스냅샷 상태 테이블을 이용한 대용량 공유 스토리지를 위한 스냅샷

김영호[O] 정성인 김명준
한국전자통신연구원 컴퓨터시스템연구부
{kyh05[O], sijung, joonkim }@etri.re.kr

Snapshot Technique using Snapshot Status Table for Shared Storage System supporting Large Capacity

Young-Ho Kim[O], Sung-In Jung, Myung-Joon Kim
Dept. of Computer System, Electronic Telecommunication Research Institute(ETRI)

요 약

본 논문에서는 대용량 스토리지를 공유하는 스토리지 클러스터 시스템을 위한 매핑 테이블 기반의 스냅샷 기법을 제안한다. 스냅샷 상태 비트를 이용한 디스크 기반의 스냅샷 기법을 개선한 스냅샷 상태 테이블을 메모리에 유지하는 스냅샷 기법을 제안한다. 스냅샷 상태 테이블은 최초 할당 비트(FAB:First Allocation Bit)와 스냅샷 상태 비트(SSB:Snapshot Stautus Bit)로 구성되고 기존 스냅샷 기법이 갖는 문제점들을 해결한다. 스냅샷 생성 시 대상 저장 장치에 대한 I/O의 중단없이 데이터의 일관성을 보장한다. 또한 쓰기 연산 수행 시 COW의 수행 여부 판단을 스냅샷 상태 테이블의 FAB와 SSB를 이용하여 매핑 블록에 대한 추가적인 I/O를 없앤다. 스냅샷 삭제 시 매핑 블록에 대한 접근 없이 COW 수행 여부 판단을 처리하여 성능을 향상시킨다.

1. 서 론

최근 엔터프라이즈 시스템에서는 고속 대용량의 데이터 처리 이외에 데이터에 대한 가용성 및 신뢰성이 크게 요구되고 있다. 가용성 및 신뢰성의 요구의 증가로 시스템의 중단 없이 데이터의 백업을 수행할 수 있는 온라인 백업의 중요성이 더욱 부각되고 있다[7]. 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 기법[4][6]을 SAN을 기반으로 하는 대용량의 공유 스토리지 시스템[1][2][3]에 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 동안 대상 공유 스토리지에 대한 데이터 변경 연산의 수행이 불가능하다. 또한 스냅샷 생성 후 데이터 블록에 대한 변경의 처리를 수행하는 Copy-on-Write(COW) 이후에 발생하는 쓰기 연산 수행 시 COW가 수행되었는지 판단하기 위해 스냅샷 매핑 블록에 대한 추가적인 I/O를 요구하게 되어 쓰기 연산의 성능 저하를 초래한다. 스냅샷 삭제 시에도 COW에 의해 할당된 데이터 블록을 해제하는 과정을 수행하기 때문에 동시 수행되는 쓰기 연산의 성능이 저하된다.

본 논문에서는 스냅샷 상태 비트(SSB:Snapshot Stautus Bit)와 최초 할당 비트(FAB: First Allocation Bit)로 구성된 스냅샷 상태 테이블(SST : Snapshot Status Table)을 메모리에 유지하여 기존의 스냅샷 기법이 대용량 공유 스토리지 클러스터 환경에서 갖는 문제점들을 해결한다. 또한 스냅샷 상태 비트를 사용하는 디스크 기반의 스냅샷 기법[5]을 개선하여 성능을 더욱 향상시켰다. 디스크 기반의 스냅샷 기법에서 COW 수행 후 FAB나 SSB 값을 변경하고 디스크에 저장하는 작업이 쓰기 연산의 성능 저하를 가져오는데, 제안하는 기법에서는 메모리의 스냅샷 상태 테이블에 FAB, SSB의 변경을 수행하여 디스크 I/O 횟수가 적어진다. 스냅샷 삭제 시에 COW의 수행 여부를 판단하기 위해 디스크 기반의 스냅샷 기법에서는 원본 매핑 블록에 대한 I/O가 요구되었다. 그러나 제안하는 기법에서는 메모리 상의 스냅샷 상태 테이블을 통해 원본 매핑 블록에 대한 I/O도 요구되지 않는다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로서 매핑 테이블 기반의 스냅샷 기법 및 대용량 공유 저장 장치에서 기존의 스냅샷 기법의 문제점에 대해 설명한다. 3장에서는 제안하는 스냅샷 기법의 특징, 구조 및 생성/삭제 및 쓰기 연산의 알고리즘에 대해 설명한다. 4장에서는 결론 및 향후 추가 연구가 필요한 부분에 대해 기술한다.

2. 관련 연구

본 장에서는 매핑 테이블 기반의 스냅샷 기법에 대해 살펴본다. 매핑 테이블 기반의 스냅샷 기법[4][6]의 개념과 데이터의 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 스냅샷 기법

스냅샷은 사용자가 원하는 특정 시점에서의 데이터 상태를 저장, 유지시켜주는 기법이다. 스냅샷 기법은 데이터 전체가 아닌 데이터에 대한 이미지만을 복사해서 스냅샷 생성 시점의 데이터를 그대로 유지하게 된다. 스냅샷은 논리 볼륨 관리자의 가상 디스크 구조에서 데이터를 중복시키는 강력한 툴이다. 또한 스냅샷은 파일 시스템이나 디스크 파티션이 사용 중인 온라인 상태에서 원하는 시점의 데이터를 백업하는 온라인 백업을 가능하게 지원한다. 따라서 원하는 시점으로의 데이터에 대한 회귀가 가능하다.

2.1.1 Copy-on-Write 메카니즘

스냅샷 생성 후 데이터 블록에 대한 변경이 발생하면 스냅샷 시점의 데이터를 유지할 수 있도록 처리하는 과정을 수행해야 한다. 이처럼 데이터의 일관성 유지를 위해 수행하는 작업을 Copy-on-Write(COW)라 한다. COW는 동일 데이터 블록에 대해 오직 한 번만 수행된다. 새로운 블록을 할당 해 스냅샷 시점의 데이터를 복사하고 스냅샷의 매핑

엔트리가 새로 할당된 데이터 블록을 매핑하도록 매핑 엔트리 값을 변경한다. (그림 1)은 매핑 테이블 기반의 스냅샷 기법에서 copy-on-write의 수행 과정의 예를 나타낸다.

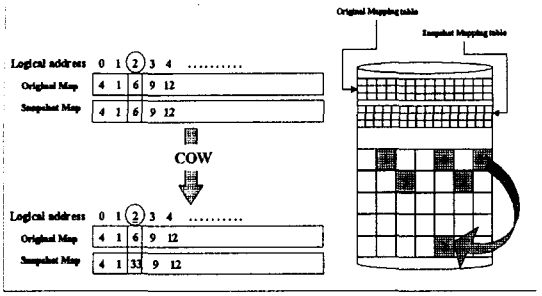

그림 1. Copy-on-Write 수행 예

2.2 기존 스냅샷 기법의 문제점

기존 스냅샷 기법과 스냅샷 상태 비트를 이용한 디스크 기반 스냅샷 기법이 대용량 공유 스토리지 시스템에서 사용될 때 나타나는 성능상의 문제점에 대해 기술한다.

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

스냅샷 생성 후 데이터 블록의 변경이 요청되면 COW 처리를 수행해야 한다. 쓰기 연산에 대한 COW의 수행 여부를 판단하는 과정은 스냅샷이 삭제될 때까지 계속 요구된다. 기존의 스냅샷 기법에서는 스냅샷 엔트리와 원본 엔트리를 모두 디스크로부터 읽어서 두 엔트리가 매핑하는 데이터 블록이 동일한지 비교하여 판단한다. 즉, 스냅샷 생성 이후에 발생하는 모든 데이터 변경 연산에서는 스냅샷 매핑 엔트리를 읽기 위한 추가적인 디스크 I/O를 계속 수행해야 한다. 디스크 기반 스냅샷 기법[5]에서는 스냅샷 매핑 블록에 대한 I/O 요구는 없앴지만, SSB나 FAB 값이 변경되면 디스크에 저장하는 추가적인 I/O를 여전히 가지게 된다.

2.2.2 스냅샷 삭제 시 성능 저하

스냅샷 삭제가 요청되면 COW에 의해 새로 할당된 데이터 블록에 대한 할당을 해제하는 작업 수행 과정을 거친다. 기존의 스냅샷 기법에서는 COW의 수행 여부를 판단하기 위해 쓰기 연산의 수행과 마찬가지로 스냅샷 매핑 블록에 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. [5]에서는 스냅샷 매핑 블록에 대한 추가적인 I/O는 없지만, SSB나 FAB 값을 초기화한 후 디스크에 반영하는 추가적인 I/O 작업이 여전히 요구된다.

3. 제안하는 스냅샷 기법의 구조 및 알고리즘

기존의 스냅샷 기법은 관련 연구에서 살펴본 것처럼 대용량의 공유 스토리지 클러스터 환경에 적용될 때 I/O 성능을 저하시키는 등의 몇 가지 문제점을 가지고 있다. 디스크 기반의 스냅샷 기법에서는 스냅샷 매핑 블록에 대한 I/O 문제는 해결하였지만, 여전히 추가적인 디스크 I/O 문제가 발생한다. 이런 문제들을 해결하기 위해 제안하는 스냅샷 기법에서는 스냅샷 상태 비트(SSB: Snapshot status Bit)와 최초 할당 비트(FAB: First Allocation Bit)로 구성된 스냅샷 상태 테이블(SST : Snapshot Status Table)을 메모리에 유지하는 개념을 도입하였다. 이 장에서는 스냅샷 상태 테이블을 기반으로 하는 제안하는 스냅샷 기법의 특징과 기존 스냅샷 기법이 갖는 문제점들을 해결하기 위한 스냅샷 연산 알고리즘에 대해 설명한다. 스냅샷 상태 테이블의 구조에 대해 설명하고, 알고리즘을 적용한 스냅샷 생성 및 삭제, 쓰기 연산의 수행 과정에 대해 자세히 설명한다.

3.1 스냅샷 상태 테이블(Snapshot Status Table)

제안하는 스냅샷 기법의 가장 큰 특징은 메모리에 스냅샷의 상태를 나타내는 스냅샷 상태 테이블(SST : Snapshot Status Table)을 유지하는 것이다. 스냅샷 상태 테이블의 엔트리는 두 개의 상태 비트들로 이루어진다. 스냅샷 상태 비트(SSB: Snapshot Status Bit)와 최초 할당 비트(FAB: First Allocation Bit)이다. COW의 수행 여부의 판단을 위해 요구되는 매핑 블록에 대한 추가적인 I/O를 없애기 위해 개념을 도입했다. (그림 2)는 제안하는 스냅샷 기법의 스냅샷 상태 테이블 및 엔트리 구조를 나타낸다. 기존의 스냅샷 기법에서는 COW의 수행 여부를 판단하기 위해 항상 원본 매핑 블록과 스냅샷 매핑 블록에 대한 접근이 요구되었다. 제안하는 기법에서는 매핑 블록의 추가적인 디스크 접근으로 인해 발생하는 쓰기 연산 및 스냅샷 삭제 연산의 성능을 향상시키기 위해 메모리에 데이터 블록에 대한 상태를 나타내는 비트 정보를 유지하여 성능 저하 문제를 해결하였다. FAB는 엔트리마다 1비트 씩 할당되고, SSB는 엔트리마다 생성되는 스냅샷의 개수만큼 할당된다.

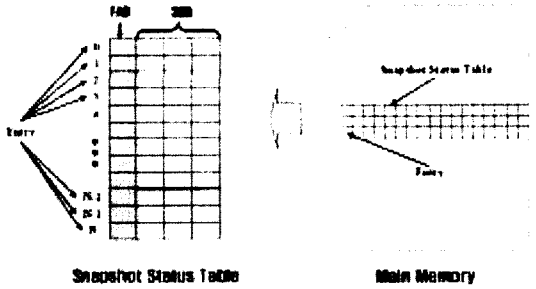

그림 2. 스냅샷 상태 테이블의 구조

SSB는 매핑 엔트리가 매핑하는 데이터 블록의 스냅샷 생성 이후의 상태를 나타내는 비트로 1이면 스냅샷 생성 후 데이터 블록에 대한 변경 연산의 수행으로 COW가 수행되었다는 의미이다. 스냅샷 생성 후 데이터 블록의 변경으로 COW가 수행되면 해당 매핑 엔트리의 SSB값을 1로 변경한다.

FAB는 데이터 블록 중 스냅샷 생성 이전에는 사용되지 않다가 스냅샷 생성 이후 처음 사용되는 데이터 블록들을 구별하기 위한 비트이다. 스냅샷 생성 후 처음 사용되는 데이터 블록들에 대한 구별과 COW의 수행을 방지하기 위해 매핑 엔트리의 첫 번째 비트를 FAB로 할당하여 사용하고 있다. SSB와 FAB의 실제 사용에 대해서는 다음의 스냅샷 연산 알고리즘에서 자세히 언급한다.

3. 2 스냅샷 생성 알고리즘

```
Snapshot Creation operation ( ori_vol_name, snapshot_name )

For ( all host loading original volume )
  change op_mode with SNAPSHOT_CREATE;

allocate and initialize the Snapshot Status Table in memory;

For ( all mapping block of original volume)
  map_lock = obtain x-lock on original mapping blk;
  If ( map_lock != NULL )  // in case of obtaining map_lock
    copy original mapping blk to the snapshot mapping blk ;
    Release x-lock on original mapping blk;
  Else // couldn't get the map_lock
    continue;

For ( all host loading origin_vol )
  change op_mode of original volume with NORMAL;
```

그림 3. 스냅샷 생성 알고리즘

제안하는 기법에서는 연산 모드를 NORMAL과 SNAPSHOT_CREATE의 두 가지 모드로 구분한다. 연산 모드는 스냅샷이 생성되는 매핑 테이블의 복사 중에 쓰기 연산이 요청되는 경우 COW를 수행하여 스냅샷 생성 시점의 데이터를 유지하도록 보장한다. 제안하는 스냅샷의 생성 기법은 매핑 테이블을 복사할 때 원본 볼륨에 대한 I/O를 동시에 처리한다. 단지, 매핑 서버 호스트의 연산 모드를 스냅샷 생성 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. 스냅샷 생성 시 메모리에 스냅샷 상태 테이블 영역을 할당하고 초기화한다. (그림 3)은 제안하는 스냅샷 생성 알고리즘의 의사코드를 나타낸다.

3.3 쓰기 연산 알고리즘

제안하는 스냅샷 기법에서는 스냅샷이 존재할 때 발생하는 데이터 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, SSB와 FAB를 이용하여 쓰기 연산의 성능 저하 문제를 해결하였다. 제안하는 기법에서는 COW의 수행 여부를 판단하기 위해 매핑 블록에 대한 접근 없이 메모리의 스냅샷 상태 테이블을 통해 처리된다. 따라서, 스냅샷 생성 이후 첫 번째 변경이 완료된 블록과 새로 할당되는 블록들에 대해 추가적인 디스크 접근이 요구되지 않기 때문에 쓰기 연산의 성능 저하를 발생시키지 않는다. (그림 4)는 제안하는 스냅샷 기법에서의 쓰기 연산의 알고리즘을 나타낸다.

```
Write operation ( logical_blkno, volume_name )

ori map blk = find a mapping block stored map entry
map_lock = obtain x-lock on map_etnry_block;
read the ori map blk from volume disk;

If ( snap_count != NULL )  // snapshot is initiated
  If ( FAB ) // write is performed after snapshot creation
     update and write old_blkno to the disk;
  Else If ( ! SSB )  // COW is not performed, yet
     new_blkno = allocate data block for COW is performed;
     copy data of old_blkno to the new_blkno;
     modify snap map entry to map the new_blkno;
     If ( ori_map_entry == NULL )  // snapshot is initiated
       set the FAB value with 1;
     Else
       set the SSB value with 1;
     write old_blkno and snap_map_blk onto the disk;
  Else // COW is already performed
     write old_blkno onto the disk;
Else If ( op_mode == SNAPSHOT_CREATE )
  If ( SSB )  // COW is already performed
     write old_blkno to the disk;
  Else
     new_blkno = allocate data block for COW is performed;
     copy data of old_blkno to the new_blkno;
     modify snapshot mapping entry to map the new_blkno;
     set the SSB value with 1;
     write old_blkno and new_blkno to the disk;
     write snap map blk  onto the disk;
release map lock;
```

그림 4. 쓰기 연산 알고리즘

3.4 스냅샷 삭제 알고리즘

쓰기 연산의 경우와 마찬가지로 COW의 수행을 판단하는 문제를 FAB와 SSB로 해결하였다. 제안하는 기법에서는 메모리 상의 스냅샷 상태 테이블의 FAB와 SSB를 통해 COW의 수행 여부를 판단할 수 있다. 삭제하는 스냅샷 위치의 SSB가 1이면 COW가 수행된 경우이다. 스냅샷의 개수가 여러개인 경우 기존의 스냅샷 기법에서는 스냅샷 개수만큼의 매핑 엔트리를 읽는 I/O 작업을 수행해야 된다. 그러나, 제안하는 기법은 [5]에서 요구되던 원본 매핑 블록에 대한 추가 I/O도 요구되지 않는다. (그림 5)는 제안하는 스냅샷 삭제 알고리즘에 대한 의사코드를 나타낸다.

```
Snapshot Destruction operation( ori_vol_name, snapshot_name )

For ( all mapping blocks of original volume )
  ori map blk = find a mapping block which including map entry
  map_lock = obtain x-lock on ori map blk;
  read the ori map blk from volume disk;

  For ( all mapping entries of ori map blk)
    If ( current_ssb )    // COW is performed after snapshot
creation
       If ( next_ssb  || ! next_snapshot )
          release allocation of data block
          set current_ssb  with 0;
       Else If ( ! next_ssb )
          set current_ssb with 0;
    Else  // data block isn' t modified
       continue;
  release map lock;

If ( snap_count == NULL )  // snapshot is not existed
    release Snapshot Status Table in the memory;
```

그림 5. 스냅샷 삭제 알고리즘

4. 결론 및 향후 과제

본 논문에서는 디스크 기반의 스냅샷 상태 비트를 개선한 메모리 기반의 스냅샷 상태 테이블을 도입하여 기존 스냅샷 기법이 갖는 성능 저하 문제를 해결하였다. 스냅샷 상태 테이블의 엔트리는 스냅샷 상태 비트(SSB)와 최초 할당 비트(FAB)로 구성된다. COW 수행 여부의 판단을 위해 메모리에 유지되는 FAB와 SSB의 사용으로 스냅샷 생성 후 발생하는 쓰기 연산과 스냅샷 삭제 연산의 성능이 향상되었다. FAB와 SSB의 사용으로 COW 이후에 발생하는 쓰기 연산에서 스냅샷 매핑 테이블에 대한 I/O를 수행하지 않게 되었다. 스냅샷 삭제 시에는 COW의 판단을 위해 매핑 블록에 대한 I/O가 전혀 요구되지 않고 메모리의 스냅샷 상태 테이블만 접근하면 된다. 향후에는 스냅샷 상태 테이블의 안정성 및 일관성 있는 데이터를 유지하는 것에 대한 연구가 필요하다.

참고 문헌

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

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

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

[4] 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.

[5] Y.H.Kim, D.J.Kang, Y.H.Bak, M.J.Kim, 밧napshot Technique for Shared Large Storage in SAN Environments? CCN2002, pp520-525, Boston, November 2002.

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

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