제 4 장 Coda 분산 파일 시스템

다양한 인터넷 응용 프로그램들이 웹기반으로 통합화되고 인터넷 사용자가 급속히 증가와 더불어, 대용량/고성능의 컴퓨팅 파워를 필요로 하는 서비스의 증가로 인해 고가의 기존 서버들이 가지는 고비용, 확장성 및 호환성에 대한 문제들을 극복할 수 있는 리눅스 기반의 클러스터 서버에 대한 관심이 증폭되고 있다. 이러한 리눅스 기반의 클러스터 환경에서 동작하는 웹 서버나 대용량의 멀티미디어 서버, 자료 검색 서버 등은 대용량의 파일 입출력의 실시간 처리뿐만 아니라 서비스에 대한 고가용성을 만족시킬 수 있어야 하고, 각 서버는 자신의 로컬 디스크뿐만 아니라 네트워크로 연결된 하나의 가상 서버내의 다른 컴퓨터들에 저장되어 있는 데이터들에 대해서도 효과적으로 대처 할 수 있어야 하기 때문에 이러한 클러스터 환경에 적합한 분산 병렬 파일 시스템이 필수적이라 할 수 있다[3,18].

이번 장에서는 Coda 고장 포용(Fault Tolerance) 네트워크 분산 파일 시스템에 대해서 그 기본 개념, 구성요소, 동작 원리 및 Coda 시스템이 리눅스 클러스터 환경에서 어떠한 방식으로 고가용성을 보장하는지에 대해서 알아보도록 하겠다. 그리고 Coda 시스템의 여러 가지 응용분야와 Coda를 실제 리눅스 클러스터 웹 서버에 적용한 결과에 대해서는 5장에서 살펴보도록 하겠다[15,19].

4.1 Coda 분산 파일 시스템 개요

Coda 분산 파일 시스템(Distributed File System : DFS)은 Carnegie Mellon 대학의 M. Satyanarayanan 팀에 의해 개발되고 있는 Advanced Networked File system으로 AFS (Andrew File System)를 기반으로 하고 있다. Coda는 최초 Mach 커널 위에서 개발된 것으로, 현재 Linux 뿐만 아니라 FreeBSD, NetBSD, Windows 95/98, Windows NT 등 다양한 platform 에서도 동작될 수 있도록 이식되었으며, 각각의 platform에서 독립적으로 동작하기 때문에 Linux 환경에서의 Coda 서버들과 Windows NT 기반에서의 Coda 서버들, Windows 95/98 기반에서

의 Coda 클라이언트 등, 다양한 platform에서의 Coda 서버와 클라이언트 들이 상호 연동되어 하나의 분산 파일 시스템을 이루어 동작될 수 있어서 확장성 및 호환성에서 특이할만한 강점을 지니고 있다[19-20].

Coda 분산 파일 시스템은 실제로 파일이 저장되는 하나 또는 그 이상의 Coda 서버와, Coda 서버에 저장된 파일에 접근하여 읽기, 쓰기, 실행 등의 작업을 수행할 수 있는 Coda 클라이언트로 구분할 수 있으며, 이들 서버와 클라이언트는 사용자의 요구에 따라 다양한 방법으로 설치되어 사용할 수 있다. 유닉스 환경에서의 대표적인 분산 파일 시스템인 NFS는 개개의 서버가 파일시스템을 export하고 클라이언트가 그것을 마운트해서 사용하게 되는데 반해서, Coda는 개개의 서버를 따로 의식하지 않고, Coda 파일시스템을 클라이언트에서 접근하면 전체 Coda 파일시스템 서버를 하나로 보이게 한다. 클라이언트의 입장에서 Coda 파일은 /coda 디렉토리에 저장되고, 각 파일이 어떤 특정 서버에 있는지에 대한 정보 없이도 각 파일을 접근할 수 있다. 이렇게 서버마다 각각 하나의 디렉토리를 마운트하지 않고 전체 분산 파일 시스템을 하나의 디렉토리로 마운트하는 방식은 Coda의 선배격인 AFS (Andrew File System)에서 시작되었다. 하나의 디렉토리에 전체 분산 파일 시스템을 마운트하게 되면 각 클라이언트가 보는 파일 이름이 모두 동일하게 된다. 따라서 모든 사용자가 보는 파일 트리가 같은 구조를 가지게 된다[3,21].

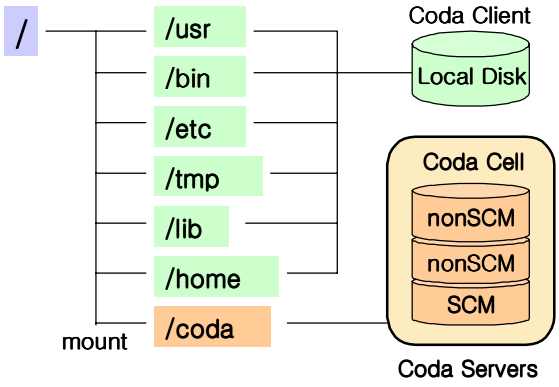

그림 4.1 Coda 클라이언트의 서버 영역 마운트
Fig. 4.1 Mount Coda Server's Area of Coda Client

Coda 시스템은 서버와의 연결이 단절된 상태에서도 클라이언트 내의 캐시 매니저인 Venus에 의해 기존의 작업을 계속 수행할 수 있다. 의도적이든지 그렇지 않던 간에, 서버와의 연결이 끊어진 상태에서 변경되거나 추가된 작업에 대해서는 추후 서버와의 연결이 복원되면 클라이언트에서 변경된 내용들은 자동으로 서버에 업데이트된다. 또, 임의의 서버내의 데이터에 대해서 Coda 서버 그룹내의 다른 서버들에 복사본을 저장해두었다가 그 서버에서 고장이 발생하여 정상적인 동작을 할 수 없을 경우에도 계속해서 나머지 서버들이 서비스를 계속할 수 있다. 마찬가지로 고장이 발생했던 서버가 복원되면 그 동안 변경된 사항에 대해서는 자동으로 해당 서버로부터 자동으로 업데이트 될 수 있다. 이러한 방식으로 Coda 분산 파일 시스템은 서비스의 고가용성을 보장하게 되는 것이다.

4.2 Coda 분산 파일 시스템의 주요 구성요소

다음은 Coda 시스템을 구성하는 중요 요소들에 대해서 설명하도록 하겠다[22].

- **A single name space** : Coda 시스템의 모든 정보는 /coda 라는 하나의 디렉토리에서 나타난다. Coda는 유닉스 환경에서의 대표적인 파일 시스템인 NFS나, 이기종 시스템간의 디스크 공유를 위한 Samba 처럼 export 및 share 의 과정을 거쳐 개별적으로 마운트되는 것과는 다르게 모든 서버들에 의해 파일들의 Volume이 /coda 디렉토리에서 export 되어 모든 클라이언트들은 Coda의 Root_volume에 대한 정보를 이용하여 자동적으로 서버들을 찾을 수 있다.
- **Coda cell** : Coda cell은 각종 설정과 DB를 공유하며 서비스를 하는 서버들의 집합이다. 하나의 cell은 단 하나의 서버로 구성될 수도 있고, 몇 백대의 서버들로 구성될 수도 있다. 단, Coda cell 내의 서버들 중에서 반드시 한 대는 SCM(System Control Machine)으로 설정되어야 한다. SCM은 해당 cell 내의 모든 서버들에 의해 공유되고 있는 설정이나 데이터베이스에 대해서 수정을 할 수 있는 권한을 가지며, 기타 변경사항을 cell 내의 다른 서버들에게 전송하는 역할을 한다. 현재 하나의 Coda 클라이언트는 하나의

cell에만 속할 수 있다. 현재 다중 셀에 대해서도 효과적으로 동작할 수 있는 Coda 시스템에 대해서 연구개발이 진행되고 있다.

- **Coda volumes** : 파일서버들은 Volume을 이용하여 파일들을 관리한다. 하나의 Volume은 일반적으로 하나의 파티션보다는 작고 하나의 디렉토리보다는 크다. Volume은 파일들의 디렉토리 구조를 포함하며 각각의 Volume은 클라이언트의 /coda 디렉토리로 마운트되고 그 아래에서 하부구조를 형성한다. 또한 volume은 다른 volume들에 대한 마운트 포인트를 포함할 수 있다. Coda에서의 마운트 포인트는 Unix나 Windows의 마운트 포인트와는 다르다. Coda 마운트 포인트는 그 volume안에 파일을 저장하는 서버를 찾기 위해서 클라이언트에 대한 충분한 정보를 포함한다. Volume을 서비스하는 서버들의 집합을 VSG(Volume Storage Group) 이라고 한다.
- **Volume Mountpoints** : Coda에서는 개개의 서버를 /coda 로 마운트하는 하나의 특별한 Root_volume이 있다. 나머지 volume들은 Coda 디렉토리 구조하에서 volume mountpoint를 설정하는 명령인 cfs mkmount를 이용하여 /coda 디렉토리 안에서 하부적으로 추가된다. 예를 들어, Root_volume외에 또다른 볼륨인 vol_ex_1과 vol_ex_2가 있다고 하면, 이 두 볼륨은 cfs mkm 명령에 의해서 /coda/vol_ex_1, /coda/vol_ex_2 와 같이 /coda 디렉토리에 하부적으로 추가된다. Coda mountpoint는 영구적인 객체이다. 시스템을 재시작한 후에 원상태로 복구 하여야 하는 Unix mountpoint와는 다르다.
- **Data storage** : Coda 서버들은 volume들을 로컬 디스크 파일시스템 안에 있는 일반적인 디렉토리에 저장하지 않는다. Coda는 서버 Replication이나 Disconnected operation과 같은 기능을 수행하기 위해서는 보다 많은 메타 데이터가 필요하고, 이러한 메타 데이터가 로컬 디스크 파일시스템 안에 존재하게 되면 복구가 어렵게 된다. 따라서, Coda 서버들은 /vicepa, /vicepb 등과 같은 데이터 저장 파티션을 할당받아 이곳에 저장하고자 하는 volume들을 숫자에 의해 구분하여 저장한다. 또, 파일 소유권, ACL(Access File Lists), version vector등과 같은 메타 데이터들은 ext2 파일 시스템과는 다른 형태로 변형된 새로운 RVM 데이터 파티션에 저장된다.

- **RVM (Recoverable Virtual Memory)** : 서버들이 재 시작되는 경우와 같이 서비스의 불연속적인 상태에 있어서 데이터의 손실과 오류를 방지하기 위해서 Coda에서 사용하는 것으로, Coda 서버 메타 데이터의 시스템 상태를 유지하는 트랜잭션 시스템이다. RVM은 각종 수정사항을 RVM log에 기록하고 재 시작시 그 수정사항을 RVM 데이터 파일에 함께 추가하는 방식의 'data logging transaction system'이다. 시스템 시작시 Coda 서버들은 Coda 시스템의 상태를 확인하고 복구하기 위해서 RVM을 사용한다. 실제 Coda 서버 구현시에는 최적의 성능을 위해서 각각의 RVM log & data 영역을 위한 새로운 파티션의 분할이 필요하다. 특히 RVM 메타 데이터를 위한 저장공간으로는 실제 Coda 데이터 저장 공간의 4% 정도로 설정하는 것이 적당하다.
- **Client data** : 클라이언트에서의 데이터도 서버와 비슷한 방식으로 저장된다. RVM내의 메타 데이터와 캐시된 파일들은 /usr/coda/venus.cache 디렉토리에 숫자들로 구분하여 저장한다. 서버들에 존재하는 파일들의 사본인 클라이언트의 캐시 데이터들은 매번 서버와의 접속을 시도하지 않고도 서버에 있는 데이터를 엑세스할 수 있으므로 그 속도가 빠르고 또한 서버와 연결이 단절된 상태에서도 효과적으로 작업을 지속할 수 있다.

4.3 Coda 클라이언트 - Venus

Coda 클라이언트는 Coda 서버의 데이터 저장 공간을 디바이스 /dev/cfs0를 이용하여 자신의 Local 디렉토리에 /coda 라는 디렉토리로 마운트하여 사용하기 때문에 전체 Coda 파일 시스템 서버를 하나 저장 공간으로 이용할 수 있다. 클라이언트의 입장에서 Coda 파일은 각각의 volume으로 /coda 디렉토리에 하부적으로 저장되고, 각각의 Coda 파일이 실제로 어떤 Coda 서버에 저장되어 있는지에 대한 정보 없이도 각 파일에 접근할 수 있다. 이와 같은 하나의 마운트 포인트을 이용함으로서 모든 클라이언트들이 동일한 설정에 의해서 동작할 수 있고 사용자는 언제나 동일한 디렉토리 구조를 볼 수 있다는 장점이 있다.

클라이언트에서 임의의 프로그램을 통해 /coda 디렉토리의 Coda 파일을 열기 위해 커널에 시스템 콜을 요청하게 되면 파일 안에 데이터를 액세스하기 위해서 필요한 정보를 담고 있는 inode가 커널에 의해서 사용되고, 커널 내의 VFS (Virtual File System)는 해당 디렉토리의 파일 시스템을 호출하게 된다. 만약 요청된 파일이 /coda 디렉토리에 있는 Coda 파일이라면 VFS는 커널 내의 Coda 파일 시스템 모듈을 이용하여 Coda 파일 시스템에 대해서 서비스를 요청하게 된다. 서비스를 요청 받은 커널 내의 Coda 모듈은 Coda 클라이언트의 캐시 관리자 (cache manager)인 Venus를 호출하게 된다. **Venus**는 클라이언트의 로컬 디스크 캐시에 요청된 파일이 있는지를 먼저 확인하고 해당 파일이 캐시 내에 존재하면 파일에 대한 서비스를 개시하고, 캐시 내에 없다면 RPC2 (Remote Procedure Calls)를 이용하여 해당 서버의 Vice에 접속하여 해당 파일을 읽어오게 된다. 일단 Coda 서버에 접속하면 읽기 전용 권한으로 접속하게 된다. 추가적으로 쓰기 권한을 얻고자 한다면 'clog' 명령을 이용하여 서버에 의해 인증을 받아야 한다. Coda 파일들에 대해서 쓰기 권한을 가질 수 있는 계정은 시스템의 일반적인 사용자 계정일수는 있으나 시스템 관리자인 'root'로는 쓰기 권한을 획득할 수 없다.

이렇게 한번 읽혀진 파일에 대해서는 클라이언트 Coda의 캐시 매니저인 Venus에 의해서 파일의 모든 속성들(소유권, 허가권, 크기, 등등)과 함께 캐시 디렉토리인 /usr/coda/venus.cache 디렉토리에 저장된다. 이제 이 캐시 디렉토리에 저장된 파일은 실제 클라이언트의 로컬 디스크에 존재하는 일반적인 파일로서 존재하게 되어 나중에 동일 파일에 대해서 서비스 요청을 받게 되면 클라이언트는 Vice에 접속하지 않고도 로컬 파일 시스템인 EXT2 에 의해서 읽기, 쓰기 작업등이 수행할 수 있게 된다[24]. 따라서 모든 작업은 자신의 로컬 디스크에서 수행하는 여타 작업과 동일한 속도로 수행될 수 있다. 이제 클라이언트에서 작업 중이던 파일에 대해서 수정이 되거나 작업이 끝나게 되면 Venus는 변경된 파일을 서버로 전송하여 변경된 내용을 즉시 업데이트한다. 파일에 대한 수정작업 뿐만 아니라 Coda 파일 시스템 내의 디렉토리의 추가/삭제, 링크의 추가/삭제 등, 기타 변경사항들도 바로 서버로 전송되어 업데이트된다. 그림 에서 Coda 시스템의 동작을 개략적으로 나타내었다.

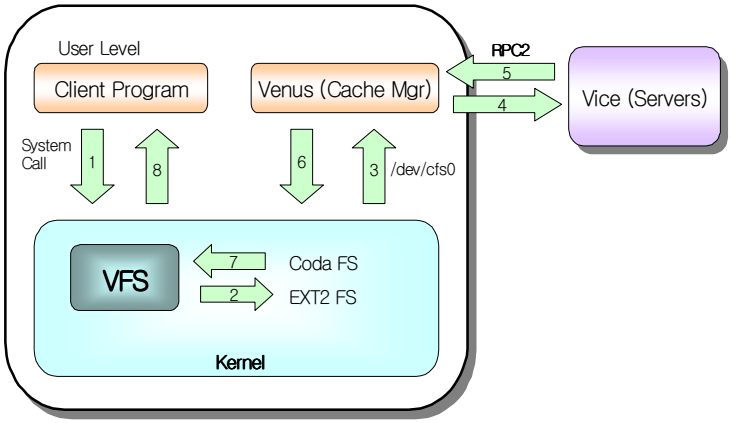

그림 4.2 Coda 파일 시스템의 파일 액세스 과정
Fig. 4.2 File Access Process of Coda File System

앞서 말한 Coda 클라이언트의 캐시 기능은 서버의 오류나 네트워크 단절로 인한 고장 발생 상황에 대해서도 효과적으로 이용된다. 앞서 말했듯이, /coda 디렉토리의 coda 파일이 처음으로 요청되게 되면 그 파일의 모든 속성뿐만 아니라 그 파일을 액세스하는데 필요한 모든 정보가 캐시에 저장되게 되고, 파일이 변경되었거나 작업이 종료되었다면 모든 변경사항은 서버로 전송되어 서버의 내용도 바로 업데이트되어야 한다. 그러나 네트워크 단절이나 서버에서 발생한 고장으로 인하여 타임아웃 등과 같은 에러메시지를 받게 되면, Venus는 그 에러를 사용자에게 보여주지 않고 서버와 연결이 될 때까지 모든 업데이트 내용에 대해서 CML (Client Modification Log)에 기록해 두고 있다가 이후에 서버와의 연결이 복원되면 CML에 기록되어 있는 내용을 이용하여 바로 서버를 업데이트한다. 서버의 업데이트가 모두 끝나면 CML의 내용은 삭제되어 최적의 상태를 유지하게 된다. 이러한 모든 과정은 Venus가 스스로 알아서 처리하기 때문에 사용자는 현재 서버와의 네트워크 단절에 대해서 크게 신경 쓸 필요가 없다. 이러한 Venus의 캐시 기능은 서버 고장 등에 의한 네트워크 단절시에 유용할 뿐만 아니라 사용자가 임의로 Coda 클라이언트를 서버로부터 분리하여 임의의 장소에서 작업한 후 이후 서버에 연결하여 자동으로 업데이트할 수 있으므로 이를 이용한 이동 컴퓨팅(Mobile Computing)분야에서도 활발한 연구가 진행되고 있다[19-23].

Coda는 캐시 매니저인 Venus를 이용하여 중요한 파일에 대해서는 캐시에 그 파일을 계속 저장해 놓을 수도 있다. 일반적으로 지정된 캐시 용량이 넘게 되면 예전의 파일은 삭제되고 최근에 액세스 된 파일이 캐시에 저장되는데 사용자가 중요한 파일에 해해서 우선권을 부여하여 캐시에 해당 파일을 계속 유지하도록 하는 것이다. 이 방법은 hoarding 이라고 알려져 있으며 Venus는 내부적으로 이러한 파일들을 hoard database로 유지할 수 있다.

4.4 Coda 서버 - Vice

Coda 서버에서의 파일들은 일반적인 Unix 파일 시스템과는 다른 방식으로 저장된다. 실제로 리눅스 머신을 Coda 서버로 동작시키기 위해서는 리눅스 설치시부터 파일서버로 설치하기 위한 파티션 작업이 필요하다. 최적의 성능을 내기 위해서 RVM Log 파티션과 RVM Data 파티션은 따로 두어야 하며, Coda 서버에 대한 각종 설정이 기록되는 /vice 디렉토리와 시스템의 log 파일이 기록되는 /var 파티션, 실제의 Coda 파일들이 저장되는 /vicepa, /vicepb, /vicepc 등의 파티션도 설치시 파티션 작업을 통하여 설치하게 되면 로컬 디스크 내에서의 해당 디렉토리의 위치 검색시간을 단축 할 수 있어서 보다 효율적인 서버로의 운영이 가능하다. 이러한 여러 가지 파티션 중에서 실제로 데이터를 저장하는 /vicepa, /vicepb, /vicepc 등의 파티션은 파일의 논리적인 기본 단위인 volume 단위로 그룹을 지어서 Coda 파일들을 관리한다. 각각의 volume들은 일반적인 파일 시스템의 트리 구조와 같이 전형적인 디렉토리 구조를 가진다. Coda 시스템에서의 volume 하나의 크기는 일반적으로 데이터 저장 파티션의 크기보다는 작고, 하나의 디렉토리보다는 클 수 있으며 보통 하나의 서버는 수 백 개의 volume을 소유하여 관리한다.

Volume은 고유한 이름과 ID를 가지며, 자신만의 루트 디렉토리와 그에 따르는 하위 디렉토리 정보 및 파일을 소유한다. 또한 volume은 /coda 디렉토리 아래라면 어느 곳이라도 마운트 프로세스에 의해 새로운 디렉토리를 만들어 마운트 될 수 있지만, 기존에 존재하고 있는 디렉토리로는 마운트될 수 없다. 이러한 과정은 Macintosh나 Windows의 "Network Drives and Volumes"과 비슷하나, 클라이언

트 쪽에서는 이러한 마운트 포인트가 보이지 않는다는 점에서 이들과 큰 차이점을 가진다. 클라이언트 쪽에서는 Coda 서버내의 파일들이 /coda 디렉토리의 일반적인 파일들로만 보일 뿐이다.

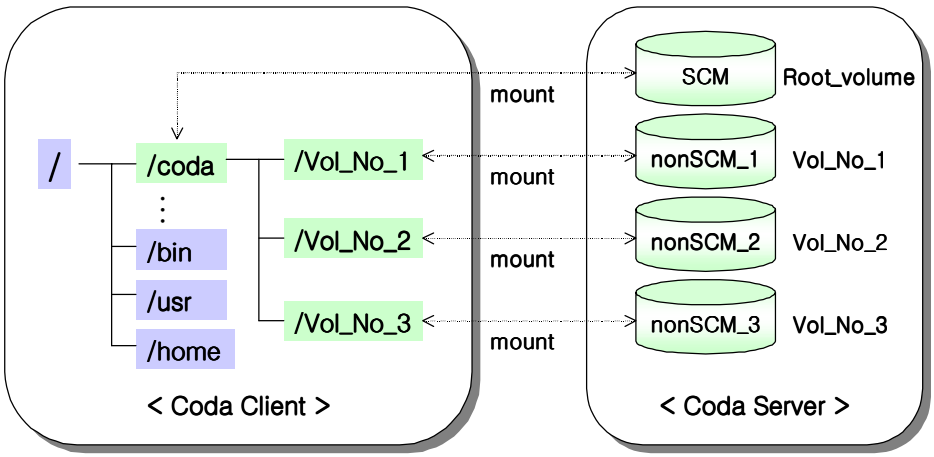

그림 4.3 Coda 클라이언트의 볼륨 마운트
Fig. 4.3 Volume Mount of Coda Client

Coda는 volume과 디렉토리 정보, ACL(Access Control Lists), 각종 파일 속성들을 일반적인 EXT2 파일 시스템과는 다른 RVM(Recoverable Virtual Memory) 파티션에 저장되어 사용한다. 이러한 메타 데이터들은 파일 액세스에 있어서 필수적이기 때문에 RVM를 통하여 이용되어짐으로서 속도와 연속성 면에서 강점을 지니게 된다. RVM은 시스템 상태를 유지하기 위한 트랜잭션 시스템으로서 서버에서 고장이 발생하거나 동작에 문제가 있을 경우에도 연속적인 상태를 유지하고 복구할 수 있는 기능을 가지고 있다. Coda는 세 개의 32비트 정수로 된 Fid를 이용하여 각각의 파일들을 구분한다. Fid는 파일이 위치하고 있는 volume을 나타내는 VolumeId, 파일의 inode 번호를 나타내는 VnodeId, 그리고 어느 것이 보다 최근의 파일인지를 구별해내기 위한 Uniquifier로 구성된다. 이러한 Fid는 전체 Coda 서버의 클러스터 내에서 유일하게 존재한다[19].

Coda는 임의의 서버에 대하여 똑같은 복제 서버들을 만들어 운영할 수 있다.(Replication) 여기에 사용되는 volume들은 복사된 volume들을 소유하고 있는

VSG(Volume Storage Group)이라는 서버들에 의해 저장될 수 있다. 이러한 기능은 VSG내의 임의의 서버가 고장에 의해 서비스를 할 수 없을지라도 다른 서버들이 그 작업을 이어받아 지속적인 서비스를 유지함으로서 파일에 대한 고가용성을 높일 수 있다는 강점을 가진다. VSG는 클라이언트들에게 파일 데이터를 분배할 수 있고, VSG 내의 모든 서버들은 하나의 변경사항에 대해서 모두 동시에 업데이트된다[19,25,26].

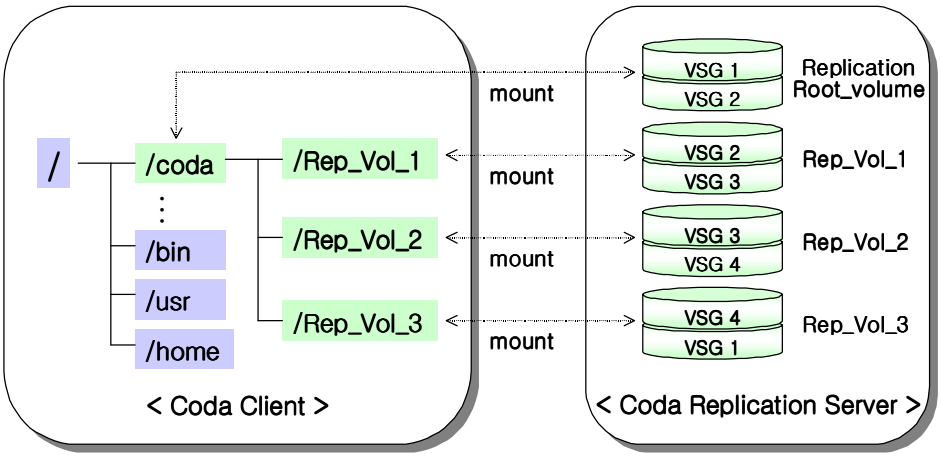

그림 4.4 Coda 시스템의 복제 볼륨 마운트
Fig. 4.4 Replication Volume Mount of Coda System

Venus가 VSG의 서버들의 파일을 액세스하고자 한다면 Venus는 먼저 파일을 포함하고 있는 volume에 대해서 VolumeInfo를 먼저 검색한다. 이 정보는 서버들의 목록과 각각의 서버의 로컬 volume Id를 포함한다. 임의의 파일에 대한 VSG 내 서버들간의 통신에 있어서는 클라이언트로의 파일 전송은 하나의 서버에서만 일어나지만, 변경된 정보에 대한 업데이트는 가용한 모든 서버(AVSG : Available VSG)에서 함께 일어나기 때문에 읽기 작업 보다 쓰기 작업이 보다 빈번하게 일어난다. Coda에서는 동시 공동 이용의 RPC을 이용함으로서 이러한 경우의 성능 저하를 방지한다[19,27].

서버 복제에 있어서는, 네트워크 단절시의 동작과 같이, 보다 최근의 파일을 가려내는 Resolutin 과 Repair 기능이 중요하게 동작한다. 네트워크나 서버의 고장으

로 인해 VSG내의 임의의 서버들이 단절되었을 경우에, 업데이트는 모든 서버들에서 일어날 수 없고 단지 AVSG에 속하는 서버들에서만 일어날 수 있다. 이후에 모든 서버들의 연결이 복원되면 각각의 서버들에 대해서 최신으로 업데이트된 파일들에 대해서 검색을 하여 파일 정보에 차이가 있을 경우에는 자동으로 모든 서버들의 내용이 최신의 내용으로 업데이트 된다. 이렇게 Coda의 복제 서버 기능은 임의의 서버의 고장에 대해서도 유연하게 대처함으로서 서비스의 고가용성을 높여 준다.