공유 디스크 파일 시스템을 위한 메타데이터 저널링 구조 설계

김신우*, 이용규*, 김경배**, 신범주**
*동국대학교 컴퓨터공학과, **한국전자통신연구원 인터넷서비스연구부

Designing Metadata Journaling Structure for a Shared Disk File System

Shin Woo Kim*, Yong Kyu Lee*, Gyoung Bae Kim**, Bum Joo Shin**
*Dept. of Computer Engineering, Dongguk University
**Internet Service Department, ETRI

요 약

파일 시스템에서 문제가 발생했을 때, 지속적인 서비스를 제공하기 위해서 보다 빠른 회복이 요구된다. 기존의 파일 시스템은 fsck를 이용하여 시스템을 회복하는 데 많은 시간이 필요하고, 회복 중에 오프라인 상태를 요구하기 때문에 서비스가 중단되는 단점이 있다. 따라서, GFS와 같은 공유 디스크 파일 시스템에서는 저널링을 이용하여 온라인 상태에서의 회복을 가능하게 하고 회복 시간을 단축시키는 효과를 거두었다. 그러나, 한 클라이언트가 디스크의 메타데이터를 수정하는 중에 다른 클라이언트가 동일한 블록을 사용하고자 할 때, 앞의 클라이언트가 메타데이터를 디스크 저널에 기록한 후 다시 디스크에 기록하기까지 기다린 후에 디스크에 접근하여 사용할 수 있다. 이처럼 동일한 블록을 사용하더라도 불필요한 디스크 접근이 발생한다. 본 논문에서는 이러한 문제점을 해결하기 위해서 클라이언트에서 클라이언트로 메타데이터를 직접 넘겨줄 수 있도록 개선한다. 성능 분석 결과 이러한 개선 방안이 기존의 저널링보다 디스크 접근 횟수를 줄임으로써 트랜잭션 처리 시간을 줄이는 결과를 얻을 수 있다.

1. 서론

대부분의 파일 시스템들은 메타데이터를 관리하는 데에 캐쉬를 이용한다. 캐쉬는 디스크에 접근하는 시간을 줄인다는 장점이 있는 반면, 시스템에 문제가 발생할 때 디스크와 캐쉬에 있는 메타데이터를 동시에 수정하지 못하므로 파일 시스템의 일관성이 없어지게 되는 단점을 가지고 있다.

이처럼 시스템에 문제가 발생될 때는 fsck(파일 시스템 체크 루틴)을 이용하여 비일관성의 원인을 체크하고 해결하는데, 이 fsck의 수행은 파일 시스템의 크기에 비례하여 많은 시간이 필요하고, 시스템의 오프라인 상태를 요구한다. 그러므로 fsck는 여러 클라이언트들이 디스크를 공유하는 형태의 공유 디스크 파일 시스템에는 적합하지가 않다. 따라서 공유 파일 디스크 시스템에서는 회복시간을 줄이고 유용성을 증가시키기 위해서 데이터베이스의 트랜잭션 개념을 이용하는 저널링(journaling)을 사용하며, 이와 같은 시스템으로는 GFS[1][2][3], Frangipani[4], 그리고 Linux ext2fs[5]를 들 수 있다.

본 연구는 한국전자통신연구원의 2000년도 위탁과제 "SAN 환경에서의 Global File Metadata 관리 기법 연구"의 일부로 수행됨.

GFS(Global File System)에서는 각각의 클라이언트가 자신만의 저널 공간을 가지고 있고 회복 시에는 모든 클라이언트들이 자신의 저널의 접근을 가능하게 한다. GFS에서는 트랜잭션 관리자와 저널 관리자를 이용하여 저널링을 구현한다. 트랜잭션 관리자는 디스크에서 필요한 메타데이터를 사용하고자할 때, 트랜잭션을 생성하고 락을 걸어 다른 클라이언트들이 접근을 못하게 한 후, 메타데이터를 수정하는 역할을 한다. 저널 관리자는 트랜잭션 관리자로부터 수정된 메타데이터를 받아 디스크 저널에 기록하고 다시 디스크에 기록하는 역할을 담당한다. 이와 같이 저널링을 하는 도중에 동일한 블록을 두 클라이언트가 접근하려고 할 때, 한 클라이언트가 디스크에 접근하여 메타데이터를 수정하고 그 메타데이터를 저널에 기록한 후 다시 디스크에 기록한 다음에야 다른 클라이언트가 디스크에 접근할 수 있다.

Frangipani 역시 각각의 Frangipani 서버에 자신의 저널을 가지고 있다. Frangipani는 저널을 순환 버퍼로 관리하고, 저널마다 순차적인 번호를 붙인다. 저널이 꽉 채워지면 일부분은 다시 회수하는데, 회수지역의 메타데이터 블록이

디스크의 원래 위치에 기록되지 않았다면, 저널이 회수되기 전에 디스크에 기록한다. Frangipani 역시 GFS처럼 서버가 디스크의 메타데이터를 읽고 수정한 것을 저널에 기록하고 다시 디스크에 기록 한 후에 다른 서버가 그 메타데이터를 사용할 수 있다.

이처럼 저널링은 디스크에 수정된 메타데이터를 기록하기 전에 저널에 기록함으로써, 시스템 문제 발생 시 이 저널을 이용하여 빠른 회복을 수행할 수 있다. 그러나, 매번 메타데이터를 수정할 때마다 저널에 기록한 다음, 그것을 원래의 디스크에 기록하는 일을 반복해야 하고, 심지어 동일한 블록을 두 클라이언트가 접근할지라도 한 클라이언트가 디스크 저널에 기록한 후 디스크에 기록한 다음에야 다른 클라이언트가 디스크에 접근하여 사용할 수 있기 때문에 불필요한 디스크 접근이 발생한다. 이를 해결하기 위해서 클라이언트에서 클라이언트로 직접 메타데이터를 전송하여 디스크 접근 횟수를 줄이는 방안을 제시하고자 한다. 앞으로 본 논문에서 제안할 저널링을 이용할 파일 시스템을 MGFS (Modefied Global File System)[6]라고 하기로 한다. 이는 GFS를 기본으로 하여 개선된 파일 시스템을 의미한다.

2. MGFS에서의 저널링

MGFS에서의 저널의 구조를 살펴보고, 메모리와 디스크에서 사용하는 저널의 자료구조를 설계한다. 이 자료구조를 이용하여 클라이언트들 간의 메시지 전송을 통한 저널링의 성능개선을 모색한다.

2.1 레이아웃

클라이언트마다 각각의 독립적인 저널 공간을 가지고 있으며, 저널 공간은 락에 의해 보호된다. [그림 1]은 4개의 클라이언트를 가정한 레이아웃이다. 저널 공간은 클라이언트 실패시 다른 클라이언트들이 접근할 수 있어야 한다.

lock 0	Super Block
lock 1	Journal 0
lock 2	Journal 1
lock 3	Journal 2
lock 4	Journal 3
locks 5-1000	Resource Group 0
locks 1001-2000	Resource Group 1

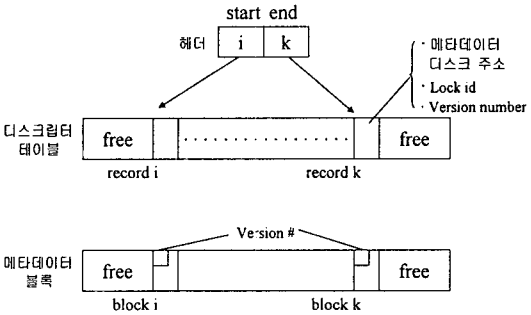
[그림 1] 저널의 레이아웃

2.2 자료구조

저널링을 구현하기 위해서는 디스크와 메모리에서 메타데이터를 관리할 각각의 자료구조가 다음과 같이 요구된다.

가. 디스크 저널 자료구조

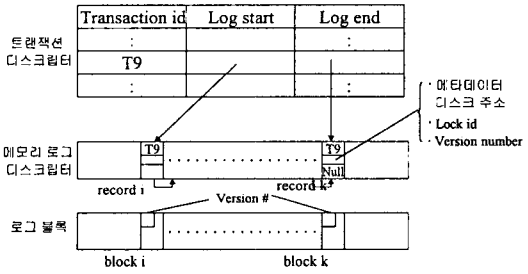
디스크에 존재하는 저널을 관리하기 위한 자료구조는 [그림 2]와 같이 저널 헤더, 디스크립터 테이블, 메타데이터 블록으로 구성되어 있다. 저널 헤더는 저널의 위치 정보를 저장하는 것으로 저널의 시작과 끝을 기록하고 있다. 디스크립터 테이블은 메타데이터의 정보를 기억하기 위해 메타데이터의 디스크 주소와 그 메타데이터를 사용한 클라이언트가 이용한 lock id, 그리고 version number를 기록한다. 또 메타데이터 블록은 실제의 데이터를 저장하고 있다.



[그림 2] 디스크 저널 자료구조

나. 메모리 저널 자료 구조

메모리에 있는 저널을 관리하기 위한 자료구조는 [그림 3]과 같이 트랜잭션 디스크립터, 메모리 저널 디스크립터, 저널 블록, 캐쉬 디스크립터로 구성되어 있다.



캐쉬 디스크립터

Cache page #	Disk page #	Dirty bit	Pin counter	Delay mark	Client id	Log page #	Lock id	Waiting Client Queue
21	P3	0	1	0	null	null	null	c3
22	P5	1	1	1	c0	1210	L5	c1 c2
23	P9	1	0	0	null	1320	null	null

[그림 3] 메모리 저널 자료구조

트랜잭션 디스크립터는 트랜잭션 정보를 가지는 것으로서 트랜잭션 id와 저널의 시작과 끝을 기록하고 있으며, 메모리 저널 디스크립터는 디스크의 디스크립터 테이블과 비슷한 역할을 하며 메타데이터의 디스크 주소와 lock id, 그리고 version number를 기억하고 있다. 저널 블록은 실제의 데이터를 저장하고, 캐쉬 디스크립터는 로깅을 위해서 필요한 모든 정보를 저장한다. 이중 delay mark란 자신이 사용한 메타데이터의 수정이 끝난 후에 자신이 끝나기만을 기다린 다른 클라이언트에게 메시지를 보내고 그로부터 다시 메시지를 받을 때까지 기다리는 시간을 측정하기 시작했다는 것을 표시하는 것을 말하며, Client id는 메타데이터를 디스크

가 아닌 다른 클라이언트로부터 받았을 경우 메타데이터를 준 클라이언트를 맡하고, Waiting Client Queue는 현재 이 메타데이터를 요구하고 사용이 끝나기를 기다리고있는 클라이언트들을 표시한다.

2.3 저널링

MGFS에서는 트랜잭션 관리자와 저널 관리자를 통하여 저널링을 한다. 이때 동시에 여러 클라이언트가 동일한 블록을 수정하고자 할 때, 먼저 요구한 클라이언트가 디스크에서 메타데이터를 사용하고 나머지 클라이언트는 그 클라이언트의 사용이 끝나기를 기다려서 메시지와 수정된 메타데이터를 그로부터 직접 받아 사용하는 것을 가능하게 한다.

가. 트랜잭션 관리자

저널링은 트랜잭션을 사용하여 파일 시스템 상태를 변화하며, 각각의 저널링 엔트리는 관련 있는 하나 이상의 락(lock)들을 가지고 있다.

트랜잭션 생성 단계는 다음과 같다.

1) 트랜잭션 시작
2) 필요한 락을 획득.
3) 메모리의 메타데이터 버퍼들을 디스크에 기록되지 않도록 함(pin)
4) 메타데이터 수정
5) 저널 관리자에 트랜잭션을 전달

나. 저널 관리자

저널 관리자는 메모리의 수정된 메타데이터를 디스크 저널에 기록한다.

저널 관리자는 다음 단계를 따른다.

1) 트랜잭션 관리자로부터 트랜잭션을 받음
2) 트랜잭션들을 모음 (비동기적 로깅)
3) 디스크 저널에 기록
4) 다른 클라이언트로부터 메타데이터를 받았다면, 로깅 메시지를 보내줌
5) 메타데이터를 기다리는 클라이언트가 있으면, 수정된 메타데이터와 락을 넘겨줌. 없으면, 8)을 수행
6) delay mark를 세팅하고, delay 시간 측정을 시작함.
7) 메타데이터를 받은 클라이언트로부터 로깅 메시지가 도착하면 메타데이터가 있는 메모리 버퍼를 해제하고 트랜잭션 종료. 일정시간(delay 시간)이 지나도 아무런 메세지가 없으면 8)을 수행
8) 메타데이터의 디스크 기록 금지 해제 (unpin)
9) 버퍼관리자에 의한 디스트 commit 수행

2.4 회복

회복 관리자는 실패한 클라이언트의 id와 저널 락을 획득하면서 회복을 시작한다. 저널 회복 단계는 다음과 같다.

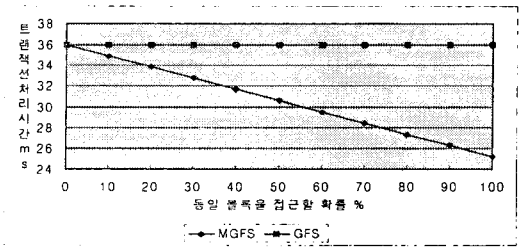
1) 저널의 처음과 끝의 엔트리를 찾음
2) 부분적으로 commit된 엔트리들은 무시
3) 각각의 저널 엔트리에 대해 회복 클라이언트들은 그 엔트리와 관련 있는 모든 락을 획득
4) 디스크에 있는 버전 번호와 회복하려는 클라이언트 저널의 버전 번호를 비교. 디스크의 버전이 저널의 직전 버전일 때는 회복을 수행. 그렇지 않으면 앞서 수행한 다른 클라이언트들이 디스크 commit를 모두 끝낼 때까지 해당 저널의 회복을 늦춤.

3. 성능평가

본 절에서는 앞에서 제안한 저널링이 기존의 시스템에 비해 성능을 얼마나 향상시킬 수 있는가를 분석하고자 한다. 여기서 트랜잭션 처리 시간은 메타데이터의 락을 획득한 후부터 락을 해제할 때까지로 하였으며, 네트워크의 데이터 전송률은 10mbps로 하였다.

3.1 평균 트랜잭션 처리 시간 성능 평가

[그림 4]는 두 개의 클라이언트가 동일 블록을 요구할 때, 각 클라이언트가 평균적으로 한 트랜잭션을 처리하는데 필요한 시간을 기존의 GFS와 비교한 것이다.



[그림 4] 평균 트랜잭션 처리 시간 비교

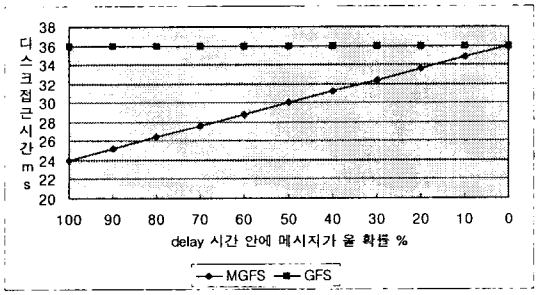
두 개의 클라이언트가 디스크의 동일한 블록을 접근할 때, GFS는 한 클라이언트가 디스크에 접근하여 메타데이터를 읽고 그 메타데이터를 수정한 후 저널에 기록한다. 그리고 그 내용을 디스크에 기록한 다음에야 다른 클라이언트가 디스크에 접근할 수 있다. 즉, 두 개의 클라이언트가 동일한 블록을 수정하려고 할 때, 각 클라이언트마다 읽고 저널에 기록하고 다시 디스크에 기록하는 3번의 디스크 접근을 요구하므로 총 6번의 디스크 접근이 필요하다.

위와 같은 경우에, MGFS는 첫 클라이언트에서 디스크에 접근하여 메타데이터를 읽고 수정한 메타데이터를 저널에 기록한 후, 수정된 내용을 디스크에 기록하는 대신에 다음 클라이언트에 직접 넘겨줌으로써 디스크 접근의 횟수를 각 클라이언트 당 1번씩 줄여서 총 4번의 디스크 접근으로 위와 같은 일을 가능하게 한다. [그림 4]에서 동일 블록을 접근할 확률이 많아짐에 따라 한 클라이언트당 평균 트랜잭션 처리시간이 점점 줄어들게 됨을 확인할 수 있다.

3.2 delay 시간에 따른 디스크 접근 시간 성능평가

[그림 5]는 두 개의 클라이언트가 동일블록을 접근할 때,

해당 delay 시간 안에 첫 클라이언트가 다음 클라이언트에게 메타데이터를 보내고, 메타데이터를 받은 클라이언트로부터 저널에 기록했다는 로깅 메시지가 도착할 확률에 따른 클라이언트들의 평균 트랜잭션 처리 시간을 기존의 GFS와 비교한 것이다.



[그림 5] delay 시간에 따른 디스크 접근 시간 비교

앞에서 설명한 바와 같이 동일한 메타데이터 블록을 요구하는 두 개의 클라이언트가 있을 때 앞의 클라이언트는 다음 클라이언트에게 메타데이터를 전달한 후 그로부터 저널 로깅이 잘 수행했는지에 대한 여부를 알려주는 메시지를 기다리는 delay 시간이 필요하다. 이 delay 시간 안에 메시지가 도착하면 다른 클라이언트의 저널 로깅이 잘 수행된 것이므로 앞의 클라이언트는 디스크 commit이 필요치 않으며, 그렇지 못하면 오류가 발생한 것으로 해석하여 디스크에 commit를 수행하여야 한다. 이처럼 delay 시간 안에 메시지가 도착하지 않으면 디스크 commit을 위한 디스크 접근을 생략할 수 없다.

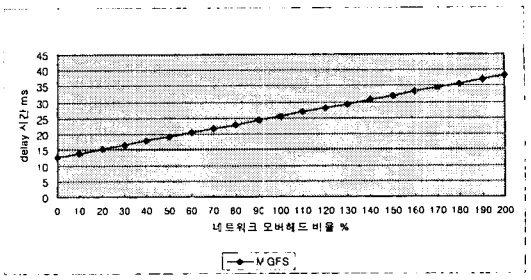
[그림 5]는 delay 시간 안에 메시지가 도착할 확률이 점차 감소됨에 따라 디스크 평균 접근 시간이 GFS와 같아지는 것을 보여준다.

3.3 네트워크 오버헤드에 따른 delay 시간 성능평가

[그림 6]은 네트워크 오버헤드에 따라 요구되는 delay 시간을 평가한 것이다.

처음 최소한으로 요구되는 delay 시간은 자신이 보낸 메시지와 메타데이터를 받은 클라이언트가 수정한 메타데이터를 저널에 기록하고 로깅 메시지를 보내올 시간이다. 이것을 기준으로 네트워크의 오버헤드의 확률에 따라 필요한 delay의 시간을 계산하였다.

[그림 6]에서 네트워크의 오버헤드의 비율이 높아짐에 따라 다음 클라이언트로부터 메시지가 도착하기를 기다려야 할 delay 시간이 길어짐을 볼 수 있다.



[그림 6] 네트워크 오버헤드에 따른 delay 시간

4. 결론

본 논문에서는 공유 디스크 파일 시스템의 일종인 GFS에서의 저널링을 개선하여 클라이언트간의 직접적인 통신을 통하여 클라이언트에서 클라이언트로 메시지를 이용하여 메타데이터를 직접 넘겨주는 것을 가능하게 함으로써 불필요한 디스크 접근을 줄이고자 하였다.

결론적으로 여러 클라이언트에서 동일 블록을 참조하고자 할 때, 한 클라이언트 당 평균 트랜잭션 처리 시간이 감소하게 되었고, 나아가 디스크 입출력 횟수를 줄임으로써 디스크 병목현상 발생도 줄어드는 장점이 있다.

[참고문헌]

[1] Kenneth W. Preslan, "Implementing Journaling in a Linux Shared Disk File System," the Eighth NASA Goddard Conf. On Mass Storage Systems and Technologies.

[2] Steven R. Soltis, Thomas M. Ruwart, Matthew T.O Keefe, "The Global File System," the Fifth NASA Goddard Space Flight Center Conf. On Mass Storage Systems and Technologies, Sept 17-19, 1996, College Park, MD.

[3] Steven R. Soltis, Thomas M. Ruwart, Matthew T.O Keefe, "A 64-bit, Shared Disk File System for Linux," the Sixteenth IEEE Mass Storage Systems Symposium, March 15-18, 1999, San Diego, California.

[4] Chandramohan A. Thekkath, Timothy Mann and Edward K. Lee, Frangipani: A Scalable Distributed File System", SOSP-16 10/97 Saint-Malo, France.

[5] Stephen C. Tweedie, "Journaling the Linux ext2fs File System," LinuxExpo'98.

[6] 김신우, 이용규, 김경배, 신범주, "대용량 파일 시스템을 위한 메타데이터 구조 설계," 한국정보과학회 '2000 추계 학술발표논문집, 숙명여자대학교, 2000. 10.