SAN 환경에서의 전역 버퍼를 이용한 효율적인 회복 기법

박 춘 서[†]·김 경 배[††]·이 용 주[†]·박 선 영[†]·신 범 주[†††]

요 약

SAN(Storage Area Network) 환경에서 대용량 파일 시스템 지원을 위한 회복 기법으로 저널링(journaling) 방법을 사용하였다. 그러나, 기존의 저널링 방법에서는 다른 호스트가 변경된 메타 데이터에 접근하는 경우 메타 데이터를 디스크에 반영하고 읽어 가기 때문에 디스크 입/출력 횟수가 증가하는 문제점을 가지고 있다. 또한, 기존의 저널링 기법에서는 메타 데이터에 대한 회복만을 지원하고 실제 중요 데이터에 대한 회복을 지원하지 않는다. 본 논문에서는 SAN 환경에서 전역 버퍼(Global Buffer)를 이용하여 다른 호스트에 존재하는 변경된 메타 데이터에 접근할 경우, 메타 데이터에 대한 디스크 입/출력을 하지 않고, 네트워크를 통해 메타 데이터를 요청한 호스트에 직접 전송해줌으로써 디스크 입/출력 수를 줄이는 회복 기법을 제안한다. 전역 버퍼를 이용한 메타 데이터 관리 기법을 통해 SAN환경의 리눅스 공유 디스크 파일 시스템에서 메타 데이터 변경 연산 시 트랜잭션 처리 속도를 향상시킬 수 있는 장점을 얻을 수 있다. 또한, 본 논문에서 제안한 기법은 메타 데이터뿐만 아니라 일반 중요 데이터에 대해서도 저널링함으로써 시스템이 실패한 경우에도 중요 데이터에 대한 정보를 유지하는 장점이 있다.

An Efficient Recovery Technique using Global Buffer on SAN Environments

Choon Seo Park[†]·Gyoung Bae Kim[††]·Yong Ju Lee[†]·
Seon Yeong Park[†]·Bum Joo Shin[†††]

ABSTRACT

The shared disk file systems use a technique known as file system journaling to support recovery of metadata on the SAN(Storage Area Network). In the existing journaling technique, the metadata that is dirtied by one host must be updated to disk space before some hosts access it. The system performance is decreased because the disk access number is increased. In this paper, we describe a new recovery technique using a global buffer to decrease disk I/O. It transmits the dirtied metadata into the other hosts through Fibre Channel network on the SAN instead of disk I/O and supports recovery of a critical data by journaling a data as well as metadata.

키워드 : SAN(Storage Area Network), 메타 데이터(metadata), 회복(recovery), 저널링(journaling)

1. 서 론

최근 몇 년 사이 인터넷 사용의 확장과 업무 환경의 급속한 변화와 함께 처리해야 할 데이터 양이 폭발적으로 증가하고 멀티미디어 데이터와 같이 매우 큰 파일을 다루게 됨에 따라 파일 시스템에 대량의 데이터를 저장하는 것이 필요하게 되었다. 그러나, 기존의 대형 단일 시스템에서는 기하급수적으로 늘고 있는 엄청난 양의 데이터를 처리하는데 한계가 있다. 그 결과로 SAN 이라는 개념이 등장하게 되었다. SAN이란 연결된 서버에 관계없이 원거리에 분산된 저장 장치들을 파이버 채널(fibre channel)[1]로 하나의 네트워크로 구성하여 SAN에 연결된 네트워크 노들이 공유할 수 있도록 해주는 초고속 통신망이다. 이러한 SAN 환경을 통해서 높은 시스템 성능(high performance), 높은 확장성(high scalability), 높은 가용성(high availability) 및 공유성(sharability)등을 제공한다. SAN 환경을 기반으로 하는 시스템으로 GFS(Global File System)[2-3], Veritas[4], SANergy[5]들이 국외에서 연구되고 있다. 국내에서는 한국전자통신연구원에서 개발중인 SANtopia[6-7] 시스템이 활발히 연구되고 있다.

이러한 SAN 환경에서 호스트들이 공유하는 데이터와 메타 데이터 관리 기법으로 변경된 내용을 디스크에 바로 반영하지 않고 캐쉬를 이용한다. 캐쉬를 이용한 기법은 디스크에

† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어연구소 컴퓨터시스템연구부 연구원
†† 종신회원 : 한국전자통신연구원 컴퓨터소프트웨어연구소 컴퓨터시스템연구부 선임연구원
††† 정 회 원 : 한국전자통신연구원 컴퓨터 소프트웨어기술연구소 컴퓨터시스템연구부 책임 연구원
논문접수 : 2001년 10월 17일, 심사완료 : 2001년 12월 20일

접근하는 시간을 줄여 빠른 응답시간을 제공하는 장점이 있다. 그러나, 시스템에 장애가 발생하였을 경우 디스크와 캐쉬에 있는 메타 데이터를 동시에 수정하지 못하므로 파일 시스템의 일관성이 깨지게 되는 문제점이 발생한다.

기존의 대부분의 단일 리눅스 파일 시스템에서는 시스템에 문제가 발생되는 경우 fsck를 이용하여 비일관성의 원인을 체크하고 해결한다. 그러나, fsck를 수행하는 데는 파일 시스템의 크기에 비례하여 많은 시간이 소요되고, 시스템이 오프-라인(off-line) 상태로 수행해야 한다. 이것은 SAN 환경을 기반으로 한 네트워크 연결형 대용량 공유 파일 시스템에서는 치명적인 단점이 된다. 따라서 SAN 환경을 기반으로 하는 공유 디스크 리눅스 파일 시스템의 회복 기법으로 파일 시스템의 고장으로부터 복구시간을 줄이고, 가용성을 증가시키기 위한 기법으로 저널링 기법[8-11]을 사용하였다.

저널링 기법은 파일 시스템에 대한 연산이 발생하는 경우 비정상적인 파일 시스템의 이상을 대비하여 슈퍼 블록, 디렉토리 엔트리, 아이-노드(inode) 정보 등과 같은 메타 데이터에 대한 정보를 저널 공간에 기록하여, 파일 시스템의 이상이 발생한 경우에 로깅된 메타 데이터를 이용하여 회복을 진행하는 기법이다. SAN 환경에 기반을 둔 GFS 시스템에서도 회복을 위해 저널링 방법을 사용하였다. 그러나, 기존의 저널링 방법에서는 변경된 메타 데이터에 다른 호스트에서 접근하려고 하면 메타 데이터를 디스크에 반영하고 나서 호스트가 디스크에 접근해서 메타 데이터를 읽어 오게 된다. 따라서 디스크 접근수가 증가되어 시스템 성능이 떨어지는 단점이 발생한다. 본 논문에서는 이러한 단점을 극복하기 위해 변경된 메타 데이터에 대해서 다른 호스트가 요청하는 경우에 메타 데이터를 디스크에 반영하지 않고 전역 버퍼에 요청하여 메타 데이터를 네트워크를 통해서 요청한 호스트에 직접 전송하게 된다. 디스크에 접근하는 시간보다 네트워크 전송속도가 빠르기 때문에 시스템 성능을 향상시킬 수 있는 장점이 있다.

또한, 기존의 저널링 방법은 메타 데이터에 대한 회복을 보장하지만 데이터에 대해서는 회복을 보장하지 않는 단점이 있다. 본 논문에서는 메타 데이터뿐만 아니라 중요한 데이터에 대해서도 저널링 기법을 이용하여 회복을 보장을 해준다.

2. 관련 연구

본 절에서는 기존 리눅스 파일 시스템의 메타 데이터 회복 기법인 fsck의 특징과 fsck의 단점을 보완하기 위한 저널링 파일 시스템인 JFS(Journaled File System)[8] 파일 시스템과 GFS 파일 시스템의 특징을 설명한다.

2.1 fsck

파일 시스템의 역할은 데이터를 저장하고 검색하고 처리하기 위해서 존재한다. 이를 위해서 파일 시스템은 모든 데이터가 조직화되고 액세스가 가능한 상태로 되어있는 내부 데이터 구조를 가지고 있어야 한다. 그 내부 데이터 구조를 메타 데이터라고 한다.

기존의 리눅스 파일 시스템에서 시스템 성능을 향상시키기 위해 메타 데이터에 대한 캐쉬를 사용한다. 그러나, 시스템에 장애가 발생하면 메타 데이터의 일관성이 깨지게 될 가능성이 있다. 파일 시스템은 메타 데이터에 대해서 일관성을 유지시켜 주어야 한다. 일관성이 깨진 메타 데이터에 대한 회복 방법으로 기존의 단일 리눅스 파일 시스템은 fsck 방법을 사용하였다. 그러나, fsck의 방식은 파일시스템의 일관성을 확인하기 위해서 모든 메타 데이터를 검사해야 한다. 모든 메타 데이터에 대해 완벽한 일관성 체크를 하는데 큰 파일 시스템인 경우 엄청난 시간이 필요하게 된다. 또한 fsck는 오프-라인(off-line) 상태에서 동작을 하기 때문에 SAN 환경과 같이 여러 서버들이 하나의 네트워크로 연결되어 저장 장치를 공유하는 파일 시스템에서는 치명적인 단점이 된다.

2.2 JFS(Journaled File System)

JFS 파일 시스템에서 메타 데이터의 회복을 위하여 저널링 기법을 사용한다. JFS는 저널링 기법을 사용함으로써 시스템에 장애가 발생하였을 경우 기존의 리눅스 파일 시스템보다 빠르게 회복을 할 수 있는 장점이 있다. 기존의 리눅스 파일 시스템의 회복 방법인 fsck는 모든 메타 데이터에 대해서 검사를 해서 일관성이 깨진 부분을 발견하여 회복을 하기 때문에 속도가 많이 걸리는 단점이 있다. JFS 파일 시스템은 변경된 메타 데이터에 로그 정보를 저널 공간에 기록하게 된다. 또한, 이러한 메타 데이터 변경 연산을 트랜잭션 처리 해줌으로써 원자성(atomicity)을 보장해준다. 이 시스템에 장애가 발생하면 적절한 트랜잭션에 대한 로그 레코드를 재실행(replay)함으로써 회복을 할 수 있다. JFS 파일 시스템은 메타 데이터를 모두 검사하는 것이 아니라 최근에 파일 시스템의 의해서 생성된 로그 레코드만을 검사하기 때문에 회복 시간이 빠르다.

그러나, JFS 파일 시스템은 단일 리눅스 환경에서의 로컬 파일 시스템이기 때문에, SAN 환경과 같이 여러 호스트가 디스크를 공유하는 형태의 리눅스 공유 디스크 파일 시스템에서는 적합하지 않다.

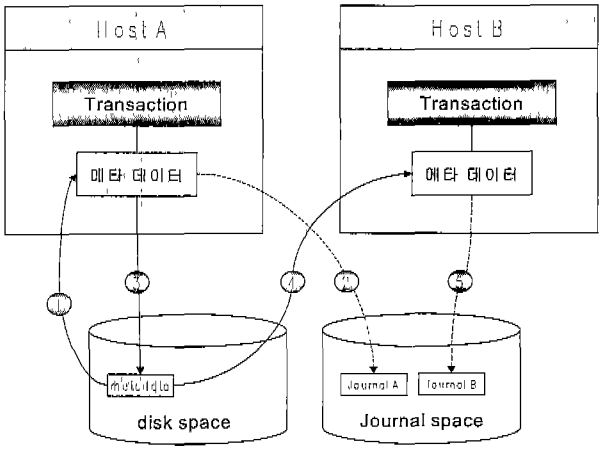
2.3 GFS(Global File System)

GFS 파일 시스템은 SAN 환경을 기반으로 한 공유 디스크 파일 시스템이다. GFS 시스템에서 메타 데이터의 회복 기법으로 저널링 방법을 사용한다. 저널링 방법을 사용함으로써 GFS 파일 시스템과 같은 공유 디스크 파일 시스템에서는 회복 시간을 줄이고 회복 중에도 시스템 온-라인(on-line) 상태를 유지하는 장점을 가지고 있다. GFS 파일 시스템에서

는 각각의 호스트가 자신의 저널 공간을 가지고, 이 저널 공간은 잠금에 의해서 다른 호스트의 접근을 보호한다. 그러나, 어느 호스트가 장애가 발생했을 경우 다른 호스트가 장애가 발생한 호스트의 저널 공간에 접근하여 회복을 실행하게 된다.

GFS 파일 시스템에서는 저널링 기법을 구현하기 위해 트랜잭션 관리자와 로그 관리자를 이용한다. 메타 데이터 변경을 일어나는 파일 연산에 대해서 트랜잭션 처리를 해줌으로써 원자성을 보장해 주고, 로그 관리자는 트랜잭션 관리자로부터 수정된 메타 데이터 정보를 전달받아 자신의 저널 공간에 로그 정보를 기록한다.

GFS 파일 시스템에서 변경된 메타 데이터에 대해서 다른 호스트에서 접근하고자 하면 메타 데이터를 디스크에 기록되기 전까지는 사용할 수 없다. 즉, 메타 데이터가 디스크에 기록되고 나서야 다른 호스트가 해당 메타 데이터에 접근 할 수 있다. 따라서 메타 데이터 변경 후에 디스크 입/출력이 두 번 발생하게 된다. 즉, 저널 공간에 기록하는 로그와 원래 메타 데이터를 모두 디스크에 기록해야 하는 단점이 발생한다. 이 과정을 (그림 1)을 통해서 자세히 살펴보면 다음과 같다.



(그림 1) GFS 파일 시스템에서의 동일한 메타 데이터에 접근시 저널링 방법

호스트 A가 메타 데이터를 디스크로부터 읽어 버퍼에 적재를 한다(①). 메타 데이터를 변경하고 호스트 A의 저널 공간에 로그를 기록한다(②). 이때 호스트 B가 호스트 A가 변경한 메타 데이터에 접근하려 하면, 호스트 A는 실제 메타 데이터를 디스크에 기록한다(③). 디스크에 기록하고 난 후 호스트 B는 디스크에 접근해서 메타 데이터를 버퍼로 읽어오게 된다(④). 호스트 B는 메타 데이터를 변경하고 트랜잭션이 끝나면 호스트 B의 저널 공간에 기록하게 된다(⑤). 이렇게 되면 디스크 입/출력이 다섯 번 발생하게 되어 시스템 성능을 저하시키는 단점이 있다.

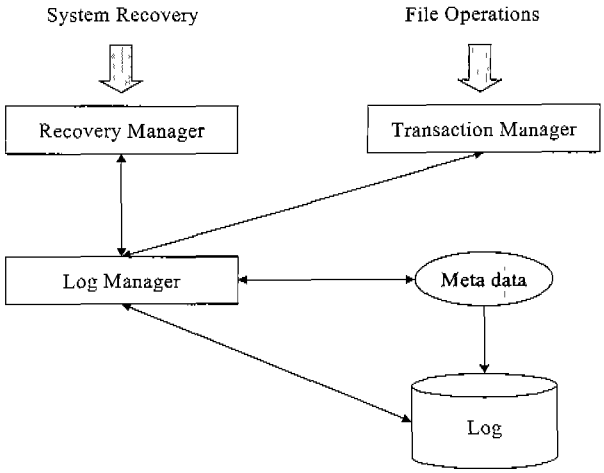
GFS 파일 시스템에서는 트랜잭션이 모두 끝나면 메타 데이터 로그 정보를 바로 디스크에 기록하는 것이 아니라 일정 수 이상이 될 때까지 로그 정보를 모아 두는 비동기적 로깅 방법을 사용한다. 이는 트랜잭션이 끝날 때마다 로그에 대한 기록을 하지 않고, 많은 양의 로그를 모아서 같이 기록해서 입/출력 빈도를 줄이기 위한 방법이다. 그러나, 트랜잭션이 완료(commit)되었지만 로그가 저널 공간에 기록되지 않은 상태에서 시스템에 장애가 발생해서 비정상적으로 종료된 경우에는 이미 완료된 트랜잭션에 대해서도 로그가 저널 공간에 기록되지 않았기 때문에 회복을 하지 못하고 무시되는 단점이 있다. 또한, GFS 파일 시스템에서는 메타 데이터에 대한 회복만을 지원하기 때문에 시스템 장애가 발생하였을 경우 중요 데이터는 회복되지 않는다.

3. SAN환경에서 전역 버퍼를 이용한 파일 시스템의 회복 기법

본 절에서는 본 논문에서 제시하는 회복 관리자의 전체 구성도를 살펴보고 각 관리자에 대해서 설명한다. 변경된 메타 데이터에 대해서 다른 호스트가 접근하였을 경우 GFS 시스템의 단점을 극복하기 위해 본 논문에서는 전역 버퍼를 이용한 메타 데이터 관리 기법을 제안한다. 또한, 비동기적 문제점을 개선하고 메타 데이터뿐만 아니라 중요 데이터 대해서도 저널링 기법을 적용하여 시스템 장애 시 중요한 데이터에 대해서도 회복을 보장해주는 회복 기법을 제안한다.

3.1 회복 관리자 전체 구성도

본 논문에서는 제시한 SAN 환경의 리눅스 공유 디스크 파일 시스템의 회복을 담당하는 회복 관련 모듈을 살펴보면 (그림 2)와 같다.



(그림 2) SAN 환경의 리눅스 공유 디스크 파일 시스템에서의 회복 관리자 구성도

파일 시스템에 대한 회복을 담당하는 회복관리자는 각 연산을 수행하기 위한 트랜잭션 관리자, 변경된 메타 데이터에

대해서 저널 공간에 기록을 담당하는 로그 관리자, 그리고 시스템이 비정상적인 종료로 인하여 이상이 발생한 경우 회복을 하기 위한 회복 관리자로 구성된다.

3.2 트랜잭션 관리자

SAN 환경의 리눅스 공유 디스크 파일 시스템에 대한 회복 기법으로 저널링 방법을 사용한다. 저널링 방법을 지원하기 위하여 메타 데이터 변경 연산에 대해서 트랜잭션 처리를 하게 된다. 트랜잭션 관리자는 데이터 변경 연산에 대해서 원자성을 보장하여 메타 데이터가 일관성을 유지할 수 있도록 한다.

SAN환경의 리눅스 공유 파일 시스템에서는 각 파일에 대한 연산을 수행하기 위한 일련의 연산을 트랜잭션으로 규정하고, 이를 회복과 연산의 기본적인 단위로 사용한다.

(그림 3)은 트랜잭션을 처리하는 과정을 나타낸 것이다. 트랜잭션을 시작하여 새로운 트랜잭션 영역을 할당하고 초기화를 한다. 변경할 메타 데이터에 대해서 잠금을 획득하고 잠금 정보를 트랜잭션에 추가하여 트랜잭션이 관리를 하게 된다. 즉, 트랜잭션 관리자가 트랜잭션 처리 중 획득하는 잠금 정보에 대해서 유지를 하고 트랜잭션이 모두 끝나는 시점에서 잠금 정보를 해제 해 준다. 그리고, 메타 데이터 버퍼에 대해서 변경이 일어나는 동안 변경된 내용이 디스크에 반영되는 것을 못하게 하고(pin), 메타 데이터를 수정한다. 수정이 모두 끝나면 수정된 메타 데이터 정보를 로그 관리자에게 전달하고, 메타 데이터에 접근하기 위해 트랜잭션 처리 시 획득한 잠금 정보를 해제하여 다른 호스트에서 변경된 메타 데이터에 접근하는 것을 허용하게 된다. 이러한 과정을 마치면 트랜잭션이 종료하게 된다.

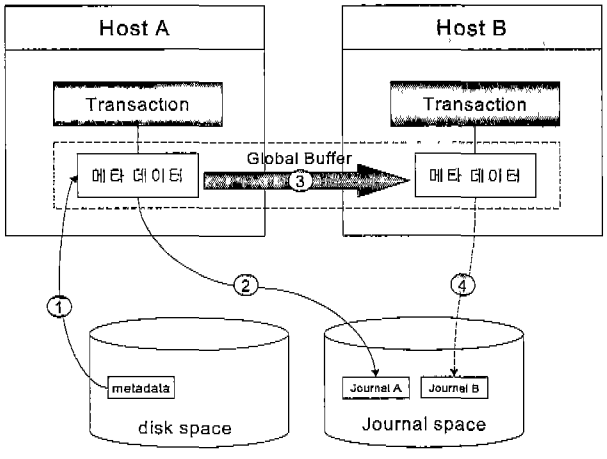
```
Transaction Manager
{
   transaction 시작 함수 호출 ;
   {
      transaction 영역 할당 ;
      transaction 영역 초기화 ;
      transaction 타입을 설정 ;
   }
   메타 데이터 버퍼를 transaction에 추가하는 함수 호출 ;
   {
      메타 데이터 버퍼에 잠금을 획득하고 transaction에 추가 ;
      메타 데이터 버퍼가 디스크 반영되지 못하도록 함(pin) ;
   }
   메타 데이터 수정 ;
   transaction 종료 함수 호출 ;
   {
      수정된 메타 데이터 정보를 로그 관리자에게 전달 ;
      메타 데이터에 대한 잠금 해제 ;
      transaction 종료 ;
   }
}
```

(그림 3) 트랜잭션 관리자 수행 과정

GFS 시스템에서는 어떤 호스트의 트랜잭션에 의해 변경된 메타 데이터에 대해서 다른 호스트가 동일한 메타 데이터 버퍼에 접근하려고 하면 해당 메타 데이터 버퍼를 디스크에 반영하고, 반영된 내용을 디스크에서 읽어 오게 된다. 이렇게 함으로써 저널링을 단순화할 수 있는 장점이 있지만, 디스크 입/출력이 발생하게 되어 시스템 성능의 저하를 가져온다.

이러한 단점을 보완하기 위해서 본 논문에서는 전역 버퍼를 이용한 메타 데이터 관리 기법을 제안한다. 여기서 전역 버퍼는 모든 호스트에 존재하는 지역 버퍼(local buffer)를 통합 관리한다.

(그림 4)는 전역 버퍼를 이용한 저널링 과정을 나타낸 것이다. 호스트 A가 메타 데이터를 디스크로부터 읽어 버퍼에 적재를 한다(①). 메타 데이터를 변경하고 호스트 A의 저널 공간에 로그를 기록한다(②). 호스트 B가 호스트 A가 변경한 메타 데이터에 대해서 접근하려고 하면 전역 버퍼 관리자에게 요청하고, 전역 버퍼 관리자는 호스트 A의 버퍼에 존재하기 때문에 해당 메타 데이터에 대한 잠금을 획득하고 네트워크를 통해 메타 데이터의 변경된 정보를 요청한 호스트 B에 전달한다(③). 이렇게 하면 디스크 입/출력 횟수는 3으로 앞에서 설명한 (그림 1)의 GFS 파일 시스템 보다 디스크 접근 수가 2 적다. 이렇게 하는 이유는 디스크 입/출력시간보다 네트워크를 통한 전송 시간이 빠르기 때문이다[12]. 특히 SAN 환경에서는 파이버 채널을 통하여 통신을 하기 때문에 시스템 성능을 더욱더 높일 수 있다.



(그림 4) SAN환경에서의 전역 버퍼를 이용한 저널링 과정

3.3 로그 관리자

로그 관리자는 트랜잭션 관리자로부터 수정된 메타 데이터 정보를 전달받아 저널 공간에 기록한다. 변경된 메타 데이터가 실제 디스크에 기록되기 이전에 항상 먼저 로그 정보를 저널 공간에 기록해야 한다(Write-Ahead Protocol[13]). 이렇게 기록함으로써 시스템 실패 시 저널 공간에 기록되어있는 로그 정보를 이용하여 이전의 상태로 되돌릴 수 있게 된다. 보통의 공유 파일 시스템에서는 하나의 저널에 다수의 서

버들이 공유하므로 저널을 관리하는 것이 복잡하고 비효율적이다. 이러한 단점을 극복하기 위하여, 본 논문에서 제안한 시스템에서는 각각의 서버가 자신만의 저널 공간을 가지고 있고, 또한 시스템 실패 시에는 다른 서버에 의해서 복구되기 위해서 모든 서버에게 자신의 저널 공간을 접근할 수 있도록 한다. 실패된 시스템에 대해서 해당하는 저널 공간을 찾아 기록된 로그 정보에 접근해서 로그를 재실행함으로써 실패 이전의 상태로 회복을 빠르게 할 수 있다.

기존의 GFS 파일 시스템에서의 메타 데이터에 대한 로깅 방법으로 비동기적 로깅을 사용하는데, 비동기적 로깅 방법은 완료된 트랜잭션에 대해서도 디스크에 로그가 반영되지 않은 상태에서 시스템에 장애가 발생하는 경우에는 회복을 보장해 주지 못하는 단점을 가지고 있다. 이런 단점을 보완하기 위해서 본 논문에서는 메타 데이터에 대해서는 동기적 로깅을 사용한다.

또한, GFS 파일 시스템에서는 메타 데이터에 대한 회복을 지원하지만, 중요 데이터에 대해서는 저널링을 하지 않기 때문에 중요 데이터에 대한 회복을 보장하지 못하는 단점을 가지고 있다. 이러한 단점을 보완하기 위해 본 논문에서는 중요 데이터에 대해서도 저널링 방법을 사용한다. 중요 데이터에 대한 회복을 지원하기 위해 데이터에 대해서도 메타 데이터처럼 버전 번호(version number)를 유지한다. 중요 데이터에 대한 로깅 방법은 비동기적 로깅 방법을 사용함으로써 입/출력 빈도를 줄여 준다. 동기적 로깅 방법은 완료된 트랜잭션에 대해서 회복을 보장하지만, 트랜잭션이 완료 될 때마다, 로그 정보를 디스크에 반영함으로서 디스크 입/출력이 자주 발생하는 문제점이 있다. 따라서 중요 데이터에 대해서도 동기적 로깅을 하면 시스템 성능이 현저히 감소하는 문제가 발생할 수 있다. 이런 문제를 줄이기 위해 중요 데이터에 대해서는 비동기적 로깅을 한다. 중요 데이터는 메타 데이터에 보다 회복의 중요도가 떨어지기 때문에 비동기적 로깅을 해도 큰 문제가 되지 않는다. 따라서 중요 데이터에 대해서는 비동기적 로깅을 함으로써 디스크 입/출력 빈도를 줄여 시스템 성능을 높일 수 있다. 이렇게 함으로써 제안하는 회복 기법은 메타 데이터뿐만 아니라 일반 데이터에 대해서도 회복을 보장해 주는 장점이 있다.

(그림 5)는 로그 관리자 수행과정을 나타낸 것이다. 트랜잭션 관리자로부터 변경된 메타 데이터 정보를 받아서 로그의 양을 계산하여 해당 로그 영역을 할당한다. 트랜잭션에 연결되어 있는 로그 버퍼에 대해서 버전 번호를 하나 증가시킨다. 버전 번호는 시스템 실패 시 로그와 디스크의 버전을 비교하여 로그의 재실행 여부를 판단하는데 사용한다. 변경된 로그 정보가 메타 데이터인 경우에는 동기적 로깅을 수행하고 일반 중요 데이터 인 경우에는 비동기적 로깅을 수행한다. 로깅을 마치면 트랜잭션의 로그 정보를 AIL(Active Item List)에 추가한다. 그리고 트랜잭션에 연결되어 있는 버퍼들에 대한 디스크 반영 금지를 해제(unpin)하여 실제 메타 데이터가 디스크에 반영하는 것을 허락한다. 이렇게 함으로서 실제 변경된 메타 데이터가 디스크에 반영되기 이전에 로그 정보를 먼저 저널 공간에 기록하게 되어 트랜잭션이 완료된 연산에 대해서 회복을 보장하게 된다. 후에 메타 데이터가 디스크에 반영되면 AIL에서 제거한다. AIL은 트랜잭션이 완료되고 실제 디스크에 반영되지 않은 버퍼들를 관리하는 데 목적이 있다.

```
Log Manager
{
    transaction 관리자로부터 얻은 변경된 log 정보를 모음 ;
    transaction에 연결된 log 영역을 할당 ;
    버전 번호(version number) 증가 ;
    if(변경된 log 정보가 메타 데이터)
    {
        동기적 로깅 ;
    }
    else
    {
        비동기적 로깅 ;
    }
    transaction에 정보를 AIL(Active Item List)에 추가 ;
    transaction에 추가된 버퍼들의 디스크 반영 금지 해제(unpin) ;
    변경된 버퍼가 디스크에 반영된 후에 AIL에서 제거 ;
}
```

(그림 5) 로그 관리자 수행 과정

3.4 회복 관리자

SAN환경의 리눅스 공유 파일 시스템에서 시스템에 장애가 발생하였을 경우 메타 데이터의 일관성을 유지시키기 위해 회복 연산이 수행된다. 호출된 회복관리자는 저널 공간에 기록된 로그 정보를 이용하여 해당 트랜잭션이 변경한 메타 데이터에 대한 내용을 트랜잭션 이전의 상태로 회복한다.

비정상적인 종료에 의해서 파일 시스템의 일관성이 깨지는 것을 회복하기 위해서 실패된 호스트 ID를 가지고 회복관리자가 호출 되게 된다. 이러한 회복 관리자가 호출되어 회복되는 과정을 나타낸 것이 (그림 6)이다. 처음으로 실패된 호스트에 해당하는 저널 공간을 찾아 로그의 헤드 정보를 읽어 시스템이 정상적으로 종료되었는지를 확인한다. 정상적으로 종료되었으면 회복 함수를 종료하고 비정상적으로 종료되었으면 로그의 처음과 끝의 엔트리를 찾게 된다. 로그를 처음부터 끝까지 읽어가면서 완료된 엔트리들 중에 부분적으로 완료된 엔트리들은 무시하게 된다. 회복을 하게 될 각각의 저널 엔트리에 대해서 엔트리와 관련된 잠금을 획득함으로써 회복하는 동안 다른 호스트가 변경하는 것을 막는다. 저널 공간에서 읽어온 로그 정보의 버전과 디스크에 기록된 버전과 비교하여 재실행 여부를 결정한다. 디스크 공간에 반영된 생성 버전보다 로그에 기록된 생성 버전이 크면 로그 정보를 디스크에 반영한다. 즉, 로그의 정보를 재실행함으로써 빠르게 회복 과정을 마치게 된다. 다음으로 저널 공간을 클리어하고 회복을 종료하게 된다. 회복이 일어나는 동안에도 실패된 호스트

에 획득한 잠금 정보를 필요로 하지 않는다면 다른 호스트는 정상적인 동작을 계속할 수 있다.

```
Recovery Manager
{
  log 헤드 정보를 읽음;
  if(시스템이 정상적으로 종료){ 회복 관리자 수행 종료;}
  else{
     log에 있는 저널의 처음과 끝의 엔트리를 찾음;
     부분적으로 완료된(commit) 엔트리들은 무시;
     for(start = 저널 엔트리의 처음; start != 엔트리의 끝; start = 다음
     엔트리)
     {
        저널 엔트리와 관련된 잠금 획득;
        디스크로부터 메타 데이터 읽음;
        저널 공간에서 log 정보를 읽음;
        if(디스크의 메타 데이터의 version number < log의 version
        number)
        {log에 기록된 메타 데이터를 디스크에 기록;
        } else {건너뜀(skip);}
     }
  }
  해당 저널 공간을 클리어 함;
}
```

(그림 6) 회복 관리자 수행 과정

3.5 SAN 환경에서의 회복 시나리오 예제

(그림 7)은 본 논문에서 제시한 SAN환경에서의 메타 데이터 변경이 일어날 때 처리되는 과정과 그 처리 과정에서 시스템에 문제가 발생하는 경우 회복하는 과정을 예제를 통해 파일 시스템의 일관성이 어떻게 유지되는지를 살펴보기 위한 시나리오를 나타낸 것이다.

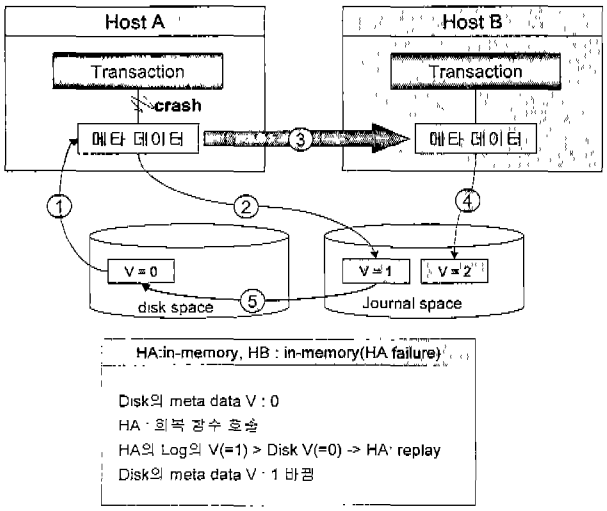
이 예제의 환경을 간략하게 설명한다.

(1) (그림 7)과 같이 호스트 2개가 네트워크로 연결되어 있고 디스크는 메타 데이터를 위한 부분과 저널을 기록하기 위한 저널영역으로 나누어 생각할 수 있다.

(2) 각각의 호스트는 로그를 기록하기 위한 자신만의 저널 공간을 가지고 있다.

시스템 실패 시 이 저널공간이 여러 호스트에 보여지게 된다.

(그림 7)에서와 같이 호스트 A가 메타 데이터를 접근하기 위해, 먼저 메모리에 있는 메타 데이터 버퍼를 접근하게 된다. 찾고자 하는 메타 데이터 버퍼가 메모리에 존재하지 않는다면, 호스트 A는 디스크에 직접 접근해서 메타 데이터에 대한 정보를 읽어 메모리에 적재하게 된다. 이때 디스크에 기록된 버전이 0이었다고 가정한다. 메타 데이터 버퍼를 변경하기 위해서 트랜잭션을 처리하고, 트랜잭션을 종료하게 되면 버전을 0에서 1로 증가하고 메타 데이터에 대한 로그 정보를 디스크에 반영을 하게 된다. 호스트 B가 호스트 A에서 사용하고 있는 메타 데이터에 대해서 접근을 할 때, 호스트 A는 트랜잭션을 종료하면 잠금을 해제하고 그 권한을 호스트 B에게 넘겨준다. 호스트 B에서는 메타 정보를 호스트 A로 부터 전송받아서 변경하기 위해서 트랜잭션을 처리하고 버전을 하나 증가하게 된다. 이 예제에서는 호스트 B에서 변경된 로그 정보를 호스트 B 저널 공간에 반영하고 호스트 A와 B의 메타 데이터 버퍼는 버퍼 정책에 의하여 실제 디스크에 반영되기 전에, 호스트 A가 실패했을 경우이다.



(그림 7) SAN 환경에서의 회복 시나리오

먼저 실패 된 호스트 ID를 가지고 호스트 A에 대한 회복이 호출 되게 된다. 먼저, 호스트 A의 저널 공간에서 로그 정보와 실제 디스크에서 메타 데이터를 읽어 디스크의 버전과 로그의 버전을 비교한다. 여기서 로그의 버전은 1 이고 디스크의 버전이 0이므로 로그를 재실행하면 된다. 따라서 디스크의 버전은 1로 바뀌고 회복을 마치게 된다.

4. SAN 환경의 리눅스 공유 디스크 파일 시스템의 회복 기법 구현

본 절에서는 본 논문에서 제안한 SAN 환경의 리눅스 공유 디스크 파일 시스템 회복 기법 구현에 대한 설명을 한다. 회복 기법 구현은 한국전자통신연구원에서 개발 중인 SANtopia 파일 시스템의 일부분으로 구현되었다. SANtopia 파일 시스템의 구현 환경과 회복과 관련된 트랜잭션, 로그, 회복 관리자의 구현 상황에 대해서 설명한다.

4.1 SANtopia 회복 관리자의 구현 환경

SANtopia 파일 시스템의 구현 환경은 <표 1>과 같다. 커널 2.2.18 버전이고 파일시스템의 블록 크기는 4k 바이트이다. 저널 영역의 크기는 128MB의 크기를 기본으로 하고 파일 시스템 생성할 때 변경할 수가 있다. SANtopia 파일 시스템의 구성된 호스트의 개수만큼 각각의 저널 공간을 유지한다. 저널 영역은 세그먼트로 나누어서 유지하는데 기본적인 세그먼트 개수는 16블록으로 하였다.

〈표 1〉 SANtopia 파일 시스템의 구현 환경

Linux kernel version : 2.2.18
The default size of block : 4096 byte - 4 kbyte
The default size of journal : 128 MB
The default size of journal segment : 16 blocks
The default number of journal : 1

4.2 트랜잭션 관리자 구현

SANtopia 파일 시스템의 트랜잭션 관리자의 자료 구조는 <표 2>와 같다. 트랜잭션의 정보를 AIL에서 관리하기 위해 리스트 구조로 유지하고, 로그 버퍼와 잠금 정보를 유지하기 위한 구조를 가지고 있다.

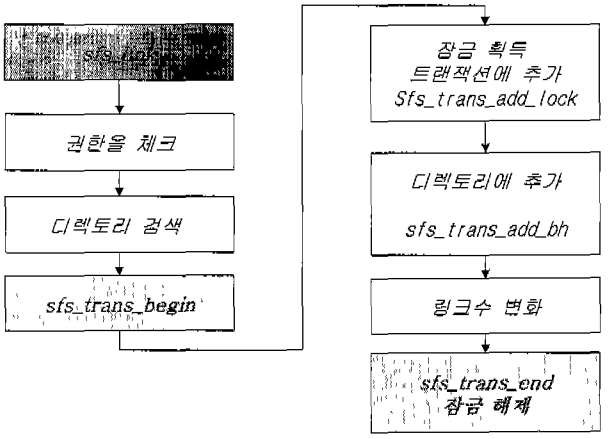
〈표 2〉 SANtopia 시스템의 트랜잭션 자료 구조

```
struct sfs_trans
{
   struct list_head tr_list ;          /* List of transactions */
   unsigned int     tr_type ;          /* The type of the transaction */
   unsigned int     tr_seg_reserved ;/* Number of segments reserved */
   unsigned int     tr_flags ;         /* state of transaction */
   unsigned int     tr_num_gl ;        /* Number of glocks in the trans */
   unsigned int     tr_num_buf ;       /* Number of buffers in the trans */
   struct list_head tr_gl ;            /* List of glocks in this trans */
   uint64 tr_log_head ;                /* The current log head */
   uint64 tr_first_head ;              /* First header block */
   struct list_head tr_bhlist ;        /* List of buffers in this transaction */
   struct list_head tr_logbhlist ;     /* List of buffers going to the log */
} ;
```

트랜잭션 관리자와 관련된 중요 모듈을 살펴보면 다음과 같다.

```
sfs_begin_transaction(sfs_sbi_t *sbip, int type)
{
    트랜잭션 영역 할당 ;
    트랜잭션 영역 초기화 ;
    트랜잭션을 리눅스의 현재 프로세스 영역에 셋업 ;
    트랜잭션 타입을 할당 ;
}
sfs_end_transaction(sfs_sbi_t *sbip)
{
    변경된 메타 데이터를 로그 관리자에 전달하여 저널 공간에 기록 ;
    트랜잭션 처리를 위한 잠금을 해제 ;
}
sfs_trans_add_bh(sfs_sbi_t *sdp, sfs_buf_t *bh)
{
    변경할 데이터 버퍼 헤드를 트랜잭션에 연결 ;
    트랜잭션의 연결된 버퍼의 수를 증가 ;
    버퍼를 트랜잭션 처리 중 디스크에 반영 금지(pin) ;
}
sfs_trans_add_lock( sfs_glock_t *gl )
{
    현재 프로세서에서 트랜잭션 정보를 얻음 ;
    잠금 정보를 트랜잭션의 잠금 리스트에 추가 ;
    트랜잭션의 잠금 개수를 증가 ;
}
```

(그림 8)은 SANtopia 파일 시스템에서 구현한 파일 시스템의 시스템 콜 함수인 심볼릭 링크(symbolic link) 연산에 대한 전체적인 수행 과정을 나타낸 것이다. 이 연산은 메타 데이터를 변경하기 때문에 비정상적인 종료에 대비하여 회복을 지원하기 위해 트랜잭션으로 처리를 해야 한다. 이 과정을 자세히 살펴보면, 처음에 링크를 처리하기 위해서 접근 권한을 확인하고 기존의 디렉토리 엔트리가 있는지를 확인한다. 기존의 디렉토리 엔트리를 찾고 트랜잭션을 처리하기 위해서 트랜잭션 시작(sfs_trans_begin) 함수를 호출하여 새로운 트랜잭션을 생성한다. 변경할 메타 데이터에 접근하기 위해 잠금을 획득하고 잠금 정보를 트랜잭션에 추가하고 디렉토리 엔트리를 하나 생성하여 추가하게 된다. 변경된 디렉토리 엔트리 버퍼에 대해서 트랜잭션에 추가하여 트랜잭션 관리자가 관리를 하게 된다. 다음으로 링크 수를 변경하고 나서 트랜잭션 종료(sfs_trans_end) 함수를 호출해서 로그 관리자에 변경된 메타 데이터 정보를 전달하여 저널 공간에 반영하고 잠금을 해제하고 트랜잭션 종료하면 링크 연산을 마치게 된다.



(그림 8) SANtopia 시스템의 심볼릭 링크(symbolic link) 수행 과정

4.3 로그 관리자 구현

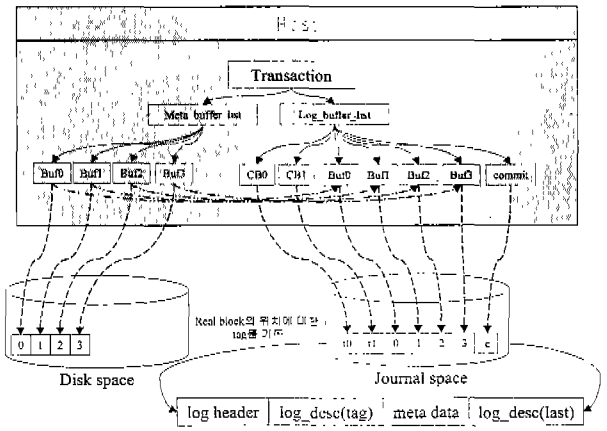
SANtopia 파일 시스템의 저널 공간은 구성된 호스트의 개수만큼 유지를 하고 각각의 호스트는 자기만의 저널공간을 관리한다. SANtopia 파일 시스템의 저널 구조는 (그림 9)과 같이 로그 헤더, 로그 디스크립터, 변경된 메타 데이터, 마지막 로그 디스크립터으로 구성이 된다. 각각의 자료 구조는 <표 3>과 같다.

SANtopia 파일 시스템에서 구현한 저널링 기법은 (그림 9)과 같다. 변경하려는 메타 데이터가 메모리에 적재되어 있지 않은 경우에는 메타 데이터가 저장되어 있는 디스크에 접근해서 읽어 오게 된다. 읽어온 메타 데이터를 현재의 트랜잭션에 추가시킨다. 메타 데이터 변경이 끝나면 로그 관리자는 디스크 상의 실제 메타 데이터 블록 위치 정보를 기록하기 위한 컨트롤 로그 버퍼를 할당한다. 또한, 실제 메타 데이터에 대한 정보를 저널 공간에 기록하기 위한 메타 데이터 로그 버퍼를 생성하고, 트랜잭션 완료 시 로그 버퍼를 저널 공간에

기록하게 된다. 즉, 로그 버퍼는 컨트롤 버퍼와 데이터 버퍼로 나눌 수 있다. 컨트롤 버퍼는 로그의 헤더 정보와 실제 디스크 상의 메타 데이터 위치 정보들이 저장되게 된다. 데이터 버퍼는 실제 메타 데이터의 내용을 저널 공간에 기록하기 위한 버퍼를 의미한다. 이렇게 컨트롤 버퍼와 데이터 버퍼를 저널 공간에 기록하게 된다.

〈표 3〉 SANtopia 시스템의 메타, 로그 헤더 및 로그 디스크립터 자료 구조

```
struct sfs_meta_header
{
  uint32 mh_magic ;       /* Magic number */
  uint32 mh_type ;        /* SFS_METATYPE_XX */
  uint64 mh_generation ;  /* Generation number */
} ;
struct sfs_log_header
{
  sfs_meta_header_t lh_header ; /* meta header */
  uint32      lh_flags ;        /* Flags */
  uint64      lh_first ;        /* Block number of first header */
  uint64      lh_sequence ;     /* Sequence number of this transaction */
  uint64      lh_tail ;         /* Block number of log tail */
} ;
struct sfs_log_descriptor
{
  sfs_meta_header_t ld_header ; /* meta header */
  uint32      ld_type ;         /* Type of data in this log */
  uint32      ld_length ;       /* Number of buffers */
  uint32      ld_num_metada ; /* number of metadata blocks
                                 In the descriptor */
} ;
```



(그림 9) SANtopia 시스템의 저널 구조 및 저널링 과정

본 논문에서 구현한 로그 관리자와 관련된 중요 모듈을 살펴보면 다음과 같다.

```
sfs_log_commit(sfs_sbi_t *sdp, sfs_trans_t *tr)
{
   메모리에 있는 트랜잭션을 완료 ;
   if(메타 데이터)
   {
      완료된 트랜잭션을 동기적 로깅(sfs_log_flush()) ;
   }
   else
   {
      비동기적 로깅인 경우 여러 트랜잭션을 모음 ;
   }
}
sfs_log_flush(sfs_sbi_t *sdp)
{
   로그 버퍼의 개수를 계산하여 로그 영역을 할당 ;
   로그를 처리하기 위한 잠금을 획득 ;
   트랜잭션 회복을 하기 위한 버전 번호를 증가 ;
   로그 정보를 디스크에 반영(sfs_disk_commit()) ;
}
sfs_disk_commit(sfs_sbi_t *sdp, sfs_trans_t *tr)
{
   트랜잭션의 로그 헤드 정보 슈퍼 블록으로부터 얻음 ;
   로그 정보를 위한 버퍼 헤드 리스트를 작성 ;
   로그 정보를 디스크 반영 ;
   현재 트랜잭션을 완료 ;
   트랜잭션의 연결된 로그 정보를 AIL에 추가하여 관리 ;
   트랜잭션에 연결된 버퍼들에 대해서 디스크 금지 해제(unpin) ;
   트랜잭션의 로그 헤드 정보를 슈퍼노드의 로그 헤드에 할당 ;
   트랜잭션의 버퍼 헤드 정보를 해제 ;
}
```

4.4 회복 관리자 구현

SANtopia 시스템의 어떤 호스트에 장애가 발생하면 sfs_recover_expired_host를 호출하여 회복 관련 쓰레드를 깨우게 된다. 본 논문에서 구현한 회복 관리자와 관련된 중요 모듈을 살펴보면 다음과 같다.

```
sfs_recover_expired_host(sfs_sbi_t *sbip)
{
   실패된 호스트 ID를 얻음 ;
   실패된 저널 공간을 회복하기 위한 함수 호출(sfs_wake_up_
   recoverd_process()) ;
}
sfs_wake_up_recoverd_process(sbip -> sbi_recoverd_process)
{
   회복 비트를 1로 셋팅 ;
   회복 함수 호출(sfs_check_journals()) ;
}
sfs_check_journals(sfs_sbi_t *sdp)
{
   for (j = 0 ; j < num_journal ; j ++)
   {
      if(회복 비트(recovery_bit) == 1)
         해당하는 클라이언트 ID를 가지고 회복 함수를 호출(sfs_
         recover_journal()) ;
   }
}
sfs_recover_journal(sfs_sbi_t *sdp, unsigned int jid)
{
   저널 인덱스 정보를 읽음 ;
   실패된 클라이언트에 대해서 회복을 위해서 저널에 잠금을 획득 ;
   저널의 시작과 끝을 찾기 위해서 로그의 헤드 정보를 읽음 ;
   트랜잭션의 잠금 획득 ;
   메타 정보를 회복(sfs_search_logdesc()) ;
   트랜잭션의 잠금을 해제 ;
   저널을 클리어(sfs_clean_journal()) ;
   회복 비트를 클리어 ;
```

```
}
sfs_search_logdesc(sfs_sbi_t *sbip, sfs_jindex_t *jdesc, uint64
start, uint64 end)
{
   로그 헤드 정보를 읽음 ;
   로그 디스크립터 정보를 읽음 ;
   If (log != 트랜잭션의 끝)
   {
      메타 데이터를 재실행(sfs_replay_metadata()) ;
   }
}
sfs_replay_metadata(sfs_sbi_t *sdp, sfs_jindex_t *jdesc, unit64
*cblk, void *data)
{
   재실행할 디스크립터 정보를 읽음 ;
   컨트롤 블록에 대한 포인터를 설정 ;
   메타 데이터 블록에 대한 포인터를 설정 ;
   for(메타 데이터 개수만큼)
   {
      메타 데이터 위치에 대한 태그 정보를 읽음 ;
      태그 정보를 넘겨주고 메타 데이터 블록을 재실행(sfs_do_
      replay( )) ;
   }
}
sfs_do_replay(sfs_sbi_t *sdp, sfs_jindex_t *jdesc, sfs_block_tag_t
*tag, unit64 *blkno)
{
   태그 정보를 통해 디스크에 있는 메타 데이터 블록 정보를 읽음 ;
   로그에 기록된 메타 데이터 정보를 읽음 ;
   if (디스크의 version number < log의 version number)
   {
      log에 기록된 메타 데이터를 디스크에 기록 ;
   }
   else
   {
      건너뜀(skip) ;
   }
}
```

5. 결론 및 향후 연구 방향

본 논문에서는 SAN 환경에서의 리눅스 공유 디스크 파일 시스템에서 문제가 발생하여 비정상적으로 시스템이 종료하였을 때, 회복 기법으로 저널링 방법을 설명하였다. 기존의 GFS 파일 시스템에서 사용하는 저널링 방법은 트랜잭션 처리하는데 있어서 변경된 동일한 메타 데이터 버퍼를 다른 호스트가 접근하려 할 때 메타 데이터 버퍼를 디스크에 반영하고 다른 호스트가 그 디스크로부터 입/출력하는 방법을 사용함에 따라 디스크 입/출력이 자주 발생하여 시스템 성능을 저하시키는 문제점을 가지고 있다. 이러한 단점을 해결하기 위해 디스크 입/출력 속도 보다 SAN환경의 파이버 채널을 통한 네트워크 전송 속도가 빠르다는 것을 전제로 하여 본 논문에서 제시한 시스템에서는 전역 버퍼를 이용하여 전역 버퍼에 존재하는 메타 데이터에 대해서는 메타 데이터에 대한 변경을 디스크에 기록하지 않고 네트워크를 통해 직접 전송을 하여 디스크 입/출력 횟수를 줄일 수 있는 기법을 제안하였다.

GFS 시스템에서 사용하는 비동기적 로깅 기법은 시스템 성능을 향상시킬 수 있지만 실제로 트랜잭션이 완료가 되었어도 로그가 디스크 반영되지 못한 상태에서 시스템이 장애가 발생하면 그 트랜잭션에 대한 연산은 무시가 되는 단점을 가지고 있다. 이러한 단점을 극복하기 위하여 메타 데이터에 대한 동기적 로깅을 사용하였다. 또한, 본 논문에서는 메타 데이터뿐만 아니라 중요 데이터에 대해서도 저널링 기법을 사용함으로써 중요 데이터에 대해서도 회복을 지원한다. 그런데, 데이터에 대한 로깅을 메타 데이터처럼 동기적 로깅을 하면 시스템 성능이 떨어진다. 중요 데이터는 비동기적 로깅을 사용하여 디스크 입/출력 빈도를 줄이는 방법을 제안했다.

본 논문에서 제시한 저널링 방법은 메타 데이터 버퍼 자체를 로그에 기록하기 때문에 파일의 크기가 커짐에 따라 로그의 크기도 증가하는 문제점을 가지고 있다. 따라서 로그의 양을 줄일 수 있는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] Clit Jurgens, "Fibre Channel : A Connection to the Future," IEEE Computer, Vol.28, No.8, pp.82-90, August, 1995.

[2] Kenneth W. Preslan, et al. "A 64-bit, Shared Disk File System for Linux," Proceedings of the 16th IEEE Mass Storage Systems Symposium, pp.22-41, San Diego, California, March, 1999.

[3] Kenneth W. Preslan, et al. "Implementing Journaling in a Linux Shared Disk File Systems," Proceedings of the 17th IEEE Mass Storage Systems Symposium, pp.351-378, College Park, Maryland, March, 2000.

[4] http : //www.veritas.com.

[5] http : //www.tivoli.com/products/index/sanergy_file_sharing/.

[6] 이용규, 김신우, 손덕주, "SAN 환경 공유 디스크 파일 시스템의 메타데이터 관리", 정보과학회지 pp.33-42, 2001.

[7] 김경배, 김영호, 김창수, 신범주, "SAN을 위한 전역 파일 공유 시스템의 개발", 정보과학회지 pp.24-32, 2001.

[8] http : //www-124.ibm.com/developerworks/oss/jfs/.

[9] http : //oss.sgi.com/projects/xfs/.

[10] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee, "Frangipani : A Scalable Distributed File System," Proceedings of the 16th ACM Symposium on Operating Systems Principles, pp.224-237, St. Malo, France, October. 1997.

[11] Stephen C. Tweedie, "Journaling the Linux ext2fs Filesystem," Proceeding of the LinuxExpo'98.

[12] C, Mohan Inderpal Narang, "Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," Proceedings of the 17th International Conference on Very Large Data Bases, pp.193-207, Barcelona, September, 1991.

[13] Philip A. Bernstein and Eric Newcomer, Principles of Transaction Processing, morgan Kaufmann Publishers, 1997.

박 춘 서

e-mail : parkcs@etri.re.kr
1999년 충북대학교 전기전자공학부(학사)
2001년 충북대학교 정보통신공학과(석사)
2001년~현재 한국전자통신연구원(컴퓨터시스템연구부 연구원)
관심분야 : 자료 저장 시스템, SAN, 병렬처리 시스템, 고차원 색인구조 등

김 경 배

e-mail : gbkim@etri.re.kr
1992년 인하대학교 전자계산학과(학사)
1994년 인하대학교 전자계산학과(석사)
2000년 인하대학교 전자계산학과(박사)
2000년~현재 한국전자통신연구원(컴퓨터시스템연구부 선임연구원)
관심분야 : 자료저장시스템, SAN, NAS, 이동컴퓨팅, 실시간데이터베이스시스템 등

이 용 주

e-mail : yongju@etri.re.kr
1999년 전북대학교 컴퓨터공학과(학사)
2001년 전북대학교 컴퓨터공학과(석사)
2001년~현재 한국전자통신연구원(컴퓨터시스템연구부 연구원)
관심분야 : 하부저장 시스템, 네트워크 스토리지, 분산파일시스템, 멀티미디어 데이터베이스 등

박 선 영

e-mail : sypark@etri.re.kr
1999년 충남대학교 컴퓨터공학과(학사)
2001년 한국과학기술원 전산학과(석사)
2001년~현재 한국전자통신연구원(컴퓨터시스템연구부 연구원)
관심분야 : 클러스터 시스템, 전역 메모리, 자료저장시스템 등

신 범 주

e-mail : bjshin@etri.re.kr
1983년 경북대학교 전자공학과(학사)
1991년 경북대학교 컴퓨터공학과(석사)
1998년 경북대학교 컴퓨터공학과(박사)]
1987년~현재 한국전자통신연구원(책임연구원, 시스템소프트웨어연구팀장)
관심분야 : 분산시스템, 고장감내 미들웨어, 이동컴퓨팅, 스토리지 클러스터 S/W 등