웹 전용 리눅스 클러스터 서버의 고가용성을 위한 분산 파일 시스템에 대한 연구

박 지 현*, 류 상 우*, 장 휘**, 김 학 배*
연세대학교 전기 · 컴퓨터공학과*, C-EISA**

A Distributed File System for Guaranteeing High Availability of a Clustering Web Server

Jihyun Park*, Sangwoo Ryu*, Whie Chang**, Hagbae Kim*
Dept. of Electrical and Computer Eng., Yonsei Univ.*, C-EISA**

Abstract - 다양한 인터넷 응용 프로그램들이 웹 기반으로 통합되고 여러 방면에서 business-critical한 경우가 많아짐에 따라 웹서버의 고가용성과 안정성이 갈수록 강조되고 있고, 이를 보장하기 위한 리눅스 기반의 클러스터링 환경에서는 다양한 조건하에서도 데이터의 손실 없이 파일 입출력을 효과적으로 지원할 수 있는 분산 파일 시스템이 필수적이다. 본 논문에서는 리눅스 클러스터 환경에 적합한 분산 파일 시스템의 하나로서 카네기 멜론 대학에서 제안되어 개발 중에 있는 네트워크 분산 파일 시스템인 Coda 에 대하여, 가용성 및 효율성, 확장성 등에 대한 장단점을 간단히 소개하고, Coda을 적용한 고가용성 웹 서버의 구현 결과와 향후 개선 방향에 대해서 설명하도록 하겠다.

1. 서 론

다양한 인터넷 응용 프로그램들이 웹기반으로 통합화되고 인터넷 사용자가 급속히 증가하면서, 기존의 파일 서버 및 프린터 서버뿐만 아니라 웹서버와 애플리케이션 서버, 멀티미디어 서비스를 위한 VOD/AOD 서버, 인터넷 기반 사업용 서버 등으로 서버의 영역이 점차 확대되어 나가고 있다. 이러한 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[4,5,7,8].

이러한 리눅스 기반의 클러스터 환경에서 동작하는 웹서버나 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[3,5].

본 논문에서는 Coda 분산 파일 시스템에 대해서 그 기본 개념과 동작 원리 및 Coda 시스템이 리눅스 클러스터 환경에서 어떠한 방식으로 고가용성을 보장하는지에 대해서 알아보고, Coda 시스템의 여러 가지 응용분야와 Coda를 실제 리눅스 클러스터 웹 서버에 적용한 결과에 대해서 살펴보도록 하겠다.

2. 본 론

2.1 Coda 분산 파일 시스템

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 platform에서도 동작될 수 있도록 이식되었으며, 각각의 platform에서 독립적으로 동작하기 때문에 다양한 platform에서의 Coda 서버와 클라이언트들이 상호 연동되어 하나의 분산 파일 시스템을 이룰 수 있어서 확장성 및 호환성에서 특이할만한 강점을 지니고 있다[1,6].

Coda 분산 파일 시스템은 실제로 파일이 저장되는 Coda 서버와, Coda 서버에 저장된 파일에 접근하여 읽기, 쓰기, 실행 등의 작업을 수행할 수 있는 Coda 클라이언트로 구분할 수 있다.[2]

2.1.1 Coda 시스템의 주요 구성요소

다음은 Coda 시스템을 구성하는 중요 요소들이다.

• **Coda cell** : Coda cell은 한 마디로 말해서 각종 설정과 DB를 공유하며 서비스를 하는 서버들의 집합이다. 하나의 cell은 단 하나의 서버로 구성될 수도 있고, 몇 백대의 서버들로 구성될 수도 있으나, Coda cell 내의 서버들 중에서 반드시 한 대는 **SCM** (System Control Machine)으로 설정되어야 한다. SCM은 해당 cell 내의 모든 서버들의 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.

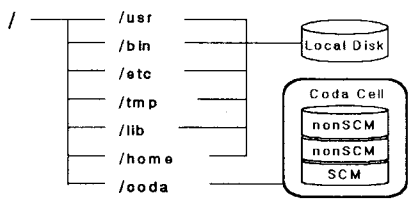

그림 1 Coda 파일 시스템

• **Coda volumes** : 파일 서버들은 볼륨을 이용하여 파일들을 관리한다. 하나의 볼륨은 일반적으로 하나의 파티션보다는 작고 하나의 디렉토리보다는 크다. 각각의 볼륨은 파일들의 디렉토리 구조를 포함하며 클라이언트의 /coda 디렉토리로 마운트되고 그 아래에서 하부구조를 형성한다. 볼륨을 서비스하고 저장하는 서버들의 집합을 **VSG**(Volume Storage Group) 이라고 한다.

• **Volume Mountpoints** : Coda에서는 개개의 서버를 /coda로 마운트하는 하나의 특별한 root 볼륨이 있다. 나머지 볼륨들은 Coda 디렉토리 구조하에서 /coda 디렉토리 안에서 하부적으로 추가된다.

• **RVM** (Recoverable Virtual Memory) : 서버들이 재 시작되는 경우와 같은 서비스의 불연속적인 상태에 있어서 데이터의 손실과 오류를 방지하기 위해서 Coda에서 사용하는 것으로, Coda 서버 메타 데이터와

본 연구는 2000년 정보통신연구진흥원 산업기술개발사업의 지원 하에 진행되었습니다.

시스템 상태를 RVM log에 기록하고 재 시작시 그 수정 사항을 RVM 데이터 파일에 함께 추가하는 방식의 'data logging transaction system'이다. 시스템 시작시 Coda 서버들은 시스템의 상태를 확인하고 복구하기 위해서 RVM을 사용한다.

2.2 Coda 클라이언트 - Venus

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

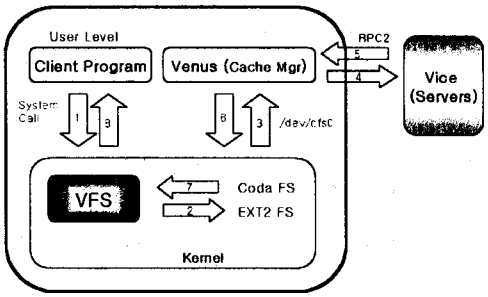

그림 2 Coda 시스템의 동작 과정

그림 2.에서 Coda 시스템의 동작을 개략적으로 나타내었다. 클라이언트에서 임의의 프로그램이 /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 **Venus**를 호출하게 된다. Venus는 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 **Vice**에 접속하여 해당 파일을 읽어오게 된다. 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 서버에 접속하지 않고도 로컬 파일 시스템인 EXT2에 의해서 읽기, 쓰기 작업등이 수행할 수 있게 된다. 따라서 모든 작업은 자신의 로컬 디스크에서 수행하는 여타 작업과 동일한 속도로 수행될 수 있어서 속도의 향상을 기대할 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 등, 기타 변경 사항들도 바로 서버로 전송되어 업데이트된다. 앞서 말한 Coda 클라이언트의 캐시 기능은 서버의 오류나 네트워크 단절로 인한 고장 발생 상황에 대해서도 효과적으로 이용된다. 서버에서 액세스된 파일이 변경되거나 종료되면 모든 변경 사항은 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 내용을 **CML** (Client Modification Log)에 기록해 두고 있다가 이후에 서버와의 연결이 복원되면 CML에 기록되어 있는 내용을 이용하여 바로 서버를 업데이트한다. 이러한 Venus의 캐시 기능은 서버의 고장이나 네트워크 단절시에 유용할 뿐만 아니라 사용자가 임의로 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[1,6].

2.3 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 파티션과 시스템의 log 파일이 기록되는 /var 파티션, 실제의 Coda 파일들이 저장되는 /vicepa, /vicepb 등의 파티션도 설치시 파티션 작업을 통하여 따로 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 등의 파티션은 파일의 논리적인 기본 단위인 볼륨 단위로 그룹을 지어서 Coda 파일들을 관리한다. Coda 시스템에서의 하나의 볼륨 크기는 일반적으로 10MB 내외이며, 하나의 서버는 수 백 개의 볼륨을 소유하여 관리하게 된다. 볼륨은 고유한 이름과 ID를 가지며, 자신만의 루트 디렉토리와 그에 따르는 하위 디렉토리 정보 및 파일을 소유하고, /coda 디렉토리 아래라면 어느 곳이라도 마운트 프로세스에 의해 새로운 디렉토리를 만들어 마운트될 수 있지만, 기존에 존재하고 있는 디렉토리로는 마운트될 수 없다. 이러한 과정은 Macintosh나 Windows의 "Network Drives and Volumes"과 비슷하나, 클라이언트 쪽에서는 이러한 마운트 포인트가 보이지 않는다는 점에서 이들과 차이점을 가진다. 클라이언트 쪽에서는 Coda 서버내의 파일들이 /coda 디렉토리의 일반적인 파일들로만 보일 뿐이다.

Coda는 볼륨과 디렉토리 정보, ACL(Access Control Lists), 각종 파일 속성등에 대해서 일반적인 EXT2 파일 시스템과는 다른 RVM(Recoverable Virtual Memory) 파티션에 저장하여 사용한다. 파일 액세스에 필수적인 이러한 메타 데이터들은 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 **Fid**를 이용하여 각각의 파일들을 구분한다. Fid는 파일이 위치하고 있는 볼륨을 나타내는 VolumeId, 파일의 inode 번호를 나타내는 VnodeId, 그리고 어느 것이 보다 최근의 파일인지를 구별해내기 위한 Uniquifier로 구성된다.

Coda는 임의의 서버에 대하여 복제 서버들을 만들어

운영할 수 있다-**Replication**. 여기에 사용되는 볼륨들은 복사된 볼륨들을 소유하고 있는 **VSG**(Volume Storage Group)이라는 서버들에 의해 저장되고 VSG 내의 모든 서버들은 하나의 변경사항에 대해서 모두 동시에 업데이트된다. 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. 서버 복제에 있어서는, 네트워크 단절시의 동작과 같이, 보다 최근의 파일을 가려내는 Resolution 과 Repair 기능이 중요하게 동작한다. 네트워크나 서버의 고장으로 인해 VSG내의 임의의 서버들이 단절되었을 경우에, 업데이트는 모든 서버들에서 일어날 수 없고 단지 가용한 모든 서버(**AVSG** : Available VSG)들에서만 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.

2.4 Coda 가 적용된 리눅스 클러스터 웹 서버

Coda 시스템의 주된 응용분야로는 여러 대의 클라이언트를 통한 프로그램의 공유, FTP 미러 사이트, WWW 복제 서버, 네트워크 컴퓨팅 등 다양한 분야에서 응용되고 있다.[1,6]

Coda 클라이언트의 캐시 기능과 Coda 서버의 Replication 기능은 접속이 많은 웹 서버에 대한 복제 서버 운영에 이용할 수 있으며, 특히 리눅스 클러스터 환경에서의 고가용성을 보장하기 위한 웹서버에 대하여 유용하게 이용될 수 있다.

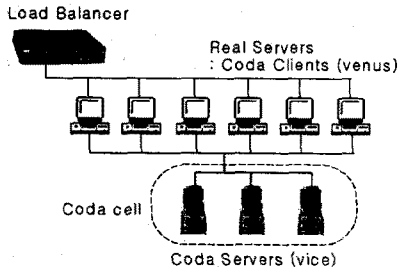

그림 3 웹서버와 분리된 Coda 서버

그림 3.은 리눅스 클러스터링 웹서버 모델에 별개의 Coda 파일 서버을 적용한 예이다. 리눅스 클러스터 내에서 웹 서비스를 제공하는 각각의 Real 서버들이 모두 Coda 클라이언트로 동작할 경우, 클라이언트들은 /coda 디렉토리에 웹 문서들을 저장할 수 있다. 이러한 경우, 임의의 서버에서 파일의 변경사항이 생기면, 모든 Real 서버들은 별도의 업데이트 없이도 변경된 웹 문서를 서비스할 수 있다. 또한, 서버의 Replication 기능을 이용하여 여러 대의 Coda 서버에 데이터를 함께 저장한다면 임의의 서버에 고장이 발생하여 파일 서비스를 할 수 없다 하더라도 VSG내의 다른 서버가 대신 파일 서비스를 할 수 있으므로 시스템의 가용성을 높일 수 있다. 또한, 클라이언트의 Venus에 의해 동작하는 캐시 정책은 Coda 서버와의 통신을 줄임으로서 네트워크 부하량을 감소시킬 수 있어서 전체적인 네트워크 속도의 향상도 기대할 수 있다.

그림 4.는 웹서버와 Coda 서버를 분리하지 않고 클러스터내의 모든 Real 서버들을 Coda 서버와 Coda 클라이언트로 함께 동작시키는 모델을 나타내고 있다. 이러한 방법은 추가적인 파일 서버를 설치하지 않고 Real 서버들의 저장공간을 이용하여 대용량의 파일 서비스를

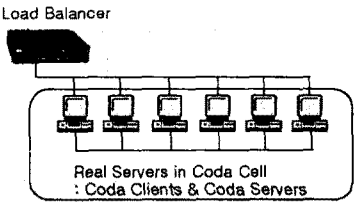

그림 4 웹서버에서 동작하는 Coda 서버

제공할 수 있으므로 클러스터 시스템 구현시 비용 절감의 효과를 볼 수 있고, 각각의 Real 서버들은 Coda Cell내의 Replication 서버로 동작하게 할 수 있으므로 Real 서버의 고장에 대해서도 고가용성을 보장할 수 있다는 강점을 가진다. 그러나, 위와 같이 구성된 서버들은 클라이언트만 동작하는 일반적인 웹서버에 비해서 성능과 속도가 저하될 우려가 있다. Coda 서버가 정상적으로 동작하기 위해서는 64MB 이상의 가상 메모리가 필요하다. 실제로 Coda 서버로 동작하고 있는 Linux 머신에서 'top' 명령을 이용하여 메모리 사용현황을 확인해보면, Coda 서버 데몬인 'codasrv'와 클라이언트 데몬인 'venus'에 의해 물리적인 메모리의 40%가량이 점유되는 것을 확인할 수 있다. 따라서, 일반적으로 Coda 서버와 클라이언트는 분리되어 사용되는 것이 권장되고 있다. 앞으로, 여러 가지 방법으로 Coda 시스템을 구성하여 성능을 평가해봄으로서 최고의 성능을 발휘할 수 있는 시스템 구성을 찾아내어 이용해야 할 것이다.

3. 결 론

지금까지 리눅스 클러스터 환경에 적합한 분산 파일 시스템으로서 Coda 분산 파일 시스템에 대해서 알아보았다. Coda 시스템은 Coda 클라이언트에서의 캐시 정책과 이를 이용한 네트워크 단절시의 유연한 대처, Coda 서버에서의 Replication 정책과 VSG의 효과적인 이용 등 여러 가지 면에서 진보된 네트워크 분산 파일 시스템이라 할 수 있다.

그러나, Coda 클라이언트는 다중 cell이 아닌 단지 하나의 Coda cell 에만 속할 수 있다는 점이나, 서로 다른 여러 대의 클라이언트에서 업데이트된 파일들에 대한 Conflict 문제, Replication 서버들에 대한 부하 분산에 대한 문제, 단일 머신에서의 Coda 서버와 클라이언트의 동시 구동시 성능 저하의 문제 등, 아직 해결되어야 할 점들이 많이 있다. 이러한 문제들이 해결된다면 Coda 시스템은 리눅스 클러스터 환경에서 높은 고가용성을 실현시킬 수 있는 훌륭한 해결책이 될 수 있을 것이라고 생각된다.

[참 고 문 헌]

[1] Braam,P.J. "The Coda Distributed File System", *Linux Journal*, June 1998.

[2] Satyanarayanan.M., "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Workshop* Sep. 1989.

[3] 서대화, "리눅스 클러스터 환경에서의 분산 병렬 파일 시스템", *정보처리학회지 Vol.6, No.6*, pp.47-48.1999.

[4] 김해진, "리눅스 연구 개발 현황 및 방향", *정보처리학회지 Vol.6, No.6*, pp.12-16, 1999.

[5] 최재영 외 3인, "고가용성 리눅스", *정보처리학회지 Vol.6, No.6*, p.19, 1999.

[6] Coda File System, http://www.coda.cs.cmu.edu

[7] Distributed file systems on Linux, http://www.idg.net/crd_file_81719.html

[8] Linux Virtual Server Project, http://www.linux-vs.org