대용량 공유 분산 화일 시스템에서 망 분할 시 순환 리스를 사용한 고장 감내성 향상

(Improving Fault Tolerance for High-capacity Shared Distributed File Systems using the Rotational Lease Under Network Partitioning)

탁 병 철 † 정 연 돈 †† 김 명 호 †††
(Byung Chul Tak) (Yon Dohn Chung) (Myoung Ho Kim)

요 약 서버를 통하여 저장 장치를 사용하는 네트워크 연결형 화일 시스템과 달리, 대용량 공유 저장 장치 화일 시스템에서는 서버들이 데이타 전용망을 통하여 저장 장치를 직접 공유하여 사용한다. 이런 구조에서는 데이타의 일관성을 유지하기 위하여 잠금 관리자가 존재하여 제어망을 통하여 잠금 정보를 주고 받는다. 또한 예기치 않은 제어망의 고장에 대비하여 리스를 사용한다. 하지만 제어망에 분할 고장이 발생할 경우 격리된 서버들은 고장이 해결되기 전까지는 더 이상 작업을 진행할 수 없게 된다. 본 논문에서는 이러한 제어망 분할 고장이 발생한 상황에서도 서버들이 계속 화일 시스템을 사용하여 작업을 진행할 수 있도록 하는 기법을 제안한다. 제안하는 기법은 주기적으로 각 서버들에게 리스를 순환하여 할당하는 방식으로 동작한다. 또한 제안하는 기법은 항상 데이타의 일관성을 유지함을 보인다.

키워드 : 공유 분산 화일 시스템, 망분할 고장, 리스, 데이타 일관성, 고장 감내성

Abstract In the shared storage file system, systems can directly access the shared storage device through specialized data-only subnetwork unlike in the network attached file server system. In this shared-storage architecture, data consistency is maintained by some designated set of lock servers which use control network to send and receive the lock information. Furthermore, lease mechanism is introduced to cope with the control network failure. But when the control network is partitioned, participating systems can no longer make progress after the lease term expires until the network recovers. This paper addresses this limitation and proposes a method that allows partitioned systems to make progress under the partition of control network. The proposed method works in a manner that each participating system is rotationally given a predefined lease term periodically. It is also shown that the proposed mechanism always preserves data consistency.

Key words : shared distributed file system, network partitioning, lease, data consistency, fault tolerance

1. 서 론

인터넷의 확산과 향상되는 계산 능력에 힘입어 응용 프로그램들은 점점 더 많은 데이타를 처리하도록 만들어지고 있으며, 화일 시스템도 이러한 요구사항에 맞추어 네트워크를 사용한 공유가 가능하도록 발전하였다. 기존의 네트워크 저장 장치의 형태는 대용량 저장 장치를 가지고 있는 화일 서버가 시스템들에게 화일 서비스를 제공하는 형태였다. 최근에 와서는 광채널과 같은 새로운 매체의 개발로 인하여 SAN(Storage Area Network)과 같은 공유 저장장치 화일 시스템 구조가 가능하게 되었다[1-4].

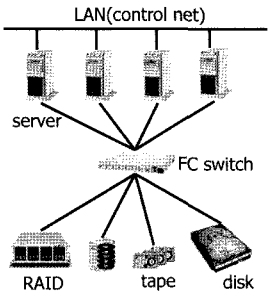
공유 저장장치 화일 시스템에서는 저장 장치만을 위한 전용의 망을 사용하므로 빠른 속도로 직접 저장 장치를 사용할 수 있다. 일반적으로 저장 장치를 연결하는 망은 전송 속도가 우수한 광채널을 사용하여 높은 데이

· 본 연구는 대학 IT연구센터 육성 지원사업의 지원을 받아 수행되었음
† 정 회 원 : 한국전자통신연구원 Embedded S/W 연구단
bctak@dbserver.kaist.ac.kr
†† 종신회원 : 동국대학교 컴퓨터공학과 교수
ydchung@dgu.edu
††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2004년 12월 30일
심사완료 : 2005년 9월 1일

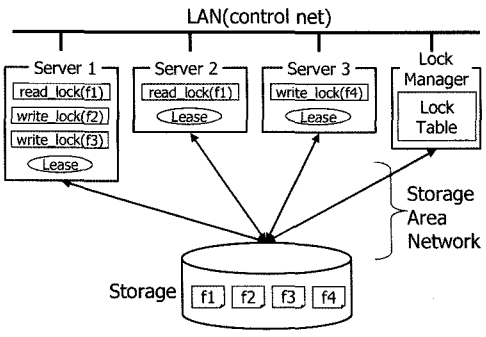
타 처리량(through-put)을 낼 수 있으며, 투명하게(transparently) 저장장치를 공유하므로 운영체제의 관점에서는 마치 자신만의 저장 장치를 가진 것과 같이 동작한다. 또한 화일 서버 역할을 하는 특화된 시스템이 존재하지 않으므로 클라이언트-서버 구조에서와 같은 병목 지점이 존재 하지 않는다.

여러 서버들이 저장 장치를 공유하게 되면 버퍼 캐시의 일치성(coherency)과 데이타의 일관성(consistency) 유지문제가 발생한다. 공유되는 저장 장치의 데이타 일관성을 유지하기 위한 보편적인 방법은 잠금 기법(lock mechanism)이다. 공유 저장장치 화일 시스템 환경에서는 잠금 정보와 메타데이타의 전송은 별도의 제어망을 사용한다. 서버 중 일부는 잠금 관리자로 지정되어 제어망을 통하여 잠금을 획득하고 반환한다. 본 논문에서 가정하는 환경은 이와 같은 잠금 방식을 사용하는 공유 저장장치 화일 시스템 환경이다.

다수의 사용자를 위한 화일 시스템에서는 서비스의 안정성이 매우 중요하다. 따라서 이와 같은 화일 시스템의 설계는 분산 환경에서 발생하는 여러 가지 고장(failure)에 대한 대처가 필요하다. 본 논문에서는 잠금과 메타 데이타 정보를 전달하는 제어망의 고장 시 발생하는 문제를 해결하고자 한다. 기존의 시스템에서는 제어망의 고장이 발생할 경우 리스를 사용하여 리스의 주기 만큼을 기다린 후 할당되었던 잠금을 회수한 것으로 간주하는 방식을 통하여 저장 장치의 데이타 일관성을 유지 하였으나, 이러한 방법으로는 데이타망이 건재함에도 불구하고 제어망의 고장이 물리적으로 해결되기 전까지는 전체 화일 시스템이 동작하지 못하게 된다. 이러한 단점은 제어망에 의존하지 않으면서도 저장 장치로의 접근을 일관되게 제어하는 방법을 적용하여 전체 화일 시스템이 고장 시에도 동작하게 함으로써 극복할 수 있다. 본 논문에서는 리스의 개념을 확장하여 주기적으로 리스를 순환하여 사용하는 순환 리스 기법을 제안한다.



(a) 물리적 구성



(b) 논리적 모델

그림 1 공유 저장장치 화일 시스템의 구조

본 논문의 구성은 다음과 같다. 제2장에서는 관련연구에 대해 알아본다. 제3장에서는 제안하는 기법에 대하여 기술하고 제4장에서는 실험을 통하여 제안한 방법의 특성에 대해 분석한다. 마지막으로 제5장에서 결론을 맺고 향후 연구방향을 제시한다.

2. 관련 연구

네트워크 기반의 대용량 저장 장치에 관한 연구는 저장 장치의 연결 형태에 따라 크게 네트워크 연결형 저장 장치(Network Attached Storage)와 공유 저장 장치(Storage Area Network)로 구분된다. 네트워크 연결형 저장 장치 방식에서는 저장 장치를 탑재한 시스템이 시스템 간에 구축된 네트워크를 이용하여 자료 저장 서비스를 제공하는 형태를 가진다. 이와 같은 구조에서는 LAN과 같이 이미 잘 구축되어 있는 네트워크를 사용하므로 확장성이 좋지만, 자료 저장 서버가 병목이 되는 단점을 가진다. 네트워크 연결형 저장 장치에 속하는 연구로는 NFS[5], AFS[6], Sprite 화일 시스템[7], xFS[8], Frangipani[9] 등이 있다.

공유 저장 장치 시스템은 저장 서비스 제공을 위한 서버가 별도로 존재하지 않고 저장 장치가 직접 네트워크에 연결되는 형태를 가진다. 공유 저장 장치 시스템에 속하는 연구는 Santopia[4], GFS[1,2], StorageTank[3,10,11], GPFS[12], CXFS[13], 등이 있다. 일반적으로 저장 장치를 연결하는 네트워크는 저장 장치만을 위한 특별한 네트워크이며 고속으로 동작한다. 따라서 공유 저장 장치에 연결된 시스템들은 원하는 저장 공간을 빠르게 접근하여 원하는 데이타를 사용할 수 있게 된다. 각 시스템들은 이와 같이 저장 장치에 대하여 일관된 접근을 할 수 있으며 서버 역할을 하는 시스템이 필요하지 않는 등의 장점을 가지지만 네트워크 연결형 저장 장치에 비하여 구축 비용이 높은 단점이 있다.

공유 저장 장치 방식에서는 각 시스템들이 저장 장치의 데이타를 읽어서 시스템의 버퍼로 가져와 필요한 작업을 수행 한다. 이러한 버퍼를 사용하는 시스템에서는

버퍼에 캐시된 데이타의 일관성 문제가 발생한다. 이를 해결하기 위하여 리스[14]가 제안되었다. 이후 단일 리스 개념을 확장하여 서로 다른 리스 유효기를 가진 두 개의 리스를 병행하는 방법인 볼륨 리스[15]가 제안되었다. Frangipani[9]에서도 리스가 도입되어 시스템들의 고장에 대처하였다. 각 시스템은 여러 개의 잠금(lock)을 가졌고, 이 잠금들은 시스템별로 하나의 리스로 관리되었다. 잠금을 무한대의 유효기를 가진 리스로 본다면 Frangipani에서도 볼륨 리스와 유사한 방식이 도입된 것으로 볼 수 있다. StorageTank[3,10,11]에서도 시스템별로 하나의 리스를 사용하였다.

공유 저장 장치에서의 고장은 서버의 고장, 망에서의 고장, 디스크의 고장으로 구분할 수 있는데 여러 SAN 기반의 연구에서 각각의 고장에 대처하기 위한 기법들이 사용된다. 본 논문에서의 관심 사항인 컨트롤 망에서의 고장에 대처하기 위해서 GPFS[12]에서는 그룹 멤버쉽 프로토콜을 사용한다. 망의 고장에 의해서 서버들이 분할되면 다수의 서버를 가지는 그룹의 서버들만 디스크 접근을 허용한다. 하지만 서버가 자신이 다수의 그룹에 속하는지를 판단하는데 시간이 걸리므로 그 동안 소수 그룹의 서버가 디스크를 사용할 수도 있다. 이를 방지하기 위해 펜싱(fencing) 기법을 병행하여 사용한다. 펜싱기법은 SAN망의 스위치가 제공하는 기능으로 SAN스위치에서 특정 서버의 I/O요청을 막는 방법이다. CXFS[13]에서도 망 분할에 대처하기 위한 방법으로 GPFS와 유사한 그룹 멤버쉽 프로토콜을 사용한다. 이 방법은 고장이 발생하였을 때 서버들 중 일부만 계속 동작할 수 있으나 최대 절반의 서버들은 동작을 멈추게 되는 단점이 있다. 또한, GPFS[12]와 CXFS[13] 모두 잠금 기법을 사용하므로 펜싱에 의해 차단된 서버들은 캐시의 변경된 데이타를 디스크에 저장할 수 없으며 잠금을 회수 할 수도 없게 된다. StorageTank[3,10,11]에서는 잠금과 리스를 같이 사용하는데 리스의 갱신을 최소화 하기 위한 기회적(opportunistic) 리스라 불리는 방법를 사용한다. 망 고장이 발생하게 되면 리스가 유효한 동안은 서버가 계속하여 파일을 사용할 수 있으므로 서버는 자신의 캐시를 디스크에 반영할 시간을 가진다. 이 후 리스 유효기가 지나면 잠금관리자는 그 서버의 모든 잠금을 회수한 것으로 판단하게 된다.

3. 순환 리스 기법

본 논문에서 가정하는 공유 저장장치 화일시스템은 데이타 전용의 망이 별도로 존재하며 잠금과 같은 제어 정보는 제어망을 통하여 전달되는 구조이다. 이와 같은 구조에서 제어망에 고장이 발생하여 화일 시스템을 사용하는 시스템들이 망 분할되었을 경우, 리스의 개념을 확장한 순환리스를 사용하여 고장에 대처하는 기법을 제안한다. 본 장에서는 순환리스가 적용되는 상황에 대하여 알아보고 순환리스를 구동하기 위한 준비 작업들에 대해 설명한다. 또한, 순환리스 구동에 필요한 계산법을 소개한다.

3.1 대용량 분산 공유 화일 시스템에서의 고장

SAN을 사용하는 분산 공유 화일 시스템에서는 저장장치 전용망과는 별도로 메타데이타와 제어 정보를 주고받거나 외부 네트워크와 연결하기 위한 용도로 제어망을 사용한다. 이와 같이 함으로써 데이타와 제어 정보의 전송이 분리되어 데이타 전송 속도(through-put)를 향상시킬 수 있다. Santopia[4]에서는 개별 잠금 관리자와 전역 잠금 관리자로 불리는 두 종류의 잠금 관리자가 존재하여, 여러 개의 잠금 관리자 역할을 하는 서버들이 전체 잠금을 분할하여 관리한다. 이 구조에서 발생하는 잠금 정보들은 제어망을 통하여 주고받게 된다. StorageTank에서는 메타 데이타를 저장하는 별도의 서버 시스템이 존재하며, 이들이 제어망을 통하여 연결되어 있다.

제어망의 고장에 의해 망 분할(network partition)이 발생하면 서버는 잠금을 얻거나 반환할 수 없게 된다. 잠금 관리자는 망 분할 되지 않은 서버가 잠금을 요청해도 잠금을 회수할 수 없으므로 요청한 서버에게 잠금을 전달할 수 없게 된다. 임의로 잠금을 전달 할 경우 한 순간에 둘 이상의 서버들이 동일한 데이타를 수정하여 데이타의 일관성(consistency)이 파괴될 수 있기 때문이다. 이러한 문제를 해결하기 위한 목적으로 리스(lease)[12]를 사용한다.

리스의 의미는 잠금 관리자가 나누어준 잠금들을 리스 유효 기간 만큼만 인정하겠다는 것이다. 리스의 동작을 요약하면 다음과 같다. 잠금 관리자는 리스 유효기의 연장 신청이 없을 경우 그 서버에 허용하였던 모든 잠금을 무효화하고 다른 서버에 그 잠금을 나누어주게 된다. 망 분할로 인해 단절된 서버는 리스를 연장할 수 없으므로 자신의 리스 유효기를 검사하여 그 기간 동안만 잠금이 유효한 것으로 간주하고, 유효기 이후로는 잠금이 자동으로 반환된 것으로 간주하여 사용하지 않는다. 그 이후로는 잠금 관리자와 동일한 구획(partition)에 존재하는 서버는 잠금을 정상적으로 사용할 수 있지만 다른 구획의 서버들은 망이 복구되기를 기다려야 한다.

3.2 순환 리스의 개념

제어망에 망 분할이 발생하면 전체 화일 시스템은 더 이상 정상적인 동작을 할 수 없게 된다. 이것은 비록 데이타 망이 건재하여 저장 장치의 데이타를 읽고 쓰는 것은 가능하다고 하여도 잠금, 리스 등과 같은 제어 정보를 시스템 간에 주고받을 수 없기 때문이다.

리스(lease)의 사용은 잠금을 수거하여 데이타의 일관성을 유지시키기는 하지만, 망 분할 된 시스템들은 리스 유효기가 지난 후에는 더 이상 작업을 진행할 수 없는 한계를 가진다. 본 논문에서 제안하는 방법은 이러한 리스의 한계를 해결하고자 한다. 즉, 망 분할이 발생하여 잠금 관리자에 접근할 수 없는 상황에서도 작업을 계속 진행할 수 있는 방법을 제안한다. 제안하는 방법은 리스의 개념을 확장시킨 방법으로 일반적인 리스는 리스 유효기 이후 효력을 상실하는데 반하여 본 논문에서는 주기적으로 리스의 효력을 되살리는 개념을 도입한다. 즉, 망 분할 이후 데이타의 일관성을 유지하며 저장 장치를 사용하기 위해서 사전에 서버들의 순번을 정하여 정해진 시간 구역에서 데이타를 사용할 권한을 부여하는 방식이다. 서버가 다수 일 경우 리스의 효력이 일정 시간 간격 동안 번갈아 가며 인정되어 주기적으로 리스를 회전시키는 것과 같은 효과가 있으므로 이러한 맥락에서 이를 순환 리스(rotational lease)라 부른다. 순환 리스를 사용하기 위해서는 우선 망 고장 발생 시 서버들 간의 순서를 정하여야 하며 최적화를 위하여 사용이 예상되는 데이타의 범위도 정하여야 한다. 그리고 망 고장 발생 시점을 감지하여 정해진 순서대로 정확한 리스 유효기를 계산하여 화일 시스템을 사용하게 된다.

3.3 순환 리스 주기의 계산

순환 리스가 올바르게 동작하기 위해서 가장 중요한 것은 저장 장치에 기록된 데이타의 일관성이 어떠한 경우에라도 유지되도록 하는 것이다. 순환 리스는 리스 유효기를 각 서버마다 다르게 하여 동시에 동일한 데이타가 수정되는 것을 방지하고자 하므로 각 서버들은 리스 유효기가 겹치는 구간이 발생하지 않도록 안전한 리스 유효기를 정확하게 계산하여야 한다.

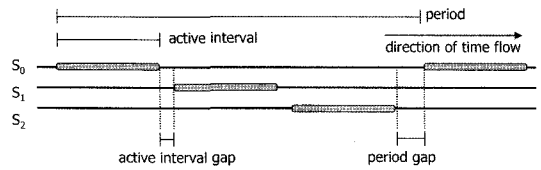


그림 2 순환 리스의 주기(period)와 활성구간(active interval)

그림 2는 순환 리스의 주기와 활성 구간의 개념에 대해 나타내고 있다. 본 논문에서 순환 리스의 계산을 위해 사용하는 두 가지 용어에 대하여 다음과 같이 정의한다.

정의 1. 활성구간(active interval)이란 미리 지정된 서버에 자유로운 디스크 접근 권한이 주어지는 시간 구간이다. ■

정의 2. 주기(period)란 모든 활성구간이 정해진 순서에 따라 한 번씩 사용되는데 소요되는 시간 구간이다. ■

각 서버들의 내부 클럭(internal clock)은 속도에 조금씩 차이가 있을 수 있는데 이러한 속도차이의 비율을 내부 클럭의 유동율이라 부르며 1이상의 범위를 가진다. 즉, 한 서버에서 1000초가 지날 때 마다 다른 서버에서는 1001초가 지난 것으로 인식된다면 내부 클럭이 1.001배 빠르게 동작했다는 의미이며 이 값이 두 서버간의 유동율이 된다. 활성구간과 같이 정확한 시간 구간의 계산을 위해서는 이러한 유동율을 계산에 반영하여야 한다. 따라서 다음과 같은 가정을 도입한다.

가정 1. 최대 내부 클럭의 유동률(drift rate) ρ의 상한이 존재하여 어떠한 두 서버 간에도 한 곳에서의 시간 t는 다른 서버에서 (t/ρ,tρ) 구간 이내의 길이가 된다. ■

이 가정이 의미하는 바는 파일 시스템을 사용하는 서버들간에 존재하는 유동율 중 가장 큰 값이 ρ를 넘지 않는다는 것이다. 이 값은 실제 측정을 통해서 구할 수도 있으나 내부 클럭의 알려진 오차율을 참고로 하여 안전하게 충분히 큰 값을 선택할 수도 있다. 넓은 의미에서의 유동율은 절대시간과의 속도차를 의미 하지만 본 논문에서는 실제 시간과의 동기화가 필요하지 않으므로 서버들 중 가장 빠른 것과 가장 느린 것의 차이를 의미하는 용어로 사용한다. 본 논문에서 필요한 추가적인 가정은 네트워크 전송 시간에 상한선이 존재함을 의미하는 동기적 시스템(synchronous system)[16]에 관한 가정이다.

가정 2. 최대 네트워크 전송 시간에 상한선이 존재하여 한 서버에서 다른 서버로 메시지를 보낼 경우 일정 시간 이내에 반드시 전송 된다고 가정한다. ■

순환 리스의 주기를 계산하기 위해 다음과 같은 기호들을 사용하기로 한다.

ρ : 최대 내부 클럭의 유동률(drift rate).

τ : 활성 구간의 시간 길이.

r : 주기 내에서의 활성 구간의 순번.

δ : 최대 네트워크 전송 지연 시간.

본 논문에서는 순환 리스를 사용하는 서버를 S, 주기 번호를 q, 활성 구간은 r로 표시한다. 각 서버들이 자신의 활성 구간의 시작 시점을 찾기 위해서는 먼저 현재 주기의 시작 시점을 찾고 주기 내에서의 활성 구간의 시작 시점을 찾는다. 이 개념을 수식화 하기 위하여 아래와 같이 몇 개의 기호를 도입한다. 그림 3에서는 이러한 개념을 도식화하여 보여주고 있다.

Hq,r : r번째 활성 구간을 사용하는 서버에서 기준점부터 q번째 주기의 시작 시점까지의 시간(Hop).

Br : 주기의 시작 점부터 r번째 활성 구간이 시작 하는 시점까지의 시간 길이(Breadth).

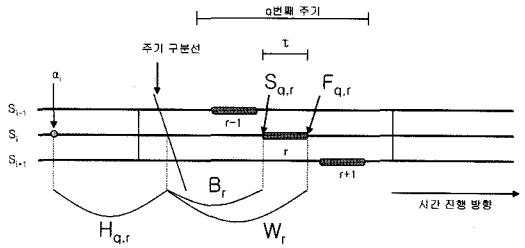


그림 3 시간 계산에 사용되는 기호들

Wr : 주기의 시작 점부터 r번째 활성 구간이 끝나는 시점까지의 시간 길이(Width).

Sq,r : r번째 활성 구간의 시작(Start) 시점.

Fq,r : r번째 활성 구간의 종료(Finish) 시점.

서버가 구하고자 하는 값은 자신의 활성 구간의 시작 시점인 $S_{q,r}$이다. 그림 3에 의하면 다음이 성립함을 알 수 있다.

$$S_{q,r} = \alpha_i + H_{q,r} + B_r \quad (1)$$

그림 3과 식 (1)에서의 α_i는 서버 S_i가 $S_{q,r}$값을 구하기 위한 계산의 시작점으로 사용하는 시점으로 서버 S_i가 순환 리스 모드로 진입하는 시점을 의미한다. 이 시점은 3.5절에서 설명되는 메시지를 사용하여 얻게 되는데 이 메시지는 주기적으로 망의 고장이 발생하였는지를 검사하기 위하여 서버들이 주고받는 메시지이다. 식 (1)을 통하여 계산된 활성구간들은 갱신작업에 대한 데이타의 일관성을 보장하기 위하여 겹치는 시간구간이 발생하기 않아야 한다. 이러한 성질을 다음과 같이 정의한다.

정의 3. 구간 비중첩의 성질(Non-overlapping property) : 임의의 서로 다른 두 활성구간에 대하여 주기를 각각 p, q라 하고 활성구간 번호를 r, s라 할 때 $S_{p,r} < S_{q,s}$면 반드시 $F_{p,r} <= S_{q,s}$이어야 한다. ■

정의 3은 임의의 주기와 활성구간에 대하여 정의하고 있으나 주기에 대한 정의(정의 2)와 연계하여 볼 때에, 서로 다른 주기에서는 활성구간이 중첩될 수 없으므로 정의 3에서는 실질적으로 동일한 주기내의 서로 다른 활성구간이 겹치지 않는다는 의미로 해석할 수 있다. 식 (1)을 사용하기 위해서 먼저 $H_{q,r}$과 B_r의 계산식을 구하여 대입한다. 식 (1)에서 α_i항은 순환리스 모드의 시작점을 의미하며 상수이다. 우선 B_r을 구하는 방법을 살펴본다.

식 (1)에서 B_r을 계산하는 방법을 그림 4를 통해 살펴보도록 한다. 그림 4는 세 서버들이 서로 다른 활성구간을 사용할 경우의 B_r의 계산을 도식화하여 보여준다. S_1이 두 번째 활성구간을 사용하기 위해서는 S_0의 첫 번째 활성구간이 끝나기를 기다려야 한다. S_1의 입장

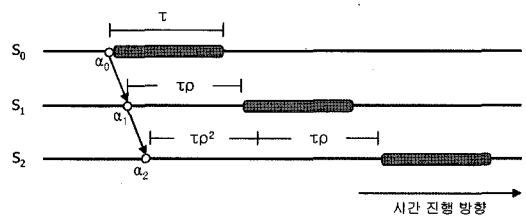


그림 4 주기내에서의 B_r의 계산

에서는 가정 1에 의하여 τρ만큼의 시간은 S_0의 τ보다 반드시 크거나 같다는 것을 확신할 수 있다. S_2의 입장에서도 마찬가지로 S_1가 사용한 τρ+τ에 ρ를 곱하면 S_1의 활성구간이 종료되었음을 확신할 수 있다. 유동율ρ는 현재 서버들 사이에서의 내부 클럭 속도의 차이 중 최대값이므로 ρ를 곱하는 것은 대기시간의 최대치를 기다려서 구간이 중첩되지 않도록 하는 효과가 있다. 모든 계산에서 항상 이렇게 ρ를 곱하므로 실제 필요 대기시간과의 차이가 매 계산마다 누적되어 점점 대기 시간은 증가하게 된다. 이와 같은 방식으로 계산할 경우 활성구간 r에 대해 일반화된 B_r은 다음과 같은 식으로 표현된다.

$$B_r = \rho^r\tau + \rho^{r-1}\tau + \ldots + \rho\tau = \tau\sum_{x=1}^{r}\rho^x \quad (2)$$

유사한 방법으로 W_r의 식은 $W_r = \tau\sum_{x=0}^{r}\rho^x$ 와 같이 표현된다. 이 계산식으로부터 한 주기 내에서의 활성구간들 사이에 겹치는 구간이 발생하지 않음을 소정리 1을 통하여 보인다.

소정리 1. 주기 q에서 활성구간의 순서 m,n에 대하여 m<n이면 항상 $W_m \le B_n$를 만족한다.

증명. 우선 인접한 활성구간의 사이에서 소정리가 성립함을 보인 후, 임의의 두 활성구간 사이에도 성립하는 것으로 일반화 한다. 두 인접한 활성구간을 나타내기 위하여 m=r과 n=r+1이라 할 경우, W와 B의 정의에 의하면 W_r과 B_{r+1}은 다음과 같은 관계식이 성립한다.

$$B_{r+1} = \tau\sum_{x=1}^{r+1}\rho^x \ , \ W_r = \tau\sum_{x=0}^{r}\rho^x$$

$$\therefore B_{r+1} = \rho\tau\sum_{x=0}^{r}\rho^x = \rho W_r$$

이 관계식을 사용하여 소정리의 $W_m \le B_n$을 정리하면 다음과 같다.

$$W_r \le B_{r+1}$$

$$W_r \le \rho W_r$$

이 식에서 부등호의 서로 반대편에 있는 W_r은 표현은 같지만 다른 τ값을 가진 다른 서버에서 계산된 W_r

이므로 절대시간으로 보면 다른 값이 될 수 있다. 따라서 서버를 구분하기 위하여 아래 첨자에 활성구간의 순번 m, n을 추가 하여 다시 아래와 같이 표현한다.

$$W_{r,m} \le \rho W_{r,n} \quad (3)$$

이 식의 부등호가 성립하는가를 살펴보면 다음의 세 가지 경우가 있을 수 있다.

① $W_{r,m} < W_{r,n}$

② $W_{r,m} = W_{r,n}$

③ $W_{r,m} > W_{r,n}$

우선, ①과 ②인 경우는 ρ가 1보다 크므로 식 (3)을 만족한다. ③의 경우는 다시 두 가지의 경우가 존재한다.

① ρ의 값이 m과 n사이의 비율인 경우

② ρ의 값이 m과 n사이의 비율이 아닌 경우

첫 번째 경우는 $W_{r,m}=\rho W_{r,n}$가 된다. 두 번째 경우는 m과 n의 비례가 ρ보다 작으므로 ρ를 곱하면 $W_{r,m}<\rho W_{r,n}$가 된다. 따라서 모든 경우에 대하여 식 (3)이 만족한다. m과 n이 인접하지 않은 경우로 확장하여도 여전히 식 (3)은 만족한다. ■

식 (1)에서 한 주기 내에서의 활성구간의 위치를 의미하는 B_r을 식 (2)와 같이 구하였으므로 남은 부분인 $H_{q,r}$를 계산하는 방법을 알아보도록 한다. 그림 5는 주기 q에서의 $H_{q,r}$를 계산하는 방법을 나타내고 있다.

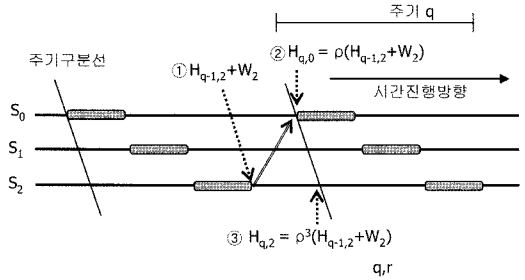


그림 5 주기 시작점 $H_{q,r}$의 계산

먼저 $H_{q,0}$는 q-1번째 주기가 완전히 끝난 후에 시작되므로 가정 1을 사용하여 ①의 식에 ρ를 곱하여 ②를 얻는다. ③에서는 세 번째 활성 구간을 사용하는 S_2의 $H_{q,2}$를 ②를 사용하여 계산하는 식을 보여준다. 활성구간의 번호가 증가한 만큼 ρ가 곱해지는 것을 볼 수 있다. 그림 5는 현재 활성구간이 세 개 존재할 경우를 나타내지만, 한 주기에 활성구간이 m개 존재한다고 할 때 이를 일반화하여 식으로 표현하면 다음과 같다.

$$H_{q,0} = \rho(H_{q-1,m-1} + W_{m-1}) \quad (3)$$

그리고, 주기 내에서 r번째 활성구간을 사용할 경우는 다음과 같이 표현된다.

$$H_{q,r} = \rho^r H_{q,0} = \rho^{r+1}(H_{q-1,m-1} + W_{m-1}) \quad (4)$$

식 (2)와 식 (4)를 식 (1)에 대입하여 원하는 시점을 계산하는 식으로 사용할 수 있다.

정리 1. 다음의 식을 사용하여 얻어진 활성구간의 시작시점은 구간 비중첩의 성질을 만족한다.

$$S_{q,r} = \alpha_i + H_{q,r} + B_r \text{ where}$$

$$B_r = \tau\sum_{x=1}^{r}\rho^x \text{ and } H_{q,r} = \rho^{r+1}(H_{q-1,m-1} + W_{m-1})$$

증명. 주기와 활성구간 번호가 각각 p,r과 q,s인 두 개의 서로 다른 임의의 활성구간이 있을 때 $S_{p,r}$와 $S_{q,s}$은 다음과 같이 표현된다.

$$S_{p,r} = \alpha_r + H_{p,r} + B_r$$

$$S_{q,s} = \alpha_s + H_{q,s} + B_s$$

구간 비중첩의 성질의 조건인 $S_{p,r} < S_{q,s}$이 만족된다고 가정할 $F_{p,r}$을 식으로 다음과 같이 표현할 수 있다.

$$\begin{aligned} F_{p,r} &= S_{p,r} + \tau \\ &= \alpha_r + H_{p,r} + (B_r + \tau) \\ &= \alpha_r + H_{p,r} + W_r \end{aligned}$$

$F_{p,r}$의 각 항에 대해서 $S_{q,s}$와 비교해 보면

정의에 의해서 $\alpha_r <= \alpha_s$이다.

p=q일 경우, 즉 같은 주기일 경우, 식 (4)를 참조하면 $H_{q,s} = \rho(s-r)H_{p,r}$이므로 $H_{p,r} <= H_{q,s}$이다.

p<q일 경우, 그림 5의 ①에서 ②으로 진행하는 식을 참조하면 주기의 값이 다르면 ρ의 곱에 의해 순환주기의 번호에 무관하게 $H_{p,r} <= H_{q,s}$이 성립함을 알 수 있다.

r<s이므로 s는 최소 r+1 이상의 값이다. s=r+1이라 할 경우 $\rho Wr = B_{r+1} = B_s$이므로 $F_{p,r}$의 W_r과 비교하면 $W_r <= B_{r-1}$이 된다. 따라서 $W_r <= B_s$이다.

그러므로 위의 세 가지 항을 종합하여 더해 보면 $F_{p,r} <= S_{q,s}$를 만족하므로 구간 비중첩의 성질을 만족한다. ■

3.4 순환 리스의 주기 결정

망 분할이 발생하기 이전에 활성구간의 개수와 활성구간에 속하는 서버가 결정되어야 한다. 가장 간단하게 한 서버가 하나의 활성구간을 차지하도록 정한다면 활성구간의 개수는 화일 시스템을 사용하는 서버의 수이다. 하지면 서버의 수가 수십 혹은 수백 개의 규모일 경우는 활성 구간의 개수가 많아져 주기가 지나치게 길어질 수 있다. 따라서 서버들이 사용하는 데이타 영역이 상호 배타적인 경우를 고려하여 한 활성구간을 공유하도록 함으로써 활성구간의 개수를 최소화할 수 있다. 사용 데이타가 서로 다른 서버들을 하나의 그룹으로 묶는 방법으로, 일반적인 화일시스템의 디렉터리 구조를 활용하는 방법을 생각해보자.

서버들은 디렉터리를 지정함으로써 망 분할이 발생하였을 경우 지정한 디렉터리 및 하위 디렉터리에 속한 화일들만을 사용하겠다는 것을 선언한다. 그림 6(a)는 예제 디렉터리 구조에서 다섯 개의 서버들이 데이타 영역을 지정한 모습을 보여주고 있다. 점선으로 표시된 삼각형에서 보이듯이 하나의 디렉터리를 지정함으로써 그 노드를 루트로 하는 서브 트리가 모두 사용할 데이타 영역에 포함된다. 링크는 점선 화살표로 표시하였다.

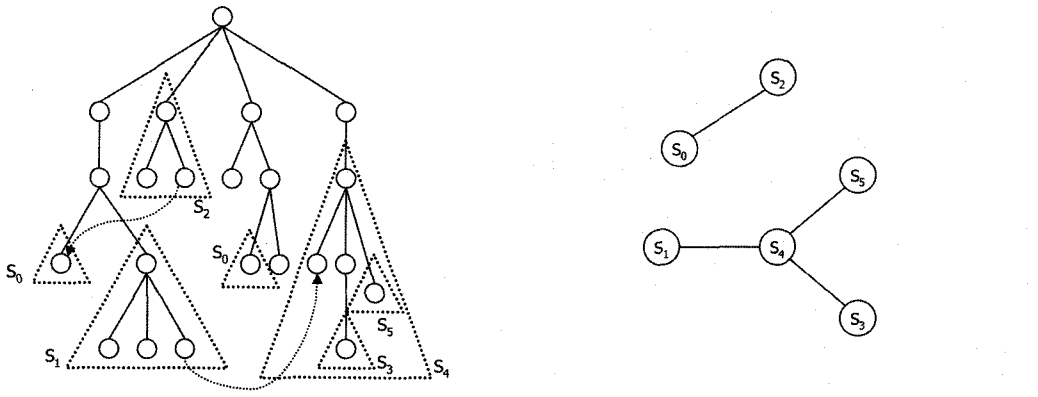
각 서버들이 지정한 작업 영역에 대한 정보를 사용하여 부모-자식 관계를 나타내는 그래프로 변환한다. 이 그래프에서 각 서버들은 노드가 되며 각 노드들 간에 부모-자식 관계가 존재할 경우 에지로 연결한다. 각 서버의 쌍마다 관계를 찾아야 하므로 서버의 개수가 n일 경우 $O(n^2)$이 걸리게 된다. 그림 6(b)는 이와 같은 방법으로 그림 6(a)를 사용하여 생성한 부모-자식 관계 그래프이다.

노드 간에 존재하는 에지는 중복된 데이타 영역을 사용하게 됨을 의미하므로, 두 서버는 하나의 활성 구간을 공유할 수 없다. 활성구간에 하나의 색을 부여할 경우 활성 구간의 최소화는 이 그래프를 사용한 그래프 컬러링 문제로 변환된다. 부모-자식 관계 그래프의 크로마틱 수(chromatic number) $\chi(G)$를 찾는 것은 최소한으로 필요한 활성 구간의 수를 구하는 것과 같으며, 각 노드에 색(color)을 부여하는 것은 활성 구간을 서버들에 배정하는 것과 같다. 그래프 컬러링 문제는 NP-complete 계열인 것이 알려져 있다[17]. 그래프 컬러링 문제는 오래 전부터 연구된 문제이며 여러 가지 휴리스틱 방법들이 개발되었다. 부모-자식 관계 그래프를 구한 이후 그래프 컬러링의 휴리스틱 알고리즘을 사용하여 최소 색의 개수에 대한 근사 값을 쉽게 구할 수 있다.

3.5 순환 리스의 구동

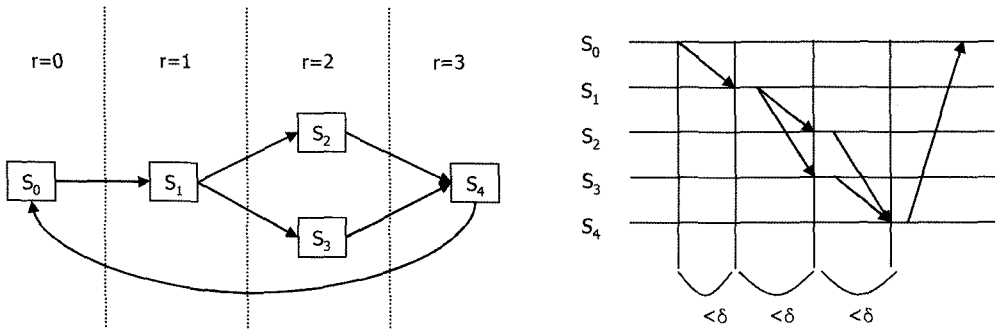
화일 시스템은 대부분의 시간동안 정상적인 동작을 하다가 망 분할이 발생하는 시점부터 순환 리스 기법이 적용된다. 따라서 화일 시스템을 이용하는 서버들은 순환 리스 기법을 적용할 시점을 찾기 위해서 망 분할을 감지할 수 있어야 한다. 분산 환경에서 두 개체간의 망 분할 감지는 전송 메시지에 대한 응답을 확인하는 방법을 사용한다. 분산 시스템의 특성상 망 분할이 발생하는 시점을 정확히 감지하기는 불가능하며 정확도를 높이려고 할수록 메시지의 수가 증가하므로 감지 비용이 증가한다. 본 논문에서는 CXFS[13]와 GPFS[12]에서 망 분할을 감지하기 위하여 사용하는 박동(heart-beat) 메시지와 유사한 방식으로 메시지 전송을 통해서 망의 고장 발생 여부를 감지한다. 이 메시지는 순환 리스의 주기에 해당하는 시간 간격마다 주고받게 된다.

모든 서버들은 주기 결정 단계에서 정해진 서버들의 순서대로 메시지를 주고받는다. 그림 7(a)에서는 메시지가 서버들 간에 사이클을 형성하며 전송되는 과정을 보여준다. 이 메시지를 편의상 기호로 RMSG로 표시한다. 서버들을 S_i로 나타내고 r을 활성구간의 번호라 할 때, 그림에서는 다섯 개의 서버가 메시지를 주고받으며 S_2와 S_3는 동일한 활성구간에 할당되었음을 볼 수 있다. 메시지가 모든 서버를 돌아 사이클이 완성되면 각 서버들은 RMSG 받은 시점을 α_i로 기록한다. 그림 7(b)는 RMSG의 전달과정을 시간 관점에서 보여준다. 본 논문에서는 메시지 전송 지연시간의 상한선이 보장되는 동기적(synchronous) 시스템을 가정하므로 그림 하단의 δ는 S_i가 RMSG를 받은 순간부터 S_{i+1}이 RMSG를 S_i로부터 받기까지 걸리는 시간의 상한선이 된다. 이렇게 동작하는 RMSG를 사용하여 그림 8에서는 실제로 망 분할이 발생하였을 경우 순환 리스 기법으로 넘어가는 과정을 보여준다.



(a) 작업 그룹의 지정 (b) 부모-자식 관계 그래프

그림 6 작업 그룹 및 부모-자식 관계 그래프



(a) 메시지 흐름 (b) 타이밍 다이어그램

그림 7 RMSG(Round Message)

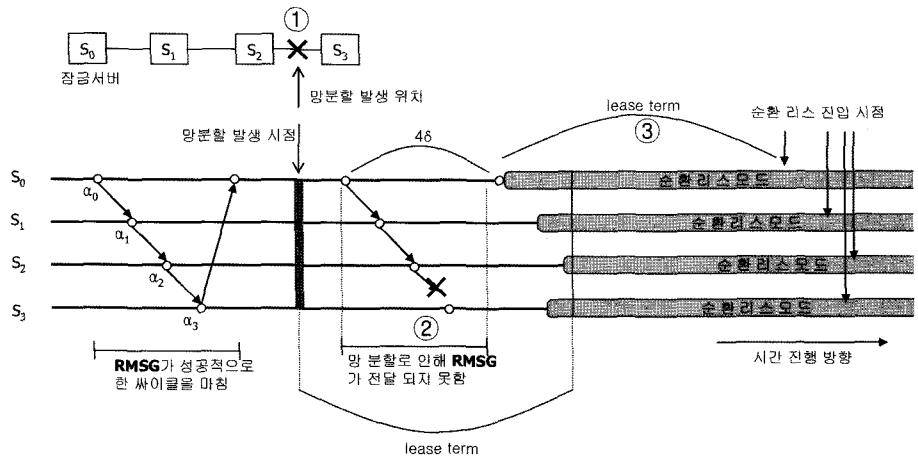


그림 8 망 분할의 감지 및 실행 단계 진입

그림 8은 네 개의 서버들이 독립적으로 서로 다른 시점에서 망 분할을 감지하고 순환 리스를 적용하는 과정을 나타내고 있다. ①에서는 네 개의 서버들의 물리적인 위상을 보여주며 S_2와 S_3사이에서 고장이 발생하였음을 볼 수 있다. 망 분할이 일어난 시점에서는 어떤 S_i도 이를 즉시 알지 못한다. S_0는 한 시점에 α_i를 새로 설정하기 위해 RMSG의 전달을 시도한다. 이때 이미 망 분할이 발생한 이 후이므로 S_0는 RMSG를 받지 못하고 고장을 감지한다. ②에서 S_0는 4δ 만큼을 기다려도 RMSG가 S_3로부터 오지 않으므로 순환 리스를 적용하기 시작한다. 동시에 망 고장의 발생을 브로드캐스팅하여 다른 서버에 알려준다. 동일한 파티션에 속한 서버는 이를 수신하고 순환리스를 적용하게 된다. 다른 파티션에 위치한 S_3의 경우도 RMSG를 받지 못하여 네트워크 분할을 감지하고 실행 단계로 진입한다.

서버들은 순환 리스를 사용하기 전 ③과 같이 한 번의 리스 유효기 만큼의 시간을 기다린 후 동작한다. 그림에서 '순환리스 진입시점'으로 표시된 부분이 이 지점을 나타내는데 이는 다음과 같은 경우를 위해서이다. 만약 S_i가 망 분할이 일어나기 직전에 리스(lease) 갱신에 성공하였다고 하면, S_i는 망 분할 이후 리스 유효기 만큼을 정상적인 작동을 할 것이다. 이 서버는 리스 유효기가 끝날 때까지 이를 감지하지 못하고 자유롭게 데이타를 사용하게 된다. 따라서 먼저 망 고장을 감지한 서버는 여전히 정상 작동하는 서버와 충돌하지 않기 위해서 적어도 한 리스 유효기를 기다려 주어야 한다. 결과적으로 망 분할 발생 후 순환 리스 모드에 진입하는 데에는 성공적인 RMSG의 전달 이후 두 주기 이후가 된다. 그림 9는 이러한 순환 리스 모드의 진입과 동작에 관한 순서를 보여주는 순서도이다.

지금까지 순환리스 기법과 관련하여 활성 구간의 결정, 망 분할 시점의 감지 및 정확한 활성 구간의 계산과 관련된 사항들을 설명하였다. 활성 구간의 결정에 관한 방법은 상황에 맞는 여러 가지 방법을 사용하는 것이 가능하며 이미 존재하는 여러 알고리즘을 사용하여 최적에 가까운 활성 구간의 개수를 결정하는 것이 가능함

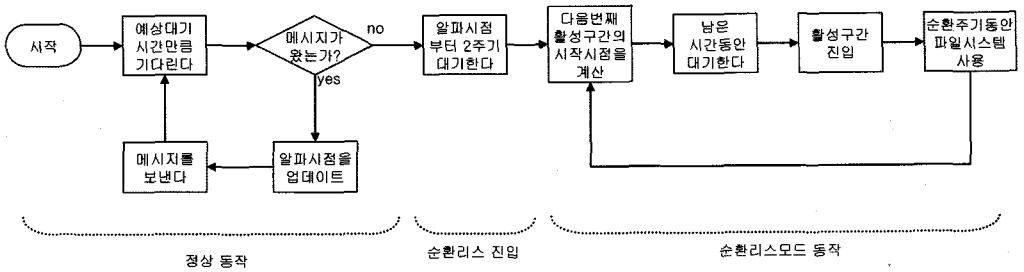


그림 9 순환리스 진입 및 동작 순서도

을 보았다. 망 분할 시점의 감지는 일반적인 방법인 박동(heart-beat) 메시지 방법을 사용하였다. 활성 구간의 계산에서는 구간의 겹치는 부분이 발생하지 않도록 유동율을 감안하여 계산을 하도록 하였다. 이러한 제안 사항들을 종합하여 최종적으로 순환 리스 기법은 데이타의 일관성을 유지하면서 동작하는 기법임을 제시한다.

정리 2. 망 분할 이후 순환 리스의 적용을 통해 공유 저장 장치의 데이타에 대한 일관성을 유지하면서 지속적으로 동작한다.

증명. 본 논문에서 제안하는 순환 리스 기법은 크게 활성 구간의 결정, 망 분할 시점의 감지, 그리고 활성 구간의 계산으로 구성된다. 공유 저장 장치에서 데이타의 일관성이 유지 되기 위해서는 한 순간에 데이타를 하나의 개체만이 접근하여야 한다. 활성 구간의 결정, 망 분할 시점의 감지는 데이타의 접근과는 무관하며 활성 구간의 계산에 있어서 활성 구간이 겹치는 구간의 발생하지 않아야 한다. 정리 1에 의해 제안된 활성 구간 계산법은 비중첩의 성질(non-overlapping property)을 만족하므로 계속적인 순환리스의 동작하에서도 데이타의 일관성은 유지된다. ■

지금까지 제안한 방법은 기존의 방법들과 비교하여 다음과 같은 차이를 가진다. StroageTank[3,10,11]에서는 제안하는 기법과 유사하게 정상 동작시에는 서버당 하나의 리스를 할당하여 사용한다. 망 분할이 발생하게 되면 이 리스가 유효한 동안 망 분할 된 서버들은 자신의 캐시에 있는 수정된 데이타를 디스크에 반영한 후 디스크 사용을 단절하게 된다. 이에 반해 제안하는 기법에서는 한 단계 더 나아가 망 분할이 된 상태에서도 시간 간격에 의해 리스를 재유효화 하여 순차적으로 서버들이 디스크를 사용하게 된다. 이렇게 함으로써 기존의 방식에서는 제공되지 않았던 파일 시스템의 가용성을 확장하여 준다. 단, StorageTank에서는 평시에 망 분할을 감지하기 위한 박동(heart-beat) 메시지를 필요로 하지 않는다는 점이 있으나 실제 시스템에서 이 메시지의 전달로 인한 오버헤드는 미미하므로 크게 문제되지는 않는다. CXFS[13]와 GPFS[12]에서는 다른 방식을 적용하여 고장에 대처한다. 이들에서 사용되는 방식은 멤버쉽 프로토콜을 적용한 파티션 구분 방식으로써, 제안 기법과 동일하게 평시에는 박동 메시지를 전송하여 망 분할을 감지한다. 망 분할을 감지한 후 각 서버는 멤버쉽 프로토콜을 사용하여 자신이 다수 그룹의 파티션에 속하는지 소수 그룹에 속하는지를 판단한 후 다수 그룹인 경우만 디스크를 계속 사용하게 된다. 이러한 방식에서는 소수 그룹에 속한 서버는 모든 작업을 멈추게 되며, 또한 소수 그룹의 서버가 캐시에 갱신한 데이타를 디스크에 반영할 기회를 가지지 못하는 문제점이 있다. 또 다른 문제점으로는 다수/소수 그룹 판단 알고리즘이 완료되기 까지 걸리는 시간이 얼마인지 알 수 없다는 점도 있다. 제안하는 기법에서는 이에 반해 모든 서버들이 계속 작업을 진행할 수 있으며 반영되지 못한 데이타로 인한 일관성 문제를 발생시키지도 않는다.

4. 실험 및 분석

본 논문에서 제안하는 순환 리스 기법의 동작을 관찰하기 위하여 시뮬레이터를 구현하여 실험을 하였다. 실험에 사용된 시스템은 Pentium 2.4GHz CPU이며 메모리는 512Mbytes를 가지고 있다. 구현된 시뮬레이터는 쓰레드 기반의 윈도우 MFC API를 사용하여 구현되었다. 화일시스템에 참여하는 서버들은 각각 하나의 쓰레드로 구현되었으며 제어망의 동작을 모델링하기 위하여 잠금 관리자와 서버(쓰레드)들은 소켓 통신을 사용하여 제어정보를 주고받는다. 제어망의 동작을 시뮬레이션하기 위하여 쓰레드간 소켓 통신을 하는 것은 실제로 제어망을 통하여 잠금 정보를 주고받는 것과 같은 방식이므로 실제 제어망에서의 메시지 전달 지연 시간을 제외하면 실제 동작과 동일하다. 시뮬레이터는 동작을 시작하면 정해진 개수의 쓰레드를 생성하여 화일 시스템을 사용하는 서버들을 구동시키며 이 쓰레드들이 무작위로 원하는 화일의 잠금을 얻거나 잠금을 반환하는 동작을 수행한다. 잠금 매니저는 별도로 하나의 인스턴스로 존재하여 서버들의 잠금 요청을 처리한다. 시뮬레이터에서는 인위적으로 망 분할 상태를 생성할 수 있으며 망 분

할을 한 후 서버들은 순환 리스 모드로 전환하여 동작을 하게 된다.

실험에서는 순환 리스 모드로 전환한 후 본 논문에서 사용한 몇 가지의 파라미터들을 변화시키면서 이 값들의 변화가 순환 리스 동작에 어떠한 영향을 끼치는지를 실험하여 보았다. 변화를 준 파라미터로는 참여 서버들의 개수, 리스 유효기 그리고 내부 클럭의 유동율이다. 실제 환경에서의 이들 파라미터들의 값과 실험에서 사용한 값들은 표 1과 같다.

실험에 사용된 파라미터들의 실제 값은 표에서와 같이 특정 범위에 속하게 되므로 실험에서는 각 범위내에서 적절한 값을 선택하였다. 내부클럭 유동율의 경우는 실제 값의 범위와 실험에 사용한 값이 동일하게 하였다. 서버의 수는 3가지의 경우로 제한하였다. 서버의 수가 변함에 따라 어떻게 다른 결과가 나오는지를 살펴보기 위해서는 적어도 두 가지 이상의 다른 서버의 수를 설정할 필요가 있었고 현실적으로 적당한 수준의 서버의 수라고 판단하였다. 그리고 서버의 수가 지나치게 많아서 CPU의 점유율이 100%가 넘어버리지 않도록 하였다. 리스유효기의 길이는 수초이상으로 설정할 경우 유동율이 1.000001과 같이 아주 작고 서버의 수가 수십개 정도인 상황에서는 한번의 주기를 마치기 위해서는 적어도 수십초가 필요하게 되고 그 주기 간격의 변화량도 매우 적어 실제 변화를 감지하기 위해서는 수십여일 동안 실험을 가동하여야 하는 비현실적인 면이 있어서 표에서와 같이 적절한 네 가지의 리스유효기만을 선택하였다.

그림 10은 세 종류의 파라미터를 변화 시켰을 경우 순환 리스 모드의 진입 후 한 서버에서의 활성 구간의 간격의 변화를 보여주고 있다. 그림 10(a)는 서버의 개수를 변화 시킨 경우이며 그림 10(b)는 리스 유효기의 길이를 변화 시킨 경우이다. 내부 클럭의 유동율을 고려한 계산에서 활성 구간 사이의 간격이 점점 벌어져서 전체 순환 리스의 동작에서 활성 구간의 시작이 점점 늦춰지는 것을 알 수 있다.

이와 같은 현상을 활성 구간을 계산하는 식으로부터 확인하여 볼 수 있다. 특정 주기 q에서의 주기의 간격은 $H_{q,0}-H_{q-1,0}$로 표현된다. 그런데 수열형태의 식(3.4)를 전개하여 정리하면 다음과 같다.

$$H_{q,r} = W_{m-1}\left(\sum_{x=0}^{q-1}\rho^{mx+r+1}\right)$$

따라서 이를 이용하면 주기간의 간격은

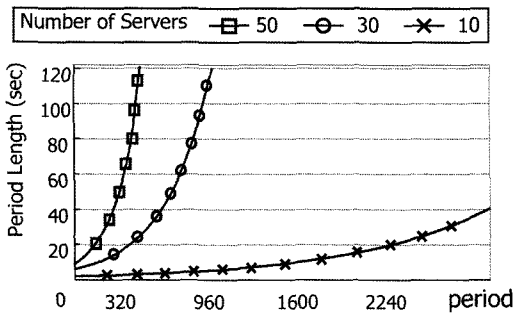
$$H_{q,0} - H_{q-1,0} = c\rho^{m(q-1)}$$

where $c = \rho W_{m-1}$

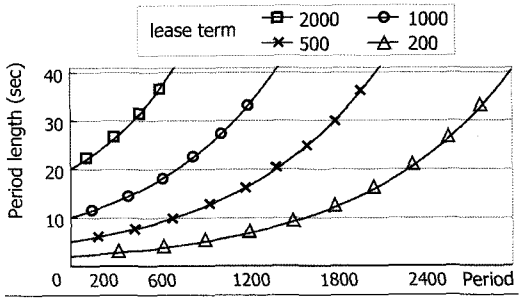
으로 정리된다. 이는 지수 함수의 형태로서 시간이 지날수록 주기의 간격은 지수 함수적으로 증가함을 알 수 있다. 이 식에 의하면 τ가 0.1초이고 서버가 50개일 경우 주기의 길이가 약 두배가 될 때 까지 약 20시간이 걸리게 된다. τ가 1초정도이면 약 8일 정도가 소요된다. 네트워크 분할이 발생한 이후로 본 논문에서 제안하는 방법을 사용하여 동작하기 시작할 경우 τ의 설정에 따라 다르기는 하지만 적어도 20시간에서 8일이 경과하기

표 1 실험 파라미터의 값 설정

파라미터	실제값의 범위	실험에 사용한 값
내부클럭 유동율 ρ	1+1*10-3 ~ 1+1*10-6 sec/sec	1.001, 1.0001, 1.00001, 1.000001 sec/sec
서버의 수	10-100 정도	10개, 30개, 50개
리스유효기 τ의 길이	수 millisec ~ 수 sec	200, 500, 1000, 2000 msec



(a) 서버 수의 변화에 따른 활성 구간의 간격 변화
(리스유효기: 200ms, 유동율: 1.0001)



(b) 리스 유효기의 변화에 따른 활성 구간의 간격 변화
(서버수: 10, 유동율: 1.0001)

그림 10 파라미터의 변화에 따른 순환 리스의 동작 변화

전에는 주기의 길이가 두 배 미만으로 유지 된다. 따라서 네트워크 분할이 복구되기 전까지 제안하는 기법은 주기의 틈을 적절히 작은 수준으로 유지한다. 이 정도의 시간은 대부분의 서버에서 네트워크 분할이 발생하였음을 알고 복구할 수 있는 충분한 길이의 시간이다.

5. 결 론

본 논문에서는 분산 공유 화일 시스템에서 망 분할 고장이 발생하여도 작업을 진행할 수 있는 순환 리스 방법을 제시하였다. 순환 리스 방법은 대용량의 공유 저장 장치를 사용하는 파일 시스템에서 망 분할 고장을 대처하기 위한 방법으로, 제어망의 고장으로 인한 잠금과 리스 제어가 불가능한 상황에서 데이타에 대한 접근을 통제하기 위하여 미리 정해진 리스 구간을 순차적으로 각 서버에 할당하는 방법을 사용하였다. 이를 위해서 첫 번째로는 한 주기의 활성 구간의 개수와 순서를 정하는 방법을 소개하였으며, 망 분할의 감지를 위한 메시지 전송에 대해서 설명하였다. 그리고, 세번째로는 실제 망 분할 상황에서 사전에 정해진 방식대로 순환 리스를 적용하는 방법을 소개하였다.

기존의 연구에서는 망 고장에 의한 데이타의 일관성을 유지하는 데에 그치거나 망 분할된 서버들은 파일 시스템의 사용을 제한하는 정책을 사용하여 전체적으로는 파일 시스템의 가동이 중지 될 수 밖에 없었다. 하지만 순환 리스 기법을 사용할 경우 공유 저장 장치 파일 시스템에서는 제어망의 분할 고장에 대해서는 전체적 파일 시스템이 계속적으로 가동할 수 있게 된다. 전반적인 파일 시스템의 동작 속도는 평시와 동일한 수준에는 못 미치지만 사용자, 혹은 클라이언트의 입장에서는 파일 시스템 서비스가 중단되지 않는다는 점에서 의미를 가진다. 파일 시스템은 순환 리스 모드로 계속 동작을 하여 파일 시스템 관리자가 망 고장의 발생을 발견하고 조치를 취할 때까지 시간을 확보하여 주는 역할을 하게 된다. 순환 리스 기법은 또한 동작의 오버헤드 측면에서도 기존의 방법에 비해 뒤떨어지지 않는다. 다른 연구에서도 망분할의 감지를 위해서 주기적 메시지를 주고 받는 방법을 사용하는데 순환 리스에서도 동일한 방식을 사용하여 망 분할을 감지한다. 그리고, 주기와 활성 구간의 계산도 복잡도가 크지 않으므로 오버헤드가 되지 않는다.

순환 리스 기법이 현실적으로 사용되기 위해서는 다른 여러 가지 사항들이 고려되어야 한다. 순환 리스 기법은 서버들이 주어진 시간 구간 내에서 모든 작업을 정리하고 디스크에 반영할 수 있다는 가정 하에 논의되었다. 하지만, 운영체제의 특성, 혹은 다른 원인에 의해 서버들이 반드시 자신의 시간 구간 내에서 수정사항을 반영하지 못할 경우도 있다. 이와 같은 경우, 데이타의 일관성을 파괴하지 않도록 하는 방법의 제시가 필요하다. 또한, 사용자의 관점에서 보면 네트워크 분할이 일어나서 제안한 기법을 사용하여 동작할 경우, 서버의 반응시간(response time)이 불규칙적인 것을 느끼게 된다. 향후 연구에서는 이러한 불규칙성을 해소하고 사용자에게 고장 투명성(*fault transparency*)를 제공하기 위하여 화일 명령의 버퍼링과 같은 기법을 도입하는 것도 고려할 수 있다.

참 고 문 헌

[1] Steven R. Soltis, Thomas M. Ruwart, Matthew T.O'Keefe, The Global File System, Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, Sept 17-19, 1996.

[2] Matthew T. O'Keefe, Shared File Systems and Fibre Channel, Sixth NASA Goddard Space Flight Center Conference on Mass Storage and Technologies in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems March 23-26, 1998.

[3] R. C. Burns, R. M. Rees, and D. D. E. Long, Safe Caching in a Distributed File System for Network Attached Storage, In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2000.

[4] Chang-Soo Kim, gyoung-Bae Kim, Bum-Joo Shin, "Volume Management for SAN environment," In Proceedings of the International Conference on Parallel and Distributed Systems, 2001.

[5] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem," Proceedings of the Summer 1985 USENIX Conference, pages 119-130 June 1985.

[6] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and Michael J. West, "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems, 6(1), pages 51-81, February 1988.

[7] M. N. Nelson, B. B. Welch and J. K. Ousterhout, "Caching in the Sprite network file system," ACM Transactions on Computer Systems, 6(1), pages 134-154, February 1988.

[8] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli and R. Y. Wang, "Serverless network file systems," In Proceedings of the 15th Symposium on Operating Systems Principles, pages 109-126, December 1995.

[9] C.A. Thekkath, T. Mann, and E.K. Lee. "Frangipani: A Scalable Distributed File System," Pro-

ceedings of the ACM Symposium on Operating Systems Principles, pp. 224-237, Dec. 1997.
[10] R. C. Burns, R. M. Rees, and D. D. E. Long. An analytical study of opportunistic lease renewal. In Proceedings of the 16th International Conference on Distributed Computing Systems, 2001.
[11] R. C. Burns, R. M. Rees, and D. D. E. Long. Semi-Preemptible Locks for a Distributed File System. In Proceedings of the 2000 International Performance Computing and Communication Conference (IPCCC), IEEE, 2000.
[12] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. Proceedings of the Conference on File and Storage Technologies (FAST'02), pp. 231-244, 2002.
[13] CXFS: A high-performance, multi-OS SAN file system from SGI. SGI White Paper. URL http://www.sgi.com/products/storage/tech/file_systems.html.
[14] C. Gray and D. Cheriton, "Lease: An efficient fault-tolerant mechanism for distributed file cache consistency," Twelfth ACM Symposium on Operating Systems Principles, pp. 202 210, 1989.
[15] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Using leases to support server-driven consistency in large scale systems. In Proc. of the 18th Intl. Conf. on Distributed Computing Systems, May 1998.
[16] George Coulouris, Jean Dollimore, Tim Kindberg, Distributed Systems Concepts and Design Third edition, Addison-Wesley 2001.
[17] Michael R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Comleteness. W.H.Freeman, 1979.

탁 병 철

2000년 연세대학교 컴퓨터과학과 학사 2003년 한국과학기술원 전산학과 석사 2003년 ㈜디지털아리아 선임연구원. 2004년~현재 한국전자통신연구원 임베디드 S/W연구단. 관심분야는 파일 시스템, 데이타마이닝, 임베디드 시스템

정 연 돈

정보과학회논문지 : 데이타베이스
제 32 권 제 4 호 참조

김 명 호

정보과학회논문지 : 데이타베이스
제 32 권 제 4 호 참조