2002 년 한국정보처리학회 춘계학술발표논문집 제 9 권 제 1 호

SAN 환경의 공유 파일 시스템을 위한 분산 락킹

최성춘, 윤희용, 추현승
성균관대학교 정보통신 공학부
choisc, youn, choo@ece.skku.ac.kr

Distributed Locking for File Sharing in Storage Area Network.

Sungchune Choi, Hee Yong Youn, Hyunseung Choo
School of Information and Communication,
Sungkyunkwan University

요 약

SAN(Storage Area Network)은 현재 폭발적으로 증가하는 데이터의 관리를 위해 대용량의 공유 저장 장치를 사용하는 분산 환경의 고속 네트워크 저장 시스템이다. SAN 환경에서는 여러 호스트들에게 대용량 저장장치의 동시 접근을 허용함으로써 사용자에게 고 확장성과 신뢰성, 그리고 고 가용성을 제공해 준다. 그 결과 여러 호스트에 의해 공유되는 저장장치의 자료에 대한 비 일관성 문제가 발생하게 된다. 본 논문에서는 여러 호스트에 의해 동시 접근되는 공유 저장장치의 데이터 일관성을 유지하기 위한 락킹 모듈을 리눅스 운영체제 기반에서 구현하고, 그 성능을 평가한다.

1. 서론

SAN 은 폭발적으로 증가하는 데이터의 저장을 위해 서버와 대용량 저장 장치들을 FC(Fiber Channel)로 연결하여 고성능 저장장치 시스템을 구성하는 새로운 개념의 저장장치 솔루션이다. SAN 환경에서는 시스템에 연결된 저장장치의 데이터를 여러 사용자들이 공유하여 사용함으로써 동시에 같은 데이터를 접근하는 동시성 문제가 발생하게 된다. 이러한 동시성 문제를 해결하기 위한 가장 기본적인 방법은 데이터에 대한 락킹이다.

락킹이란, 분산 환경 파일 시스템에서 데이터의 일관성을 위해 동시에 여러 사용자가 같은 파일을 고치거나 수정할 수 없도록 사용자의 접근을 제어하는 기법을 말한다. 사용자의 파일 요청이 있을 때 락관리자(lock manager)는 요청된 파일이 다른 사용자에 의해 사용되고 있는지 확인하여, 사용되고 있지 않은 경우에 사용자에게 파일에 대한 권한을 주고 다른 사용자가 같은 파일에 접근하여 수정하는 것을 방지하게 된다. 만일 이와 같은 적당한 동시성 제어가 없다면, 같은 데이터를 한명 이상의 사용자가 동시에 수정하려고 시도할 때, 데이터에 접근한 결과가 정확하지 않은 것으로 판명되어 저장 시스템 전체에 영향을 미칠 수도 있다. 따라서 본 논문에서는 다수의 사용자에 의해 저장장치 데이터를 공유하는 SAN 환경에서 일관성을 유지하기 위해 필수적으로 필요한 락킹 모듈을 리눅스 운영체제 기반에서 구현하고, 그 성능을 평가한다.

본 논문의 구성은 다음과 같다. 2 장에서는 동시성 제어에 관한 기존 연구에 대하여 알아보고, 3 장에서 락킹 모듈을 위한 동작 환경, 구현 및 성능을 평가하고, 4 장에서 결론 및 향후 연구과제를 제시한다.

2. 관련 연구

락킹 방법은 기본적으로 중앙 집중식과 분산 방식 두 가지로 나뉘어 진다.[1]

중앙 집중식 락킹 방법은 하나의 락 서버가 모든 락에 대한 제어를 관리하는 방식이다. 호스트들은 락 서버에게 락을 요청하고, 만약 요청한 락이 사용가능하면 락 서버는 해당 락에 대한 호스트의 요청을 수락하게 된다. 이 방식의 단점은 락 서버에 모든 부하가 집중되어 응답시간이 증가하게 되고, 락 요구 시 락 서버로 적어도 한번의 통신이 필요하게 되므로 락 서버간의 통신 비용이 증가하게 되고, 지연시간이 길어지게 된다.

분산 락킹 방식은 중앙 집중식 방식의 단점인 응답시간을 줄이기 위해 여러 개의 락 서버에 락을 분할

하여 관리하는 방식이다. 그러나 락을 분할 관리함으로 인해 교착 상태에 쉽게 빠질 수 있다는 단점을 가지고 있다. 이러한 교착 상태를 해결하기 위해 보다 복잡한 교착상태 발견 방식이 필요하게 된다.[2]

현재의 분산 환경 파일 시스템을 위한 대표적인 락킹 방식으로는 GFS 의 Dlock[3]과 VAXCluster DLM[4]이 있다.

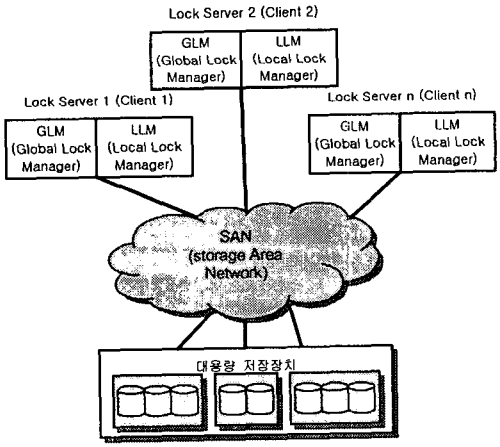
GFS 의 Dlock 은 저장장치 디바이스에서 락 서버를 내장하고, 디바이스가 락 서버의 역할과 저장장치의 역할까지 대신하게 된다. 디바이스 락에 대한 호환성이 없는 디바이스에 대해 IP(internet protocol)을 기본으로 한 락을 제공할 수 있는 모듈도 포함되어 있다.[3]

DLM 은 락과 세마포어(semaphore)를 사용하여 공유 데이터의 일치성을 유지하고, GFS 의 Dlock 과 유사하게 버전넘버(version number)를 통해 각 노드에 저장된 락에 대한 일치성을 유지 시킨다.[4]

3. 락킹 모듈의 동작 환경과 구현

3.1 동작환경

(그림 1)은 락킹 모듈이 동작하는 SAN 환경과 락 서버의 기본적인 구조를 보여주고 있다. 사용자의 데이터 접근을 담당하는 각각의 클라이언트에는 데이터에 대한 락을 담당하는 락 서버가 모듈로 동작하게 된다.



(그림 1) SAN 환경에서의 락킹 모듈 구성

각 락 서버는 각 클라이언트에 요청한 락을 관리하기 위한 LLM(local lock manager)과 전체 시스템의 락을 관리하는 GLM(global lock manager)로 구성된다. 모든 사용자는 데이터에 대한 사용 권한을 얻기 위해 락을 요청한 클라이언트에게 데이터에 대한 LLP(local lock pointer)를 얻고, 해당 데이터의 전역 락을 관리하는 락 서버에게 GLP(global lock pointer)를 얻어야 한다. 락 서버에 대한 락의 분배 방식은 (락 번호 mod 수행중인 락 서버의 수)의 연산에 의하여 이루어진다. 락 번호는 모든 시스템에서 데이터에 대한 유일한 식별자로 여기서의 락 번호는 inode+2 로 부여된다. 파일 시스템에서 inode 번호는 전체 시스템에서 유일하기 때문에 락 번호도 전체 시스템에서 유일하게 된다.

3.2 모듈 코드 구성

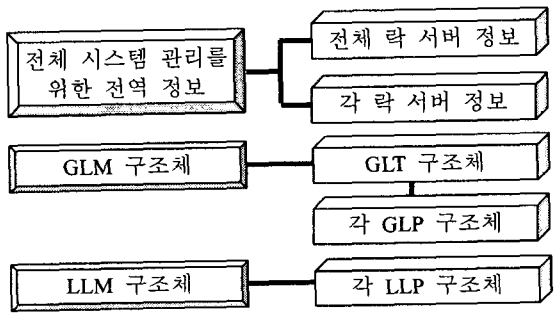
락킹 모듈의 구현은 실제 동작을 수행하는 3 개의 c 코드와 락 서버 구조체 및 함수 정의를 나타내는 10 개의 헤더 파일로 구성되며 각 코드의 이름과 역할은 (표 1)과 같다.

(표 1) 락 모듈 코드 구성

이름	역할
lm_init.c	락 서버 초기화 및 종료
lm_basicop.c	락을 구현하기 위한 기본적인 함수
lm_lock.c	락킹 메커니즘에 따른 락킹 동작 제공
lm_init.h	lm_init.c 의 함수 정의
lm_basicop.h	lm_basicop.c 의 함수 정의
lm_lock.h	lm_lock.c 의 함수 정의
lm_head.h	락 서버 구조체 정의
lm_comm.h	락 서버간에 통신을 위한 통신 모듈
init_api.h	시스템에 구성된 클라이언트 정보 정의

3.3 락 서버 초기화

SAN 환경에 클라이언트를 추가할 경우, 클라이언트는 락 모듈을 동작시키고 락 서버로서의 동작을 수행하게 된다. 락 서버의 초기화는 (그림 2)에서와 같이 LLM 과 GLM 을 초기화하는 부분과 LLM 과 GLM 에서 사용하게 되는 LLP(local lock pointer)와 GLT(global lock table)및 GLP(global lock pointer)를 초기화하는 과정으로 나뉘게 된다. 락 모듈은 파일 시스템을 장치에 마운트할 때 init_module 함수에 의해 동작하고, 언마운트 하는 과정에서 cleanup_module 함수에 의해 종료된다.



(그림 2) 락킹 모듈 구조체 구성

락 서버가 시스템에서 동작하게 되면 전체 락 서버 정보에 동작중인 락 서버의 수를 증가시키고, 해당 락 서버 정보를 리스트로 연결하여 관리하게 된다. 변화된 락 서버의 수는 요청된 락을 락 서버에 분배하기 위해 사용되고, 이 전역 정보는 락 서버가 추가되거나

제거 될 때마다 통신 모듈을 통한 락 서버간의 통신으로 전체 서버가 같은 정보를 유지하게 된다.

3.4 락의 객체 와 락 모드

락킹 프로토콜에서 락의 객체와 granularity 는 전체 락킹 모듈을 구성하는데 있어 중요한 문제이다. 일반적으로 파일 시스템에서 락을 걸 수 있는 단위는 inode, 블록, 레코드, 블록 범위, 바이트, 그리고 바이트 범위로 구분될 수 있다. 일반적으로 정제된 락 granularity 는 데이터에 대한 동시성을 높일 수 있지만, 반면에 정제되지 않은 락 granularity 는 이와 반대이다. 본 논문에서 구현된 락킹 모듈에서는 inode 와 메타데이터를 락의 객체 사용하므로, granularity 는 파일 단위가 된다.

일반적인 동시성 접근 제어 정책은 S(shared lock), X(exclusive lock), IS(intentionally shared lock) , IX (intentionally exclusive lock), 그리고 SIX(shared and intentionally exclusive lock)의 5 가지의 락 모드를 지원한다. 위의 락 모드는 파일 수준보다 더욱 정제된 락 모드 granularity 를 위해 필요하며, 파일 수준의 락 granularity 에서는 shared lock 과 exclusive lock 만으로 전체 락을 관리한다. (그림 3)은 구현된 락킹 모듈의 락 모드별 동작을 보여준다.

요청 \ 현재상태	Shared (S)	Exclusive (E)
Shared (S)	O	X
Exclusive (E)	X	X

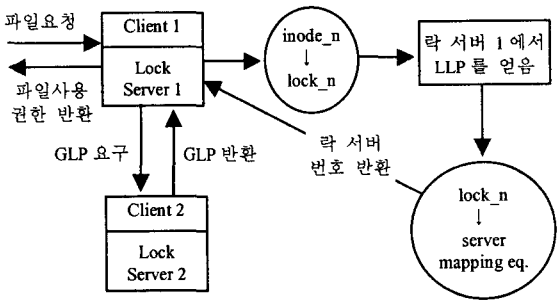
(그림 3) 구현된 락킹 모듈의 락 모드

(그림 3)에서 S 락은 사용자가 파일에 대하여 읽기를 요청할 때 락 서버가 제공하는 락이고, E 락은 파일을 생성하거나 수정할 때 제공되는 락 모드이다. (그림 3)에서 보는 것과 같이 만일 사용자가 어떤 파일을 읽기 위해 S 락을 가지고 있고, 다른 사용자가 같은 파일을 읽기 위해 S 락을 요청한다면, 같은 파일을 두 명의 사용자가 접근해도 파일을 수정하지 않으므로 동시성 문제가 발생하지 않는다. 그러므로 락 서버는 락을 허가하게 된다. 하지만 E 락에 대해서는 파일을 수정하는 것 이므로 파일에 E 락이 걸려 있다면, 오직 한명의 사용자만이 파일의 사용이 가능하고 다른 사용자는 파일의 사용이 끝날 때 까지 대기 큐에서 대기해야 한다.

3.5 락킹 절차

본 논문에서 구현된 락킹 모듈에서는 락을 요청하는 객체 수에 따라 두 가지 종류의 락 기법을 제공한다. 단일 락 방식은 하나의 객체에 대하여 락을 요청하는 것이고, 다중 락은 락 서버의 성능을 향상 시키기 위하여 한번에 여러 개의 객체에 대하여 동시에 락을 요청하는 것이다. 단일 락과 다중 락 방식 모두 객체 하나에 대한 락킹 절차는 기본적으로 같다.

락킹 절차는 이미 3.1 장에서 언급한 것과 같이, 어떤 파일에 대한 락을 얻기 위해서는 LLM 의 LLP 와 GLM 의 GLP 를 모두 얻어야만 한다.



(그림 4) 파일에 대한 권한을 얻기 위한 락킹 절차

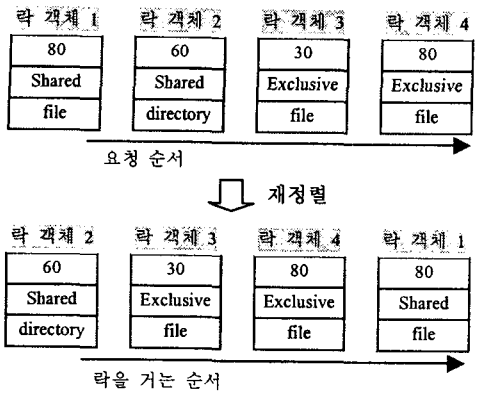
(그림 4)는 파일에 대한 락을 얻기 위한 락킹 절차를 보여주고 있다. 사용자가 파일에 대한 요청을 하면 요청을 받은 클라이언트는 자신의 LLT 에서 새로운 LLP 를 얻어 락 정보를 기입하고, 서버 맵핑 함수를 이용하여 실제 락을 처리할 락 서버를 선택하게 된다. 만일 반환된 락 서버 번호가 LLP 를 가지고 있는 락 서버 번호와 일치하게 되면, LLP 를 가지고 있는 락 서버가 자신의 GLM 을 이용하여 파일에 대한 GLP 를 얻게 된다. 하지만 반환된 락 서버 번호가 자기 자신이 아닐 경우, LLP 를 가지고 있는 락 서버는 통신 모듈을 이용하여 반환된 번호의 락 서버에게 GLP 를 요구하게 되고, 이를 수신한 락 서버는 자신의 GLT 를 검사하여 파일에 대한 접근이 가능할 경우 응답으로 GLP 를 요청한 락 서버에게 전송하게 된다. 파일에 대한 사용을 끝마치고 락을 해제하는 경우에도 락을 얻을 때와 같은 방식으로 동작하게 된다.

3.6 교착상태 발견(deadlock detection)

일반적으로 락킹 프로토콜에서의 교착상태는 사용자가 하나 이상의 락을 한번에 요청하는 다중 락의 경우에 발생하게 된다. 여러 객체에 대한 락의 요청은 락 간의 환형 대기를 형성할 수 있고, 이것으로 인해 교착상태가 발생할 수 있다. 특히 락을 분산하여 관리하는 분산 락킹 방식에서는 중앙 집중 방식에 비해 교착 상태에 쉽게 빠질 수 있다. 이러한 교착 상태를 방지하기 위한 방법으로는 2 단계 락 기법, 타임 스탬프 기법, 그리고 객체 정렬 기법 등이 있다. 본 논문에서 구현된 락킹 모듈에서는 다중 락에서 발생할 수 있는 교착 상태 문제를 해결하기 위해 객체 정렬 기법을 사용하였다.

객체 정렬 기법은 정렬 알고리즘에 의한 정렬 순서에 따라 객체들을 재정렬한 후, 재정렬 순서에 따라 락을 거는 방식이다. 즉 둘 또는 그 이상의 객체가 존재할 때 높은 순서의 객체보다 낮은 순서의 객체에 락을 거는 것이 선행 된다. (그림 5)는 구현된 락킹 모듈에서 락 번호와 객체 종류에 따라 객체를 재정렬하는 방식을 보여준다. (그림 5)에 나타난 다중 락에 대한 구조체는 락 번호, 락 상태, 그리고 객체 종류로

구성된다.



(그림 5) 교착상태 방지를 위한 락 재정렬

(그림 5)에서와 같이, 교착 상태를 피하기 위해서는 디렉토리와 파일에 대하여 순서를 정하고, 파일의 경우에는 모든 사용자가 락 번호에 따라 순서를 재 정렬함으로써 락 번호가 중간에 교차하는 일이 발생하지 않도록 한다. 같은 파일에 대해서는 항상 E 락을 우선으로 하여 동작하게 함으로써 S 락으로 인해 교착 상태가 발생하는 것을 방지한다.

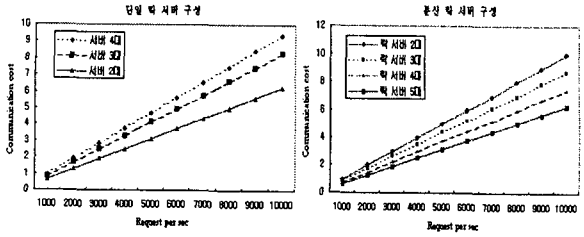
3.7 락 서버 재구성 및 락 전환

락 서버 재구성은 락을 분산하여 관리하는 분산 락 서버 환경에서 분산된 락 정보를 유지하기 위해 반드시 필요한 부분이다. 분산된 락 서버 중에 하나의 락 서버가 갑작스러운 시스템 문제로 GLT 정보를 분실하거나 재 구성을 요구할 경우 전체 락 서버가 자신의 LLT 정보를 이용하여, 해당 락 서버의 GLT 를 재구성하게 된다. 모든 락 서버는 락에 관련된 동작을 수행하기 전에 항상 락 서버 정보의 재구성 여부를 확인한 후, 재구성이 필요할 경우 요청된 락을 큐에 넣고 재구성이 끝난 후에 다시 처리하게 된다. 이렇게 함으로써 전체 락 서버의 LLT 와 GLT 정보에 동기화를 달성할 수 있다.

락 전환은 락이 걸려 있는 동안 호스트가 그 락 모드를 바꾸고 싶어할 때, 그것은 첫번째로 그 락을 해제하고 다른 모드로 락을 다시 요청 해야 한다. 이 과정에서 같은 객체에 대한 락이 같은 호스트에게 제공된다는 어떠한 보장도 없다. 때문에 락킹 모듈은 락을 해제하지 않고 객체에 대한 락 모드를 바꿀 수 있는 락 전환 정책을 제공해주어야 한다. 락 전환은 E 락을 S 락으로 전환하는 하향(demotion) 락 전환과 S 락을 E 락으로 전환하는 상향(promotion) 락 전환이 있다. 이러한 락 전환 정책으로 락 분배에 대한 공평성을 최대화 할 수 있다.

3.8 성능분석

(그림 6)은 전체에 오직 하나의 락 서버만이 GLM 을 구성하고 락을 처리하는 단일 락 서버 경우와 전체 락 서버가 모두 GLM 을 구성하여 락을 처리하는 분산 락 서버 경우에 대한 성능 비교이다.



(그림 6)락 서버 구성에 따른 통신 비용

성능 측정을 위한 환경 설정은 다음과 같다. 락에 대한 요청은 부하 분산을 이용하여 각 서버에 같은 비율(1/n)로 나뉘어 전송되고, 요구된 락 중에 50%는 요청된 락 서버에 의해 처리된다. 그러므로 단일 락 서버의 경우는 n-1/n 의 락 서버에서 통신이 필요하게 되므로 요청 수와 서버 수가 증가할수록 통신 비용은 급격히 증가하게 되지만, 분산 락 서버 경우는 요청된 전체 락 중에 최대 50%만 통신에 의해 처리되므로 요청이 증가하여도 전체 시스템에 커다란 영향을 미치지 않게 된다.

4. 결론 및 향후과제

본 논문은 SAN 환경의 공유 데이터의 일관성을 지키기 위한 락킹 모듈 구현과 실제 동작에 대하여 설명하였다. 새로운 스토리지 솔루션으로 대두 되고 있는 SAN 에서 보다 효율적이고 안정된 락킹 모듈을 위해 많은 노력이 필요할 것으로 본다. 따라서 향후 연구 과제로 이미 구현된 락킹 모듈에 정제된 락 granularity 를 제공하기 위하여 기존의 모듈에 락 모드를 추가하고, 추가된 락 모드를 위해 보다 안정된 락 전환 기법을 제공하여 락 서버의 성능을 향상시키는 것이 목표이다.

참고문헌

[1] Erhard Rahm. "Concurrency and Coherency Control in Database Sharing Systems," University of Kaiserslautern, 1991

[2] K.Amiri, G.Gibson and R.Golding. "Highly concurrent shared storage," Intl. Conference on Distributed Computing Systems, April 2000.

[3] Ken Perslan et al., "Dlock 0.9.6 specification," http://www.sistina.com/gf

[4] N. Kronenberg, H. Levy, and W. Strecker. "VAXCluster: Computer Systems," 4(3):130-146, May 1986.

[5] Knottenbelt, W.J, Zertal, S. and Harrison, P.G, "Performance analysis of three implementation strategies for distributed lock management," Computers and Digital Techniques, IEE Proceedings, Volume: 148 Issue: 4-5, July-Sept. 2001, Page(s): 176 -187

[6] Preslan, K.W., Soltis, S.R., Sabol, C.J., O'Keefe, M.T., Houlder, G., and Coomes, J., "Device Locks: mutual exclusion for storage area networks," 16th IEEE Symposium on, 1999, Page(s): 262 -274