 지원하는 볼륨 관리자에 대한 많은 연구가 있었다. 대표적인 시스템으로 Linux LVM[4]이 있다. 이들은 단일 시스템에서 사용될 수 있도록 구성되어 있다. 하나의 시스템에서 방대한 양의 모든 데이타를 처리하는 것은 엔터프라이즈 환경에서의 다양한 요구사항을 만족시킬 수 없다. 여러 호스트가 함께 연결되어 있는 SAN 환경에서는 동시에 동일 저장 공간을 공유 가능하다. 공유되는 저장 공간은 정확한 데이타를 유지하기 위해 제어될 필요가 있으며 단일 시스템을 위한 볼륨 관리자는 적절한 제어를 수행할 수 없다.

일부 최근의 연구에서는 SAN환경에서 여러 호스트가 저장 장치를 공유할 수 있도록 적절한 제어를 수행하는 볼륨 관리자를 제안했다. 대표적인 시스템으로 GFS의 Pool Driver[5,6]가 있다. Pool Driver는 Linux에서 동작하며 SCSI 디바이스 드라이버와 파일 시스템 중간에 위치한 커널 내 모듈이다. 이것은 GFS 파일 시스템[7]과 함께 SAN 환경에서 저장 장치들을 공유하기 위해 사용될 수 있다.

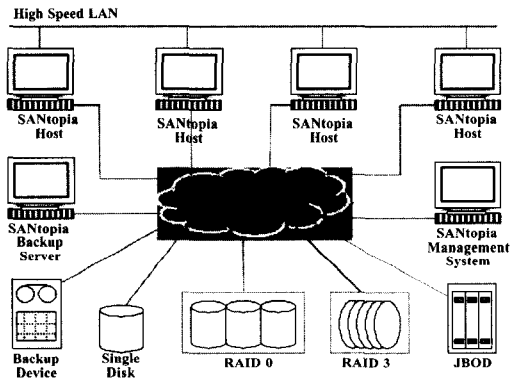

그림 1 SAN 환경의 구성

최근 엔터프라이즈 시스템에서는 고속 대용량의 데이타 처리 이외에 데이타에 대한 가용성 및 신뢰성이 크게 요구되고 있다. 가용성 및 신뢰성의 요구의 증가로 시스템의 중단없이 데이타의 백업을 수행할 수 있는 온라인 백업의 중요성이 더욱 부각되고 있다[8]. 백업 수행 시 시스템을 정지하고 백업이 완료된 후에 다시 서비스를 수행하는 시스템 형태의 운용은 불가능하므로 매핑 테이블 기반의 온라인 스냅샷의 제공이 필수적이다. 하지만 매핑 테이블 기반의 스냅샷 기법을 SAN을 기반으로 하는 대용량의 공유 스토리지 시스템에 적용하기에는 몇 가지 문제점을 가진다. 엔터프라이즈 시스템에서는 대용량의 저장 공간 이외에도 공유되는 호스트에서 제공되어야 하는 응용 프로그램의 계속적인 서비스가 요구된다. 그러나 스냅샷 기법에서는 스냅샷 생성이 수행될 때 스냅샷 매핑 테이블을 복사하는 기간동안 원본 볼륨에 대한 모든 공유 호스트의 데이타 변경 연산을 수행할 수 없다. 볼륨의 크기가 커질수록 매핑 테이블의 크기도 증가해서 스냅샷 생성에 의한 I/O 접근 제한 시간도 비례해서 증가한다. 또한 스냅샷 생성 후 데이타 블록에 대한 변경의 처리를 수행하는 Copy-

on-Write(COW)와 COW 이후에 발생하는 데이타 블록의 변경 수행 시 COW가 수행되었는지 판단하기 위한 스냅샷 매핑 블록에 대한 추가적인 I/O를 요구하게 되어 볼륨에 대한 I/O 성능의 저하를 초래한다. 삭제 수행 시에도 COW에 의해 할당된 데이타 블록의 할당을 위해 원본 볼륨의 데이타 블록과 스냅샷 볼륨의 데이타 블록이 변경되었는지를 검사해서 변경 할당된 데이타 블록을 자유공간 관리자에서 해제해야 하는 과정을 수행하기 때문에 스냅샷 삭제 수행 시간이 길어지게 된다.

이 논문에서는 스냅샷 상태 비트(SSB:Snapshot Stautus Bit)와 최초 할당 비트(FAB: First Allocation Bit)를 도입하여 기존의 스냅샷 기법을 대용량 공유 스토리지 클러스터 시스템에 적용시킬 때 발생하는 문제점들을 해결한다. 스냅샷 생성 시 I/O의 지연 및 서비스의 중단을 없애고, COW 이후에 발생하는 데이타 변경 연산의 성능을 향상시킨다. 또한, 스냅샷 삭제 시에도 COW의 수행 여부를 판단하기 위한 추가적인 디스크 접근으로 인한 성능 저하를 막는 스냅샷 기법에 대해 설계하고 구현한다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로서 매핑 테이블 기반의 스냅샷 기법 및 대용량 공유 저장 장치를 위한 스냅샷 기법에서 고려되어야 할 점들과 기존의 스냅샷 기법의 문제점에 대해 설명한다. 3장에서는 제안한 스냅샷 기법의 특징, 구조 및 생성/삭제 및 쓰기 연산의 알고리즘에 대해 설명한다. 4장에서는 제안한 스냅샷 기법을 SAN 환경의 대용량 공유 스토리지 클러스터 시스템 관리자인 SVM[9,10]에 적용하여 구현하고 기존의 스냅샷 기법과 성능을 비교 평가한 결과에 대해 기술한다. 5장에서는 결론 및 향후 추가 연구가 필요한 부분에 대해 기술한다.

2. 관련연구

본 장에서는 매핑 테이블 기반의 스냅샷 기법에 대해 살펴본다. 매핑 테이블 기반의 스냅샷 기법[8,11,12]의 개념과 데이타의 일관성 유지를 위해 수행되는 COW의 처리 과정에 대해 설명한다. 또한 기존의 스냅샷 기법을 대용량 공유 저장 장치 환경에 적용할 때 발생하는 문제점에 대해 살펴본다. 대용량 공유 저장 장치에서 사용되는 스냅샷 기법에서 성능 저하를 초래하는 요인과 발생 원인에 대해 자세히 설명한다.

2.1 매핑 테이블 기반의 스냅샷 기법

스냅샷은 사용자가 원하는 특정 시점에서의 데이타 상태를 저장, 유지 시켜주는 기법이다. 스냅샷 기법은 데이타 전체가 아닌 데이타에 대한 이미지만을 복사해서 스냅샷 생성 시점의 데이타를 그대로 유지하게 된다. 스냅샷은 논리 볼륨 관리자의 가상 디스크 구조에서 데이타를 중복시키는 강력한 툴이다. 또한 스냅샷은 파일 시스템이나 디스크 파티션이 사용 중인 온라인 상태에서 원하는 시점의 데이타를 백업하는 온라인 백업을 가능하게 지원한다. 따라서 원하는 시점으로의 데이타에 대한 회귀가 가능하다. 스냅샷 기법은 스냅샷 생성 시 해당 시점의 원본(original) 볼륨의 매핑 테이블을 스냅샷 볼륨을 위한 매핑 테이블로 복사한다. 복사된 매핑 테이블은 스냅샷 볼륨이 삭제될 때까지 유지된다. 볼륨을 접근 사용하는 클라이언트의 사용 요구에 의해 볼륨 데이타의 변경이 발생하면 원본 볼륨의 데이타를 새로운 공간을 할당받아 복사해 놓고 새로운 데이타의 변경을 수행한다. 스냅샷 볼륨에 대해서는 읽기 연산의 수행만 가능하고 스냅샷 볼륨을 위한 매핑 테이블을 이용해서 처리를 한다.

매핑 테이블 기반의 스냅샷 생성은 다음과 같이 수행된다. 사용자로부터 스냅샷 생성 요청을 받은 호스트는 먼저 스냅샷을 생성하려는 원본 볼륨을 접근하는 모든 호스트에서 수행되는 I/O의 수행을 막고 스냅샷 생성을 수행한다. 이러한 I/O 차단은 스냅샷이 생성되는 시점의 데이타를 일관성 있게 유지하기 위해 필요하다. 호스트별로 관리하는 매핑 영역에 대해 스냅샷을 생성하라는 메시지를 전달한다. 메시지를 수신한 호스트들은 각 각 자유공간 관리자를 호출하여 호스트 자신이 관리하는 매핑 테이블의 크기만큼 공간을 할당받아 매핑 테이블을 복사하는 과정을 수행해서 스냅샷 생성을 마친다.

스냅샷 볼륨 생성 이전에 존재하던 데이타 블록에 대한 데이타의 변경이 발생하면 Copy-on-Write(COW) 기법을 이용해서 스냅샷 볼륨의 데이타를 생성 시점의 상태로 유지한다. 스냅샷 생성 이후에 할당된 데이타 블록에 대한 변경은 스냅샷 매핑 테이블에 반영되어 있지 않으므로 COW를 수행할 필요없이 일반 입/출력 연산처럼 처리를 수행하면 된다.

한번 생성된 스냅샷은 시스템의 재부팅과는 상관없이 사용자가 스냅샷을 삭제할 때까지 유지된다. 사용자에 의해 스냅샷 삭제가 요청되면 이를 요청받은 서버는 매핑을 담당하는 모든 호스트에게 스냅샷 삭제 요청을 전달한다. 스냅샷 삭제는 스냅샷 생성 후 발생한 변경을 처리하는 COW에 의해 새로 할당된 데이타 블록을 해제하는 과정을 수행해야 한다. 각 서버의 매핑 관리자는 스냅샷 매핑 테이블과 원본 매핑 테이블을 논리주소의 순서로 비교를 하면서 물리주소가 서로 다른 엔트리를 찾는다. 서로 다른 물리 주소는 COW에 의해서 할당된 블록이므로 자유공간 관리자를 호출하여 물리 블록에 대한 할당을 해제한다. COW에 의해 할당된 블록들을 모두 해제하고 난후 스냅샷 볼륨에 대한 매핑 테이블을 삭제하는 작업을 수행한다. 스냅샷 볼륨을 위한 매핑 테

이블의 삭제가 완료되면 스냅샷 볼륨에 대한 정보를 삭제하면 스냅샷의 삭제 과정이 완료된다.

2.1.1 Copy-on-Write 메카니즘

스냅샷 생성 후 데이타 블록에 대한 변경이 발생하면 스냅샷 시점의 데이타를 유지할 수 있도록 처리하는 과정을 수행해야 한다. 이처럼 데이타의 일관성 유지를 위해 수행하는 작업을 Copy-on-Write(COW)라 한다. COW는 동일 데이타 블록에 대해 오직 한 번만 수행된다. 새로운 블록을 할당 해 스냅샷 시점의 데이타를 복사하고 스냅샷의 매핑 엔트리가 새로 할당된 데이타 블록을 매핑 하도록 매핑 엔트리 값을 변경한다.

COW의 수행 과정을 설명하면 다음과 같다. COW를 수행하기 위해서는 먼저 원본 매핑 테이블에서 변경이 발생한 데이타 블록에 대한 매핑 엔트리를 찾아서 해당 엔트리의 논리 블록과 매핑되는 디스크 블록의 물리 주소를 얻어낸다. 그리고 생성된 모든 스냅샷의 매핑 테이블로부터 동일 논리 블록에 대한 물리 블록의 주소를 얻어낸 후 이를 원본 매핑 테이블 엔트리의 물리 주소와 비교한다. 두 매핑 엔트리의 물리 주소가 같으면 스냅샷 생성 후 해당 데이타 블록에 대한 변경이 처음 발생했다는 의미이다. 따라서 해당 데이타 블록에 대해 COW를 수행한다. 물리 주소가 서로 다르면 이미 COW가 수행된 경우이므로 데이타 블록에 대한 변경을 디스크에 반영하면 된다. COW를 수행해야 되는 데이타 블록에 대해서는 먼저 새로운 물리 블록을 할당받고 원본 데이타 블록의 데이타를 새로 할당받은 물리 블록에 복사한다. 그 다음 원본 데이타 블록에 대한 변경을 수행한다. 마지막으로 스냅샷 매핑 엔트리의 매핑 주소를 새로 할당된 데이타 블록의 물리 주소로 변경하면 COW의 처리가 완료된다. 이러한 COW의 과정은 매핑 테이블을 기반으로 하는 스냅샷 기법에서는 모두 동일하게 수행해야 하는 과정이다.

그림 2는 매핑 테이블 기반의 스냅샷 기법에서 copy-on-write의 수행 과정의 예를 나타낸다. 원본 볼륨은 가장 최근의 데이타를 가지고 있으며 스냅샷 볼륨은 스냅샷이 생성되었을 때 변경된 데이타에 대해서 스냅샷이 걸린 상태를 유지하기 위해 필요한 데이타를 저장한다. 스냅샷이 생성되기 전에 파일A에 대한 읽기 요청이 들어오면 파일 시스템에 저장되어 있는 파일 데이타 블록을 읽어서 서비스를 수행한다. 스냅샷이 생성된 이후의 쓰기 연산은 첫 번째 갱신인지를 검사하는 과정이 필요하다. 만일 파일 B의 첫 번째 블록에 대한 쓰기 연산이 들어온 경우에 첫 갱신이라면 파일 시스템에 있는 파일 B의 첫 번째 블록의 내용을 스냅샷 영역에 복사하

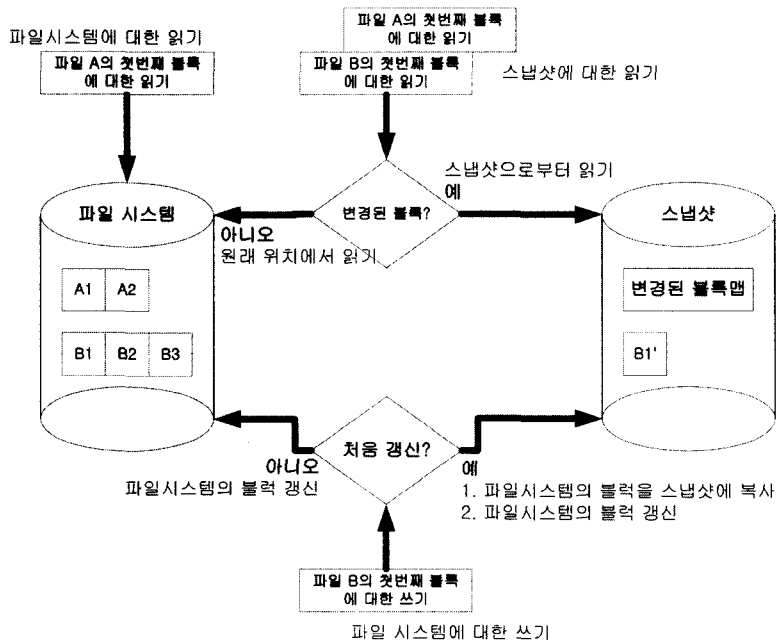

그림 2 Copy-on-Write의 수행 과정

고 이의 정보를 변경된 스냅샷 블록맵에 기록한다. 실제 갱신할 내용은 파일 시스템 내의 파일 B의 첫 번째 블록에 기록한다. 스냅샷에 대한 읽기 연산에 대해서도 변경된 블록인지를 검사하는 과정이 필요하다. 만일 변경되지 않은 블록이라면 파일 시스템의 데이타를 바로 읽는다. 하지만 변경되었다고 판단이 되면 스냅샷 영역의 변경된 블록맵을 검사하여 변경된 블록의 물리적인 위치를 찾아 데이타를 읽는다.

2.2 대용량 공유 스토리지 클러스터 환경에서 기존 스냅샷 기법의 문제점

스냅샷 기법은 데이타베이스에서 주로 사용되던 개념이다. 많은 양의 데이타를 관리 저장하는 환경에서 특정 시점의 데이타들을 이용하고자 그 시점의 데이타들을 그대로 백업해야 했다. 그러나, 데이타 전체에 대한 백업은 시간이 많이 소요되는 작업이고, 백업 작업을 수행하면서 특정 시점의 데이타를 그대로 유지하기는 어려웠다. 그래서 도입된 개념이 스냅샷이다. 스냅샷은 데이타 전체를 저장하지 않고 데이타를 얻을 수 있는 이미지만 복사하고 변경이 발생하면 Copy-on-Write라는 기법을 사용해서 스냅샷 시점의 데이타를 유지하는 기법이다. 이러한 개념이 일반 대용량의 데이타를 처리하는 엔터프라이즈 환경에서 요구되고 있다. 응용 분야가 시스템의 가용성뿐만 아니라 무정지 시스템의 환경을 요구하기 때문이다. 일반 백업에서는 특정 시점의 데이타를 백업받기 위해서는 반드시 시스템의 정지 및 오랜 시간의 백업 처리 작업 과정을 가져야 한다. 이러한 문제를 해결하기 위해 스냅샷을 이용한 온라인 백업 시스템이 도입되어 사용되고 있다. 그러나, 스냅샷 기법이 적용되는 시스템 환경이 대용량의 공유 스토리지 클러스터 환경으로 변화되면서 몇 가지 성능상의 문제를 나타내는데 아래에서 각각의 문제점에 대해 상세히 기술한다.

2.2.1 스냅샷 생성 시 볼륨의 데이타 블록에 대한 변경 연산의 지연

스냅샷은 특정 시점의 데이타 이미지를 그대로 복사 유지하여 사용자가 스냅샷 생성 시점 당시의 데이타를 그대로 이용할 수 있어야한다. 따라서, 기존의 스냅샷 기법에서는 특정 시점의 데이타 이미지를 유지하기 위해 데이타의 이미지를 가지고 있는 매핑 테이블의 복사가 완료되기 전까지 해당 데이타 저장 공간에 대한 접근 및 변경 연산의 수행을 차단하였다. 매핑 테이블 복사가 가지는 문제점은 스냅샷 동작이 완료될 때까지 스냅샷을 복사 중인 원본 볼륨에 대하여 액세스 할 수 없다는 것이다. 이것은 스냅샷 동작이 수행되는 순간과 동시에 볼륨을 액세스하거나 갱신할 수 없다는 것을 의미한다. 데이타 저장 공간이 적고 저장 공간을 여러 호스트 노드에서 공유하지 않는 시스템 환경에서는 스냅샷 생성이 큰 성능상의 저하를 초래하지는 않았다. 수초 이내의 시간이면 매핑 테이블의 복사가 완료되고 스냅샷의 생성이 이루어지기 때문이다. 그러나, SAN 기반의 대용량 스토리지 클러스터 시스템은 수~수십 TB 이상의 저장 공간을 지원한다. 수십~수백 GB의 저장 공간에 대한 스냅샷을 생성하는데에도 수 십초 이상의 소요된다. 따라서, 수 TB 이상에서는 훨씬 오랜 기간의 시간이 소요되고, 그 기간동안 해당 저장 공간에 대한 접근 및 I/O 수행이 지연된다. 공유 스토리지 클러스터 시스템에서는 여러 호스트들이 동시에 동일 저장 공간에 접근하고 서비스를 받는다. 따라서 스냅샷 생성으로 인한 오랜 시간의 서비스 중단은 허용되기 어렵다. 소프트웨어와 하드웨어의 발달로 요구되는 시스템 환경의 변화에 의해 스냅샷 생성을 수행하는 작업의 성능 저하를 가중시키고 있고, 현실적으로 수행하기 어려운 상황으로 발전하고 있다.

2.2.2 Copy-on-Write(COW)이후 발생하는 쓰기 연산의 성능 저하

스냅샷 생성 후 데이타 블록의 변경이 요청되면 스냅샷 생성 시점의 데이타를 유지하기 위해 변경이 발생한 데이타 블록에 대해 COW 처리를 수행해야 한다. 스냅샷 생성 이전에 사용되지 않은 영역에 대해 쓰기 연산이 발생하면 COW를 수행하지 않고 일반 쓰기 연산처럼 처리하면 된다. 스냅샷 생성 이전에 이미 할당 사용되는 데이타 블록에 대한 변경 연산이 발생하면 처음 변경인 경우에는 스냅샷 데이타를 보존하기 위한 COW를 수행하고, 이미 COW가 수행된 경우에는 역시 일반 쓰기 연산처럼 처리한다. COW는 스냅샷 생성 후 동일 블록에 대해 오직 한 번 수행되는 작업이다. 데이타 변경 연산에 대한 COW의 수행 여부를 판단하는 과정은 스냅샷이 삭제될 때 까지 계속 요구된다. 기존의 스냅샷 기법에서는 스냅샷 엔트리와 원본 엔트리를 모두 디스크로부터 읽어서 두 엔트리가 매핑하는 데이타 블록이 동일한지 비교하여 판단한다. 즉, 스냅샷 생성 이후에 발생하는 모든 데이타 변경 연산에서는 스냅샷 매핑 엔트리를 읽기 위한 추가적인 디스크 I/O를 계속 수행해야 한다. 저장 공간의 대용량화로 백업 수행 시간이 길어지므로 스냅샷이 유지되는 기간도 더 늘어나게 되고, 스냅샷이 유지되는 동안 COW 수행 여부의 판단으로 인해 데이타 변경 연산에 대한 성능 저하가 계속된다. 스냅샷이 추가로 생성되어 하나 이상의 스냅샷이 존재하면 COW를 판단하기 위해 스냅샷 개수 만큼의 매핑 엔트리에 대한 읽기 연산을 추가로 수행하게 된다. 즉, 스냅샷 개수 만큼의 디스크 I/O가 계속 수행되어 성능 저하는 더욱 커지게 된다.

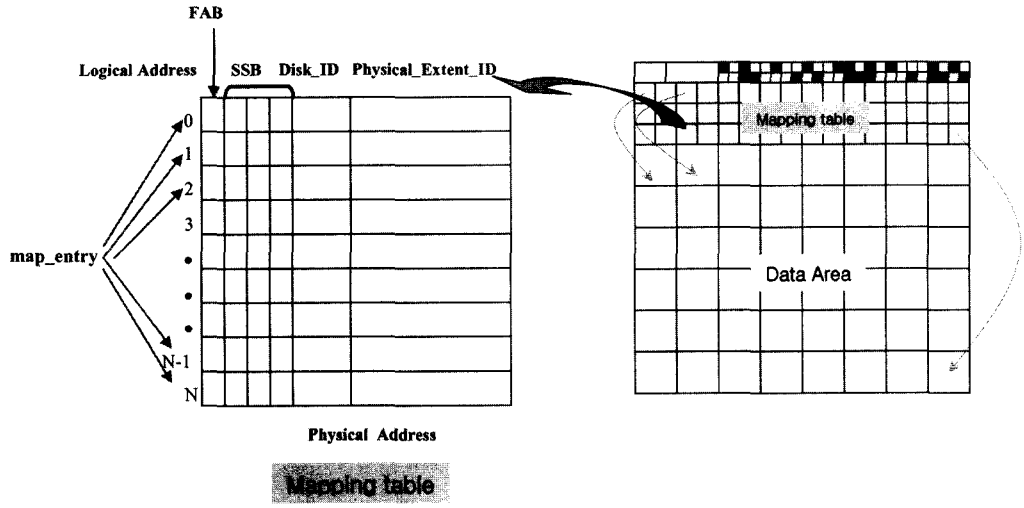

그림 3 제안한 스냅샷 기법의 매핑 테이블 및 매핑 엔트리 구조

2.2.3 스냅샷 삭제 시 COW 할당 해제로 인한 성능 저하

스냅샷 삭제가 요청되면 COW에 의해 새로 할당된 데이타 블록에 대한 할당을 해제하는 작업 수행 과정을 거친다. 기존의 스냅샷 기법에서는 COW의 수행 여부를 판단하기 위해 쓰기 연산의 수행과 마찬가지로 원본 매핑 엔트리와 스냅샷 매핑 엔트리를 동시에 읽어서 비교해야 한다. 원본 매핑 엔트리만으로는 스냅샷 이후에 새로 할당된 데이타 블록인지, 스냅샷 이전 할당되고 변경이 발생하지 않은 블록인지, 스냅샷 이전 할당되고 스냅샷 생성 이후 변경이 발생한 블록인지 구별할 수 없다. 스냅샷과 원본 매핑 엔트리를 비교하여 두 엔트리가 매핑하는 데이타 블록의 주소가 서로 다르면 COW가 발생한 경우로 데이타 블록의 할당을 해제한다. COW의 수행을 판단하기 위해서는 스냅샷 매핑 엔트리에 대한 추가적인 디스크 I/O가 요구된다. 만약, 스냅샷의 개수가 하나 이상이면 스냅샷의 개수만큼 추가적인 디스크 I/O를 수행해야 하기 때문에 성능 저하는 더욱 커지게 된다. 저장 공간의 크기가 커지면 비례하여 데이타 블록이 증가되고 스냅샷 삭제 수행 시간은 더욱 길어지게 된다.

3. 제안한 스냅샷 알고리즘의 특징 및 수행 과정

기존의 스냅샷 기법은 관련 연구에서 살펴본 것처럼 대용량의 공유 스토리지 클러스터 환경에 적용될 때 I/O 성능을 저하시키는 등의 몇 가지 문제점을 가지고 있다. 스냅샷 생성 및 삭제 시의 소요 시간 및 원본 볼륨에 대한 I/O 및 서비스의 지연, 스냅샷 생성 후 발생하는 데이타 블록에 대한 쓰기 연산 시 COW의 수행으로 인한 쓰기 성능의 저하 등의 문제이다. 이런 문제를 해결하기 위해 제안하는 스냅샷 기법에서는 스냅샷 상태 비트(SSB: Snapshot status Bit)와 최초 할당 비트(FAB: First Allocation Bit)라는 개념을 도입하였다. 이 장에서는 SSB와 FAB를 기반으로 하는 제안한 스냅샷 기법의 특징과 기존 스냅샷 기법이 갖는 문제점들을 해결하기 위한 스냅샷 연산 알고리즘에 대해 설명한다. 제안한 스냅샷 기법의 특징 및 수행 과정에 대해 설명하고, 알고리즘을 적용한 스냅샷 생성 및 삭제, 데이타 변경 연산의 수행 과정에 대해 자세히 설명한다.

3.1 FAB(First Allocation Bit)와 SSB(Snapshot Status Bit)의 도입

제안한 스냅샷 기법의 가장 큰 특징은 스냅샷 상태 비트(SSB: Snapshot Status Bit)와 최초 할당 비트(FAB: First Allocation Bit)라는 개념을 도입했다는 것이다. COW의 수행 여부의 판단을 위해 요구되는 스냅샷 매핑 블록에 대한 추가적인 I/O를 줄이기 위해 개념을 도입했다. 온라인 백업을 지원하는 대용량 공유 스토리지 시스템에서는 매핑 테이블 기반의 매핑을 제공한다. 그림 3은 제안한 스냅샷 기법의 매핑 테이블 및 매핑 엔트리 구조를 나타낸다. 매핑 테이블 기반의 물리 디스크의 레이 아웃은 크게 2 부분으로 이루어진다. 매핑 엔트리 블록이 저장되는 매핑 테이블 영역과 실제 데이타가 저장되는 데이타 영역이다. 매핑 테이블은 물리적인 실제 데이타 영역과 사용자 영역의 논리적인 블록을 매핑 시켜주는 매핑 엔트리들의 집합이다. 기존 스

냅샷 기법의 매핑 엔트리는 물리 블록의 주소 정보만 유지하였다. 따라서 COW의 수행 여부를 판단하기 위해 항상 원본 매핑 블록 이외에 스냅샷 매핑 블록에 대한 접근이 요구되었다. 제안한 기법에서는 스냅샷 매핑 블록의 추가적인 디스크 접근으로 인해 발생하는 쓰기 연산 및 스냅샷 삭제 연산의 성능 저하를 줄이기 위해 매핑 정보 이외에 데이타 블록에 대한 상태를 나타내는 비트 정보를 유지하여 성능 저하 문제를 해결하였다. 초기 할당 비트(FAB: First Allocation Bit)와 스냅샷 상태 비트(SSB: Snapshot Status Bit)가 매핑 엔트리에 추가로 유지되는 상태 정보들이다. FAB는 엔트리마다 1비트 씩 할당되고, SSB는 엔트리마다 생성되는 스냅샷의 개수만큼 할당된다. SSB와 FAB는 공유 스토리지의 매핑 테이블 초기화 단계에서 0으로 초기화된다.

SSB는 매핑 엔트리가 매핑하는 데이타 블록의 스냅샷 생성 이후의 상태를 나타내는 비트로 1이면 스냅샷 생성 후 데이타 블록에 대한 변경 연산의 수행으로 COW가 수행되었다는 의미이고, 0이면 아직 스냅샷 생성 후 데이타 블록에 대한 변경이 발생하지 않은 상태임을 표시한다. 즉, COW가 아직 수행되지 않은 상태이다. 스냅샷 생성 후 데이타 블록의 변경으로 COW가 수행되면 해당 매핑 엔트리의 SSB값을 1로 변경한다. SSB값은 스냅샷 삭제 시 COW가 수행된 데이타 블록에 대한 할당을 해제할 때 다시 0으로 초기화된다.

FAB는 데이타 블록 중 스냅샷 생성 이전에는 사용되지 않다가 스냅샷 생성 이후 처음 사용되는 데이타 블록들을 구별하기 위한 비트이다. 데이타 블록에 대한 변경 연산이 발생하면, 처음 변경인 경우 새로운 데이타 블록을 할당하고 디스크에 쓰기를 수행한다. 스냅샷 이전에 사용되지 않은 블록이므로 COW는 수행되지 않는다. COW가 수행되지 않기 때문에 SSB의 값은 초기값을 그대로 유지한다. 두 번째 변경부터는 데이타 블록이 사용 중인 상태이고 SSB값이 0이기 때문에 COW를 수행해야 한다. 그러나 이러한 경우에는 COW를 수행하면 안 된다. 기존의 기법에서는 스냅샷이 존재하는 경우 데이타 블록의 변경이 발생하면 항상 원본 매핑 엔트리와 스냅샷 매핑 엔트리를 비교하여 처리하기 때문에 이러한 경우 문제가 되지 않는다. 그러나, 제안한 기법에서는 스냅샷 매핑 엔트리를 읽지 않고 원본 매핑 엔트리만 접근하므로 SSB 만으로는 스냅샷 생성 후 처음 사용된 블록과 스냅샷 생성 이전에 할당되고 COW가 수행되지 않은 블록의 구별이 불가능하다. 스냅샷 생성 후 처음 사용되는 데이타 블록들에 대한 구별과 COW의 수행을 방지하기 위해 매핑 엔트리의 첫 번째 비트를 FAB로 할당하여 사용하고 있다. 따라서 언제나 스냅샷 매핑 엔트리에 대한 추가적인 I/O를 수행하지 않아도 처리가 가능하다. SSB와 FAB의 실제 사용에 대해서는 다음의 스냅샷 연산 알고리즘에서 자세히 언급한다.

3.2 I/O 중단없는 스냅샷 생성 알고리즘

기존의 스냅샷 기법에서는 스냅샷 생성이 요청되면 먼저 스냅샷 생성의 대상이 되는 원본 볼륨에 대한 I/O를 차단한다. 스냅샷 생성 시점의 볼륨 데이타에 대한 일관성을 유지하기 위해서는 매핑 테이블의 복사가 완료될 때까지 원본 볼륨에 대한 변경이 수행되어서는 안 된다. COW 수행을 통해 데이타 블록의 변경으로 인한 스냅샷 시점의 데이타 유지가 보장되어야 한다. 매핑 테이블 기반의 스냅샷 기법에서 COW를 수행하기 위해서는 스냅샷 볼륨을 위한 매핑 테이블이 생성되어야한다. 볼륨의 크기가 커지면 저장되는 데이타 블록의 수가 증가되고 매핑 엔트리의 수도 데이타 블록에 비례하여 증가하므로 복사되는 매핑 테이블의 크기는 대상 볼륨의 크기에 비례하여 커진다. 매핑 테이블의 크기가 커지면 매핑 테이블을 복사하는 시간이 증가하고 비례하여 I/O 지연 시간도 증가하게 된다. 엔터프라이즈 시스템 환경을 위한 대용량 논리 볼륨 관리자에서 제공하는 볼륨의 크기는 일반적으로 수백GB부터 수TB까지 이른다. 이러한 경우 매핑 테이블의 복사 시간만 수분에서 이상 소요된다. 하지만 기존의 스냅샷 기법에서는 데이타의 일관성 유지를 위해 매핑 테이블을 복사하는 동안 대상 볼륨에 대한 I/O 요청을 완전히 차단한다.

제안한 알고리즘에서는 스냅샷 생성 시 볼륨에 대한 I/O의 지연을 최소화하기 위해 연산 모드를 구분하였다. 연산 모드를 NORMAL과 SNAPSHOT_CREATE의 두 가지 모드로 나눈 것이다. 연산 모드는 스냅샷이 생성되는 매핑 테이블의 복사 중에 데이타 변경 연산이 요청되는 경우 COW를 수행하여 스냅샷 생성 시점의 데이타를 유지하도록 보장한다. 제안한 스냅샷의 생성 기법은 매핑 테이블을 복사할 때 원본 볼륨에 대한 I/O를 동시에 처리한다. 단지, 매핑 서버 호스트의 연산 모드를 스냅샷 생성(SNAPSHOT_CREATE)모드로 변경하는 동안만 I/O의 지연이 발생하게 된다. 연산 모드 변경으로 인한 지연은 매핑 테이블을 복사하는 시간과 비교하면 무시할 수 있을 정도의 아주 짧은 시간만 소요된다.

그림 4는 제안한 스냅샷 생성 알고리즘의 의사코드를 나타낸다. 제안한 스냅샷 생성의 알고리즘을 살펴보면 다음과 같다. (1) 스냅샷 생성이 요청되면 매핑 서버 호스트의 볼륨의 연산 모드를 정상(NORMAL) 모드에서 스냅샷 생성(SNAP_CREATE) 모드로 변경시킨다. (2)(3)(4)원본 매핑 테이블의 모든 블록에 대해 첫 번째 매핑 블록부터 모든 블록에 차례대로 배타 모드의 잠금을 획득하고 매핑 블록의 복사 작업을 수행한다. 잠금을 획

```
Snapshot Creation operation
{
 Input : original_name, snapshot_name

 for(모든 매핑 서버 호스트에 대해)
     호스트 논리 볼륨의 연산 모드를 SNAPSHOT_CREATE로 변경;          (1)

 for( 모든 매핑 블록에 대해 )
 {
     map_blk_no = 매핑 정보가 저장된 매핑 블록을 구함;              (2)
     map_lock = map_blk_no에 대한 배타 모드의 잠금 획득;            (3)
     map_blk = 매핑 블록을 물리 디스크에서 읽어옴;                  (4)

     if( map_lock != NULL ) // 매핑 블록에 대한 잠금을 획득한 경우  (5)
     {
         map_blk의 내용을 snap_map_blk로 복사;                      (6)
         snap_map_blk를 디스크에 기록;                              (7)
         map_lock의 잠금 해제;                                      (8)
     }
 }
 for(모든 매핑 서버 호스트에 대해) // 볼륨의 연산 모드가 snapshot인 경우
     호스트 논리 볼륨의 연산 모드를 NORMAL로 변경;                  (9)
}
```

그림 4 스냅샷 생성 알고리즘의 의사코드

득하지 못하면 모든 매핑 블록의 복사가 완료되었는지 검사한다. 잠금을 획득하지 못하는 경우는 동일 블록에 대해 변경을 수행하는 다른 프로세스가 이미 잠금을 획득하고 수행중인 경우이다. 이 때에는 잠금을 획득하고 수행중인 쓰기 연산에서 연산 모드를 검사해서 COW 작업과 매핑 블록의 복사를 모두 수행한다. (5)잠금을 획득한 경우에는 FAB나 SSB값을 검사한다. 데이타 변경의 동시 수행으로 이미 COW와 매핑 블록의 복사가 완료된 블록이 존재할 수 있기 때문이다. (6)FAB와 SSB의 값이 모두 0인 경우에만 매핑 블록의 복사를 수행하고, 두 값중 하나라로 1인 경우에는 이미 복사가 완료된 경우로 복사를 하지 않는다. (7) (8)복사가 완료되면 매핑 블록에 대한 잠금을 해제한다. 하나의 매핑 블록에 대한 복사가 완료되면 원본 볼륨의 모든 매핑 블록에 대해 복사가 완료되었는지 검사하고 완료되지 않은 경우에는 다음 블록에 대한 잠금 획득 후 복사 작업을 계속 수행하고, (9)완료된 경우에는 매핑 서버 호스트의 연산 모드를 정상 모드로 변경시킨다. 이러한 방법으로 스냅샷 생성을 수행하면 원본 볼륨에 대한 I/O 중단 없이 스냅샷 생성을 수행할 수 있다.

3.3 스냅샷 존재 시 성능 저하를 최소화하는 쓰기 연산 알고리즘

스냅샷이 존재하는 경우의 읽기 연산은 스냅샷이 존재하지 않을 때의 읽기 연산과 동일하게 처리된다. 매핑을 통해 논리 블록과 일치하는 물리 블록을 구하고 디스크의 물리 블록에서 데이타를 읽어오는 과정으로 수행된다. 하지만 쓰기 연산은 관련 연구에서 언급된 것처럼 COW의 수행 여부를 판단하는 추가적인 처리가 요구되고, 이러한 처리는 디스크에 대한 추가적인 I/O가 요구되어 쓰기 연산의 성능 저하를 초래한다. 제안한 스냅샷 기법에서는 스냅샷이 존재할 때 발생하는 데이타 변경 연산을 두 가지로 분류하여 처리한다. 스냅샷이 생성 중에 수행되는 변경 연산과 스냅샷 생성 후의 변경 연산으로 나뉘어진다. 스냅샷 생성 중의 변경 연산은 COW와 매핑 블록 복사를 수행한다. 스냅샷 생성 후의 변경은 COW만 수행하면 된다. 쓰기 연산의 수행 전에 이미 COW가 수행 되었는지 판단해야 하는데, 기존의 스냅샷 기법에서는 원본 매핑 엔트리 및 스냅샷 매핑 엔트리를 모두 읽어서 두 엔트리가 매핑하는 물리 블록의 주소를 비교하여 COW의 수행 여부를 판단하였다. 즉, 일반적인 쓰기 연산 보다 매핑 블록에 대한 I/O가 최소 한번 더 요구된다. 스냅샷의 수가 증가하면 스냅샷의 수에 비례하여 I/O 횟수도 증가한다. 스냅샷 수에 비례하여 쓰기 연산의 성능이 저하된다.

본 논문에서는 SSB와 FAB를 이용하여 기존의 스냅샷 기법이 갖고 있는 쓰기 연산의 성능 저하 문제를 해결하였다. 그림 5는 제안한 스냅샷 기법에서의 쓰기 연산의 알고리즘을 나타낸다. 볼륨에 대한 I/O 요청이 발생하면 먼저 (1)매핑 엔트리가 저장된 매핑 블록의 물리적인 디스크 및 블록의 주소를 구한다. (2)매핑 블록에 대한 배타 모드의 잠금을 획득하고 (3)매핑 블록을 디스크에서 읽어오고 논리 주소에 해당되는 매핑 엔트리를 얻는다. (4)연산 모드가 스냅샷 생성(SNAPSHOT_CREATE)모드인지 검사한다. 연산 모드가 정상 모드(NORMAL)이면 스냅샷이 존재하는지 검사한다. 스냅샷이 존재하지 않는 경우에는 일반 쓰기 연산처럼 데이타 블록을 디스크에 기록하고 매핑 블록의 잠금을 해제하고 종료한다. (5)볼륨에 대한 스냅샷이 존재하는 경우에

```
WRITE operation
{
 Input : logical blkno
 * old_blk  : 변경이 발생한 원본 데이터 블록
 * new_blk : COW를 위해 새로 할당된 데이터 블록

  map_blk_no = blk_no의 매핑 정보가 저장된 매핑 블록을 구함;                    (1)
  map_lock = map_blk_no에 대한 배타 모드의 잠금 획득;                            (2)
  map_blk = 매핑 블록을 물리 디스크에서 읽어옴;                                  (3)

  if(snap_count != NULL) // 스냅샷이 존재하는 경우                               (4)
  {
      if( first_ssb == 1 )   // 첫 번째 쓰기 연산이 스냅샷 생성 이후 수행된 경우   (5)
          데이터 블록 old_blk를 디스크에 기록;                                    (6)

      else if( current_ssb == 0 )                                               (7)
      {  // COW가 수행되지 않은 스냅샷 생성 후 첫 변경 연산인 경우
         snap_map_blk = 스냅샷 매핑 블록을 디스크에서 읽어옴;                     (8)
         new_blk_no = 스냅샷 데이터 블록이 복사 될 물리 블록을 할당;              (9)
         데이터 블록 old_blk의 내용을 new_blk로 복사;                            (10)
         스냅샷 매핑 엔트리의 물리 블록의 주소를 new_blk_no로 변경;              (11)
         원본 볼륨 매핑 엔트리의 스냅샷에 대한 ssb(current_ssb)를 1로 변경;     (12)
         데이터 블록 old_blk를 디스크에 기록;                                    (13)
         snap_map_blk과 map_blk를 디스크에 기록;                                 (14)
      }
      else  // COW가 이미 수행 된 경우                                          (15)
      {
         데이터 블록 old_blk를 디스크에 기록;                                    (16)
      }
  }
  else if(op_mode == SNAPSHOT_CREATE) // 매핑 테이블 복사가 완료되지 않은 경우  (17)
  {
      if( current_ssb == 1 ) // COW가 이미 수행된 경우                           (18)
          데이터 블록 old_blk를 디스크에 기록;                                    (19)
      else                                                                      (20)
      {
          new_blk_no = 스냅샷 데이터 블록이 복사 될 물리 블록을 할당;           (21)
          데이터 블록 old_blk의 내용을 new_blk로 복사;                          (22)
          스냅샷 매핑 엔트리의 물리 블록의 주소를 new_blk로 변경;               (23)
          원본 볼륨 매핑 엔트리의 스냅샷에 대한 ssb(current_ssb)를 1로 변경;   (24)
          데이터 블록 old_blk를 디스크에 기록;                                  (25)
          snap_map_blk과 map_blk를 디스크에 기록;                               (26)
      }
  }
  map_lock의 잠금 해제;                                                         (27)
}
```

그림 5 쓰기 연산의 의사코드

는 데이타 블록이 스냅샷 생성 이전에 사용되었는지 검사한다. 스냅샷 생성 이후 할당된 데이타 블럭에 대해서는 COW가 수행되지 않는다. 따라서, (6)변경된 내용을 데이타 블록을 디스크에 기록하고 (27)매핑 블록에 대한 잠금을 해제하고 종료한다. 데이타 블록이 스냅샷 생성 이전에 할당된 경우에는 FAB와 SSB값을 조사하여 COW가 수행 여부를 판단한다. (15)COW가 이미 수행된 경우(SSB가 1인 경우)에는 (16)데이타 블록을 디스크에 기록하고 (27)매핑 블록에 대한 잠금을 해제하고 수행을 종료한다. (7)COW가 아직 수행되지 않은 경우에는 COW를 수행해야 한다. (8)원본 매핑 블록과 동일 논리 주소의 스냅샷 매핑 블록을 버퍼로 읽어서 스냅샷 매핑 엔트리를 얻고 COW를 수행하기 위해 (9)새로운 물리 데이타 블록을 할당받은 다음 (10)데이타 블록의 내용을 새로 할당받은 데이타 블록에 복사하고 복사된 (13)데이타 블록을 볼륨 디스크에 기록한다. (11)스냅샷 매핑 엔트리가 가리키는 물리 주소를 새로 할당받은 데이타 블록의 주소로 변경하고 COW가 수행되었음을 나타내는 (12)원본 매핑 엔트리의 현재 스냅샷에 대한 SSB값을 1로 변경하고 (14)스냅샷 매핑 블럭을 디스크에 기록하고 원본 매핑 블록을 디스크에 기록한다.

COW 과정이 완료되면 데이타 블록의 내용을 디스크에 기록한다. (27)매핑 블록에 대한 잠금을 해제하고 수행을 종료한다. (17)볼륨의 연산 모드가 정상 모드가 아니고 스냅샷 생성 모드이면 매핑 엔트리를 포함하는 매핑 데이타 블록에 대한 복사가 수행되었는지 검사한다. (18)복사가 완료된 경우에는 COW가 수행되었는지 판단하기 위해 SSB값을 검사한다. SSB값이 1이면 이미 COW가 수행된 경우로 (19)데이타 블록을 디스크에 기록하고 (27)매핑 블록에 대한 잠금을 해제하고 수행을 종료한다. (20)SSB값이 0이면 COW 작업을 수행해야 한다. (21)(22)(23)(24)(25)(26)COW 작업을 수행하고, 데이타 블록을 디스크에 기록하고 (27)매핑 블록의 잠금을 해제하고 수행을 종료한다. 만약 복사가 완료되지 않았다면 COW가 수행되어야 한다.

지금까지 본 논문에서 제안한 쓰기 연산의 수행 알고리즘에 대해 설명하였다. 이미 언급된 것처럼 제안한 기법에서는 COW의 수행 여부를 판단하기 위해 원본 매핑 블록에 대한 접근만 요구된다. 따라서, 스냅샷 생성 이후 첫 번째 변경이 완료된 블록과 새로 할당되는 블

록들에 대해 추가적인 디스크 접근이 요구되지 않기 때문에 쓰기 연산의 성능 저하를 발생시키지 않는다.

3.4 향상된 스냅샷 삭제 알고리즘

기존의 스냅샷 기법에서 성능 저하를 초래하는 요인 중의 하나가 COW의 수행으로 할당된 데이타 블록에 대한 할당을 해제하는 작업이다. 기존의 스냅샷 기법에서는 COW가 수행되었는지 여부를 판단하기 쓰기 연산과 동일하게 스냅샷 매핑 블록에 대한 추가적인 I/O가 요구된다. 쓰기 연산의 경우와 마찬가지로 COW의 수행을 판단하는 문제를 FAB와 SSB로 해결하였다. 제안한 스냅샷 기법에서는 스냅샷 매핑 엔트리를 읽어서 비교하지 않고, 원본 볼륨의 매핑 엔트리만 읽으면 COW의 수행 여부를 판단할 수 있다. 삭제하는 스냅샷 위치의 SSB가 0이면 COW가 수행되지 않은 경우이고, SSB가 1이면 COW가 수행된 경우이다. 스냅샷의 개수가 여러 개인 경우 기존의 스냅샷 기법에서는 스냅샷 개수만큼의 매핑 엔트리를 읽는 I/O 작업을 수행해야 된다. 그러나, 제안한 기법은 스냅샷의 개수에 상관없이 원본 매핑 엔트리 하나만 읽으면 모든 처리를 수행할 수 있다. 따라서, 스냅샷이 많을수록 제안한 스냅샷 기법의 성능이 더욱 좋아진다.

그림 6은 제안한 스냅샷 삭제 알고리즘에 대한 의사코드를 나타낸다. 스냅샷 삭제는 스냅샷 생성 후 COW에 의해 할당된 데이타 블록의 할당을 해제하고, 스냅샷 매핑 테이블을 삭제하는 과정을 수행한다. 스냅샷 삭제가 요청되면 매핑 서버의 (1)볼륨 연산 모드를 스냅샷 삭제 모드(SNAPSHOT_DESTROY)로 변경한다. 스냅샷 삭제 모드로 변경하는 이유는 스냅샷 삭제가 완료되기 전에 발생하는 데이타 블록에 대해 해당 스냅샷에 대한 COW가 수행되는 것을 방지하기 위해서이다. 데이타 블록에 대한 COW의 수행 여부를 판단하기 위해 원본 매핑 테이블 엔트리의 FAB와 SSB값을 검사한다. (2)데이타 블록에 대한 매핑 엔트리를 읽기 위해 원본 볼륨 매핑 엔트리가 저장된 디스크 블록의 위치를 구한다. (3)매핑 블록에 대한 배타 모드의 잠금을 획득하고 (4)디스크 에서 매핑 블록을 읽는다. 매핑 블록의 모든 엔트리에 대해 차례로 COW의 수행 여부를 검사한다. (5)매핑 엔트리의 FAB와 SSB 값을 통해 COW가 수행되었는지 판단한다. (11)COW가 수행되지 않은 블록이면 (12)다음 매핑 엔트리에 대해 검사하는 작업을 계속

```
Snapshot delete operation
{
  Input : original_name, snapshot_name

  for(모든 매핑 서버 호스트에 대해)
     호스트 논리 볼륨의 연산 모드를 SNAPSHOT_DESTROY로 변경;        [1]

  for( original_name 볼륨의 모든 매핑 블록에 대해 )
  {
      map_blk_no = 매핑 엔트리가 저장된 매핑 블록을 구함;           [2]
      map_lock = map_blk_no에 대한 x-mode 잠금 획득;                [3]
      map_blk = 원본 볼륨의 매핑 블록 map_blk_no에 대한 읽기 수행;  [4]

      for( map_blk의 모든 매핑 엔트리에 대해 )
      {
          if( current_ssb == 1 )  // 스냅샷 생성 후 COW가 수행      [5]
          {
              if( next_ssb == 1 || next_snapshot == NULL )          [6]
              {
                  스냅샷 엔트리가 매핑하는 물리 블록에 대한 할당을 해제; [7]
                  current_ssb를 0으로 변경;                         [8]
              }
              else if( next_ssb == 0 )                              [9]
                  current_ssb를 0으로 변경;                         [10]
          }
          else  // 스냅샷 생성 후 COW가 수행되지 않은 경우          [11]
              continue;                                             [12]
      }
      map_blk를 디스크에 기록;                                      [13]
      map_lock의 잠금 해제;                                         [14]
  }
  for(모든 매핑 서버 호스트에 대해)
     호스트 논리 볼륨의 연산 모드를 NORMAL로 변경;                  [15]
}
```

그림 6 스냅샷 삭제 알고리즘의 의사코드

한다. COW가 수행된 데이타 블록이면 데이타 블록의 할당을 해제할 것인지 판단하는 작업을 수행한다. COW가 수행된 데이타 블록의 할당을 해제하는 경우는 두 가지이다. (6)첫째로 다음 스냅샷이 존재하지 않는 경우와 둘째로 다음 스냅샷이 존재할 때 다음 스냅샷 생성 후 동일 데이타 블록에 COW가 발생한 경우이다. 위의 두 가지 경우에 할당된 (7)데이타 블록을 해제하고 (8)매핑 엔트리의 현재 스냅샷에 대한 상태 비트 값인 SSB를 0으로 초기화한다. (9)데이타 블록을 해제하지 않는 경우에는 바로 (10)SSB 값만 0으로 초기화한다. 여기까지 수행하면 하나의 매핑 엔트리에 대한 수행이 완료된다.

매핑 블록에 존재하는 모든 엔트리에 대해 수행이 완료되었는지 검사한다. 완료되지 않은 경우에는 다음 매핑 엔트리에 대한 삭제 작업을 수행한다. 매핑 블록의 모든 엔트리에 대해 수행이 완료된 경우, COW가 한 번 이상 수행 되었으면 (13)매핑 블록을 디스크에 반영하는 작업을 수행한다. (14)매핑 블록에 대한 쓰기 작업이 완료되면 매핑 블록에 대한 잠금을 해제한다. 잠금 해제 후에는 모든 매핑 블록에 대해 수행이 완료되었는지 검사한다. 수행할 매핑 블록이 존재하는 경우에는 다음 매핑 블록을 구하고 매핑 블록에 대한 스냅샷 삭제를 계속 수행한다. 모든 매핑 블록에 대해 수행이 완료된 경우에는 (15)매핑 서버에 존재하는 볼륨의 연산 모드를 정상 모드로 변환하면 스냅샷 삭제 과정이 완료된다. 제안한 기법의 스냅샷 삭제 기법에서도 쓰기 연산과 마찬가지로 COW의 판단 과정에서 스냅샷 매핑 블록에 대한 I/O가 요구되지 않기 때문에 향상된 삭제 성능을 보인다.

4. 구현 및 성능평가

본 논문에서 제안한 스냅샷 알고리즘을 대용량 공유 논리 볼륨 관리 시스템인 SVM(SANtopia Volume Manager)에 적용하여 구현하였다. 제안한 알고리즘과 기존의 스냅샷 기법을 구현하고 성능을 비교 평가하였다. 성능 평가를 수행한 실험 환경은 표 1과 같다. FC Switch는 브로케이드의 16포트 스위치인 Silkworm 2800을 사용하였으며 스토리지 풀(Storage Pool)을 구성하는 디스크 어레이는 RAID, JBOD를 합하여 1TB의 공유 스토리지 공간을 사용하였다. S/W 플랫폼은 레드햇 6.2를 기반으로 커널 버전은 2.2.18을, 4개의 Compaq서버를 연결하여 공유 스토리지 클러스트 환경으로 구성하였다. 실험에서 성능 평가의 척도로 사용한 것은 세 가지이다. 첫 번째 스냅샷 생성과 동시 수행되는 데이타 블록의 변경을 수행하는 쓰기 연산의 수행 시간이다. 스냅샷 생성을 수행하는 프로세스와 쓰기 연산을 수행하는 프로세스를 동시에 수행시키고 스냅샷 생성 시간 및 쓰기 연산의 응답 시간을 측정하였다. 프로세스의 일반 I/O가 중지되는 시간을 볼륨의 크기를 변경시키면서 측정하였다. 두 번째 실험에서는 스냅샷 생성 후 COW가 수행된 이후 발생하는 데이타 블록의 변경에 대한 쓰기 연산의 수행 시간을 측정하였다. 스냅샷의 개수와 변경된 데이타 블록의 개수를 변경시키면서 쓰기 연산의 응답 시간을 측정하였다. 세 번째 실험에서는 스냅샷 삭제 수행 시간을 측정하였다. 스냅샷의 개수와 변경된 데이타 블록의 크기를 변경시키면서 측정을 수행하였다.

표 1 실험 환경

S/W 플랫폼	O. S	Redhat Linux 6.2
	Kernel Version	2.2.18
	Compiler	gcc egcs-2.91.66
H/W 플랫폼	FC Switch	Brocade Silkworm 2800(16port)
	FC Disks	RAID(Eurologic), JBOD(Voyager)
	HBA(Host Bus Adaptor)	QLA2200F(qlogic)
	Memory	512 MB
	CPU	Intel 733/133 MHz (Dual)

4.1 스냅샷 생성과 동시에 수행되는 쓰기 연산에 대한 성능 측정

첫번째 실험은 스냅샷 생성 과정에서 동시에 수행되는 쓰기 연산에 대한 실험이다. 스냅샷 생성과 동시에 데이타 블록의 변경을 수행하고 스냅샷 생성 시간 및 데이타 블록 I/O 시간을 측정하였다. 실험은 스냅샷의 수와 변경되는 데이타 블록의 수를 변화시키면서 측정을 수행하였다. 변경이 발생하는 데이타 블록의 수를 1000, 2000, 5000으로 증가시키면서 데이타 블록에 대한 I/O를 동시에 수행시켰다. 스냅샷 생성 전에 변경이 발생하지 않은 논리 블록에 대한 쓰기 연산과 변경이 수행된 논리 블록에 대한 쓰기 연산을 각각 스냅샷 생성과 동시에 수행하였다. 또한 스냅샷 생성의 대상이 되는 볼륨의 크기도 대용량 측정을 위해 50GB에서 300GB까지 50GB씩 증가시키면서 측정을 수행하였다. 스냅샷의 생성 시간과 쓰기 연산의 수행 시간을 각각 측정하였다. 측정 결과 그림 7처럼 볼륨의 크기가 커질 때 제안한 기법은 쓰기 연산의 응답 시간이 볼륨의 크기와 상관없이 일정한 성능을 보인다. 반면에 기존의 스냅샷 기법은 볼륨 크기에 비례하여 수행 시간이 증가하는 것을 볼 수 있다. 50GB에서는 2배 정도의 성능 차이를 보이고 300GB에서는 10배 이상의 응답 시간의 차이를 보인다. 볼륨의 크기와 비례하여 성능차가 발생하는 것을 볼 수 있다. 제안한 기법에서는 스냅샷 생성 시 I/O에 대한 지연이 발생하지 않고 동시 수행이 가능하다. 그러나 기존

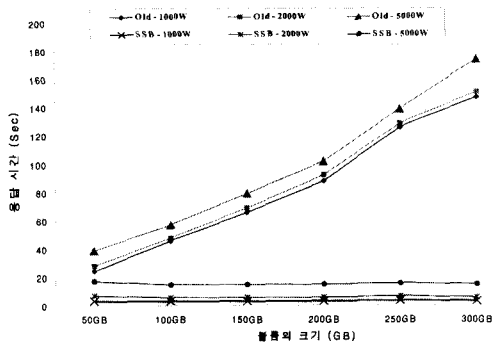

그림 7 스냅샷 생성과 동시 수행되는 쓰기 연산에 대한 응답 시간

의 스냅샷 기법은 생성 중에 쓰기 연산의 동시 수행이 불가능하기 때문에 볼륨의 크기가 커지면 비례하여 쓰기 연산의 수행 시간이 급격히 증가한다. 제안한 기법에서는 동시 수행이 가능하므로 볼륨의 크기에는 영향을 받지 않고 동시 수행되는 쓰기 연산이 일정한 응답 시간을 보인다.

4.2 COW 수행 이후의 쓰기 연산에 대한 성능 측정

두번째 실험은 COW가 수행된 데이타 블록에 대한 쓰기 연산의 성능 측정이다. 스냅샷 생성 전 데이타를 볼륨에 쓰기 연산을 수행하고, 스냅샷 생성 후 동일 데이타 블록에 대해 쓰기 연산을 수행하여 COW를 수행하였다. 다시 동일한 데이타 블록들에 대해 쓰기 연산을 수행하고 I/O 연산의 응답 시간을 측정하였다. 스냅샷의 개수와 변경되는 데이타 블록의 수를 증가시키면서 측정하였다. 스냅샷의 개수는 1~3까지 증가시키면서 수행을 하였다. 변경되는 데이타 블록 역시 1KB 데이타를 5000에서 20000까지 5000씩 증가시키면서 쓰기 연산의 수행 시간을 측정하였다. 그림 8은 쓰기 연산에 대한 응답 시간의 측정 결과를 보여준다. 제안한 스냅샷 기법은 스냅샷 개수와 관계없이 일정한 응답 시간을 나타낸다. 하지만 기존의 스냅샷 기법은 스냅샷 개수가 1개 일 때는 20%에서 100% 정도의 응답 시간이 길어진다. 스냅샷 개수가 증가하면 3개일 때는 2배에서 최대 4배까지 응답 시간이 길어지는 것을 볼 수 있다. 이러한 결과는 COW 이후에 수행되는 쓰기 연산에서 COW의 수행 여부를 판단하기 위해 제안한 기법은 원본 매핑 블록에 대한 1번의 I/O만 요구되지만 기존의 스냅샷 기법은 원본 매핑 블록 이외에 스냅샷 개수 만큼 I/O가 추가로 요구되기 때문이다.

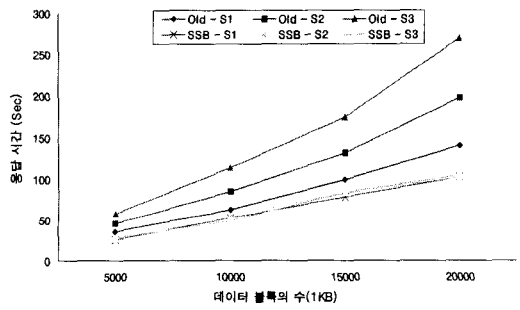

그림 8 COW이후 쓰기 연산의 응답 시간

4.3 스냅샷 삭제에 대한 성능 측정

세번째 실험은 스냅샷 삭제에 대한 성능 평가를 위한 실험이다. 스냅샷 삭제 수행 시간을 측정하였다. 스냅샷의 개수와 볼륨의 크기를 증가시키면서 성능 측정을 수행하였다. 그림 9는 스냅샷 삭제에 대한 성능 측정 결과를 보여준다. 삭제 수행 시간은 볼륨의 크기에 비례하여 증가하였다. 그러나 동일 크기의 볼륨에 대해 제안한 기법이 기존의 스냅샷 기법보다 삭제 시간이 50% 정도 밖에 소요되지 않는다. 또한 제안한 스냅샷 기법은 스냅샷 개수와 관계없이 일정한 수행 시간이 소요되지만 기존의 스냅샷 기법은 스냅샷 개수에 비례하여 수행 시간이 요구되는 것을 볼 수 있다. 이러한 결과는 제안한 기법에서는 COW가 수행되었는지 판단하기 위해 원본 매핑 테이블만 접근하면 되지만, 기존의 매핑 테이블 방식에서는 원본과 스냅샷 매핑 블록을 모두 접근해야 되기 때문이다. 또한 스냅샷의 수가 증가해도 제안한 기법은 원본 볼륨만 접근하면 되므로 스냅샷 개수에 상관없이 동일한 성능을 보이지만, 기존의 스냅샷 기법은 스냅샷이 여러 개 존재하는 경우 COW의 판단을 위해 모든 스냅샷 엔트리의 값을 비교해야만 COW의 수행 여부와 할당 해제 여부를 판단할 수 있다.

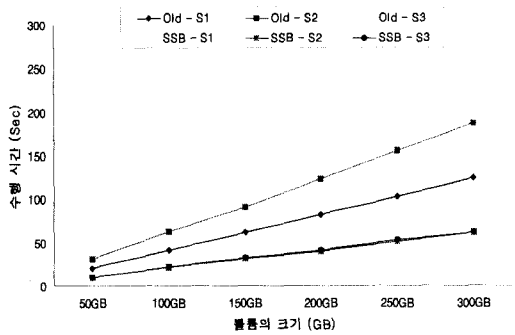

그림 9 스냅샷 삭제 수행 시간

5. 결 론

대용량 스토리지의 응용 분야에서 24*7*365의 엔터프라이즈 환경의 지원에 대한 요구가 점점 커져가고 있다. 또한, 신뢰성과 가용성 보장에 대한 요구도 증가하고 있

다. 저장 공간의 데이타가 커지면 데이타에 대한 백업 수행 시간이 길어지게 된다. 그러나 응용 분야의 특성상 시스템의 서비스를 중단한 백업을 수행하는 cold back-up의 수행이 불가능하고, 시스템의 정지나 서비스의 중단 없이 백업을 수행하는 온라인 백업(hot backup)이 요구된다. 온라인 백업의 지원을 위해서는 매핑 테이블을 기반으로 하는 온라인 스냅샷의 지원이 필수적이다. 하지만 대용량 공유 스토리지 시스템 환경으로 가면서 기존의 스냅샷 기법에서는 몇 가지 성능상의 문제점을 가지게 되었다.

본 논문에서는 스냅샷 상태 비트(Snapshot Status Bit : SSB)와 FAB(First Allocation Bit)를 매핑 엔트리에 도입하여 기존의 스냅샷 기법이 갖는 성능 저하 문제를 해결하였다. 스냅샷 생성 시 모든 해당 볼륨을 사용하는 호스트에서의 I/O 중단 문제를 해결하기 위해 연산 모드 정보를 나타내는 필드를 추가하였다. 제안된 기법에서는 스냅샷 볼륨을 위한 매핑 테이블의 복사를 수행하며 일반 다른 프로세스에서 데이타 블록에 대한 변경이 요청되면 연산 모드가 스냅샷 모드인지, 복사가 완료된 매핑 블록인지 아닌지를 판단하여 COW 또는 일반 쓰기 연산으로 처리한다. 스냅샷 생성 중에도 기존의 스냅샷 기법에서는 지원되지 못했던 I/O의 동시 수행이 가능해졌다. 또한, FAB와 SSB의 사용으로 스냅샷 생성 후 발생하는 데이타 블록의 변경에 대한 쓰기 연산과 스냅샷 삭제 연산의 성능이 향상되었다. FAB와 SSB의 유지로 COW 이후에 발생하는 데이타 블록의 변경 연산에서 스냅샷 매핑 테이블에 대한 I/O를 수행하지 않게 되어서 스냅샷의 수가 많을수록 변경된 데이타 블록이 많을수록 성능이 향상되었다. 스냅샷 삭제 시에도 기존의 기법에서 COW 데이타 블록에 대한 할당 해제로 인한 동시 수행되는 I/O의 성능 저하 문제를 SSB와 FAB를 통해 해결하였다. 성능 평가를 위해 기존의 기법과 제안한 기법을 대용량 공유 스토리지 시스템인 SVM에 적용하여 구현하였다. 스냅샷 생성, 삭제 및 COW 이후에 발생되는 쓰기 연산의 수행 시간을 측정한 결과 제안한 기법이 모든 면에서 2배 이상의 성능 향상을 보인다. 향후 추가 연구 되어야 할 부분은 COW 수행에 대한 성능을 개선하는 것이다. 제안한 스냅샷 기법에서도 COW가 수행될 때는 여전히 여러 번의 I/O가 요구되고 스냅샷이 존재하는 공유 스토리지의 성능을 저하시킨다. 따라서 COW에 대한 개선을 통하여 스냅샷이 존재할 때 수행되는 쓰기 연산의 성능 저하를 최소화해야 된다.

참 고 문 헌

[1] C. Jurgens, "Fibre Channel: A Connection to the Future," IEEE Computer, pp. 82-90, August 1995.

[2] A. F. Benner. 'Fibre Channel: Gigabit Communications and I/O for Computer Network,' McGraw-Hill, 1996.

[3] R. H. Katz, "High-Performance Network and Channel Based Storage," Proceedings of IEEE, Vol.80, No.8, pp.1238-1261, 1992.

[4] Heinz Maulschagen. "Logical Volume Manager for Linux," http://linux.msede.com/lvm/.

[5] David C. Teigland, Heinz Mauelshagen. "Volume Managers in Linux," http://www.sistina.com.

[6] David Teigland. [Slides] "The Pool Driver: A Volume Driver for SANs," http://www.sistina.com.

[7] David Teigland, Ken Preslan, Matthew O'Keefe, "An Overview of The Glolbal File System," http://www.sistina.com.

[8] Paul Massiglia, "Block-Level Incremental Backup," a storage management solution, Veritas Software Corporation, February, 2000.

[9] 신범주, 김경배, 김창수, 김명준, "네트워크 저장 장치를 위한 클러스터 파일 시스템 개발," 정보처리학회지, 8권, 4호, 2001.

[10] C. S. Kim, G. B. Kim and B. J. Shin, "Volume Management in SAN Environment," IEEE ICPADS2001, June 2001.

[11] M. T. OKeefe, "A Persistent Snapshot Device Driver for Linux," 5th Annual Linux Showcase and Conference on in cooperation with USENIX Association, pp.1-16, Oakland, California, November. 2001.

[12] Snapshots in VxFS, http://www.veritas.com.

김 영 호

1999년 충북대학교 정보통신공학과(학사)
2001년 충북대학교 정보통신공학과(석사)
2001년~현재 ETRI 디지털홈연구단 인터넷서버그룹 연구원. 관심분야는 자료저장시스템, 클러스터링 시스템, 데이타베이스 시스템

강 동 재

1999년 인하대학교 전자계산공학과(학사)
2001년 인하대학교 전자계산공학과(석사)
2001년~현재 ETRI 디지털홈연구단 인터넷서버그룹 연구원. 관심분야는 자료저장시스템, GIS, 데이타베이스, 시스템프로그램

박 유 현

1996년 부산대학교 전자계산학과(학사) 1998년 부산대학교 전자계산학과(석사) 2000년 부산대학교 전자계산학과(박사과정 수료). 2000년 1월~2000년 12월 한국국방연구원(KIDA) 자원관리연구부 연구원, 2001년 1월~현재 ETRI 디지털홈연구단 인터넷서버그룹 연구원. 관심분야는 자료저장시스템, 분산시스템, 데이타베이스, 모바일 시스템

김 창 수

1993년 광운대학교 전자계산학과(학사) 1995년 서강대학교 전자계산학과(석사) 1995년~1999년 LG 소프트(현재 LGEDS) 1999년~현재 ETRI 디지털홈연구단 인터넷서버그룹 선임연구원. 관심분야는 데이타베이스 시스템, 자료저장시스템, 클러스터 시스템, 분산 시스템

김 명 준

1978년 서울대학교 계산통계과(학사). 1980년 KAIST 전산학과(석사). 1998년 프랑스 Nancy 제1대학 전산학과(박사). 1986년~현재 ETRI 디지털홈연구단 인터넷서버그룹 책임연구원, 컴퓨터시스템연구부장. 관심분야는 데이타베이스, 분산시스템, 소프트웨어공학