효율적인 멀티미디어 서비스를 위한 리눅스 클러스터 파일 시스템*

강미연, 홍재연0, 김형식
충남대학교 컴퓨터과학과
{herbtea, lh0701⁰, hskim}@cs.cnu.ac.kr

A Linux Cluster File System for Efficient Multimedia Services

Mi-Yeon Kang Jae-Yeon Hong⁰ Hyong-Shik Kim
Dept. of Computer Science, Chungnam National University

요 약

최근 리눅스 클러스터 시스템의 활용 범위가 커지면서 멀티미디어 서비스를 제공하려는 시도가 나타났다. 리눅스 클러스터 시스템 상에서 효과적으로 멀티미디어 서비스를 제공하려면 적합한 파일 시스템의 도움이 필수적이다. 즉, 클러스터 파일 시스템을 통하여 응용 프로그램에 대한 단일 입출력공간을 제공하고 효율적인 파일/디렉토리 연산을 제공하는 기술이 중요하다. 본 논문에서는 리눅스 클러스터 시스템을 위한 클러스터 파일 시스템(Cluster File System, 이하 CFS로 표기)을 설계 및 구현한다. CFS는 리눅스 파일 시스템 위에 사용자 수준에서 구현된 시스템으로 사용자에게는 단일 시스템 이미지를 제공한다. 내부적으로는 대용량의 파일이 분산되어 저장되며 이를 위해 파일/디렉토리 정보도 각각의 노드에서 분산 관리 된다. 사용자에게는 응용 프로그램의 개발이 용이하도록 API가 제공되며, 또한 CFS를 관리하기 위한 도구들이 제공된다.

1. 서 론

2000년대에 들면서 고속 네트웍의 보급으로 사용자들은 고품질의 멀티미디어 데이터를 실시간으로 제공 받기를 원하게 되었다. 이를 위해서는 고성능 대용량의 서버가 필요한데 리눅스 클러스터 시스템은 가격 대 성능비가 우수하고 고 성능, 고 가용성, 고 확장성을 지원할 수 있는 장점을 가지고 있다는 점에서 기존의 단일 서버의 대안으로 떠오르고 있다.

멀티미디어 서비스에 적합한 리눅스 클러스터 시스템을 구현하기 위해서는 사용자의 요구에 대해 원활한 처리를 보장할 수 있고 더불어 기억 공간 및 프로세서의 무한한 확장성 보장, 장애 발생 시 중단 없이 지속될 수 있는 서비스를 제공하는 신뢰성 보장, 이를 운영하기 위해서 드는 비용 및 비용 대비 성능에 있어서의 경제성 등을 제공하는 클러스터 파일 시스템의 개발이 필요하게 되었다.

이에 본 논문에서는 위에서 언급한 내용을 충족할 수 있는 파일 시스템을 제안한다. 우선 사용자 수준의 CFS의 구조를 설계하고, CFS 상에서 수행되는 응용 프로그램을 쉽게 구성할 수 있도록 응용 프로그램을 위한 라이브러리를 구현한다. 또한 CFS 관리를 용이하게 하기 위한 관리 도구를 개발한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 시스템에 대한 요구 사항을 분석하고, 3장에서는 시스템의 전체적인 개요를 살펴본다. 4장에서는 시스템 설계 및 구현 방법에 대하여 설명하고, 5장에서는 결론 및 향후 연구 과제에 대해 기술한다.

2. 요구 사항 분석

본 시스템은 다음과 같은 요구 사항들을 만족시켜야 한다.

- 리눅스 파일 시스템 위에 구현
- 멀티미디어 서비스에 적합
- 단일 시스템 이미지(Single system image) 제공
- 모든 노드에서 대칭적으로 동작
- 커널 수준이 아닌 사용자 수준에서 구현
- 응용 프로그램 개발을 위한 API 함수 제공
- 분산 디렉토리 지원
- 파일 시스템 관리를 위한 도구 제공

* 이 논문은 정보통신부의 선도기반기술개발사업에 의하여 수행된 과제의 결과임.

3. CFS 시스템

CFS는 그림 1과 같이 전체 n개의 노드로 구성 되어 있으며 각 노드에는 별도의 파일 시스템이 존재한다. 각 노드에 존재하는 파일 시스템의 일부를 할당하여 전체 CFS를 구성한다. cfsd는 파일/디렉토리 관리, 노드간 통신 등의 서비스를 제공한다. 응용 프로그램은 CFS daemon(cfsd)를 통해서만 CFS에 저장된 멀티미디어 데이터에 접근할 수 있으며 CFS shell은 응용 프로그램처럼 동작한다.

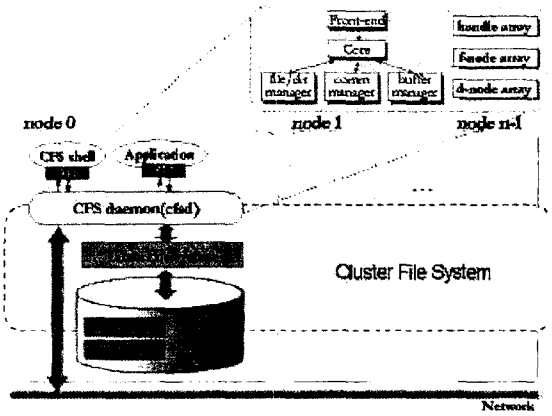

그림 1: CFS 시스템 구조

CFS는 기본적으로 리눅스 파일 시스템 위에 구현된다. 각 노드에서는 cfsd라는 데몬이 존재하여 상위의 응용 프로그램 및 CFS 쉘에 API 함수를 통하여 CFS를 제어할 수 있도록 하며, cfsd는 각 노드에 분산된 파일과 디렉토리 구조를 관리하는 파일/디렉토리 관리자(file/directory manager), 응용 프로그램과 데몬 또는 데몬 사이의 통신을 담당하는 커뮤니케이션 관리자(communication manager), 효율적인 데이터 엑세스를 위한 버퍼 관리자(buffer manager)로 구성되어

있다.

각각의 노드에는 스트라이핑된 데이터 블럭들(blocks)이 저장되며 각 노드에서의 파일/디렉토리 정보를 유지하기 위해 각각 f-node array, d-node array라는 자료 구조를 저장한다.

4. 설계 및 구현

4.1 저장 구조

CFS에서는 대용량의 멀티미디어 파일의 입출력 시간을 단축하고 결함 허용 기능을 제공하기 위해서 단일 파일은 각 노드에 분산 저장 되어야 한다. 또한 이렇게 분산되어 있는 파일을 관리하기 용이하도록 하기 위해서는 기존의 파일 시스템과 동일한 디렉토리 구조를 제공하여야 한다. 이를 위해서는 디렉토리, 파일, 핸들 등의 정보가 각각의 노드에서 분산 관리된다.

예를들면, 현재 두개의 노드로 구성된 클러스터 시스템에서 아래 그림 2와 같이 디렉토리 구조를 생성한다고 하면, 어떤 파일은 file3의 저장 형태와 같이 부모 디렉토리와 파일이 다른 노드에 저장 될 수 있다. 이때, 노드 0번에서는 파일 자체에 대한 정보를 가지고 있어야 하며, 노드 1번에서는 해당 파일이 어느 노드의 어느 위치에 존재하는지에 대한 정보를 가지고 있어야 한다.

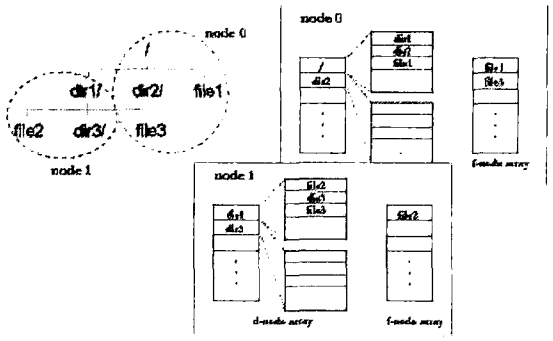

그림 2: CFS 디렉토리 저장구조의 예

이를 위해서 CFS에서는 각각의 노드에 디렉토리 구조를 위한 d-node array, 파일의 정보를 위한 f-node array, 파일 디스크립터를 위한 handle array 등의 데이터 구조와 파일의 조각들인 데이터 블럭을 저장하고 관리한다.

4.1.1 데이터 블럭의 표현

하나의 파일은 일정 크기로 스트라이핑되어 내부 표현용 이름으로 저장된다. 이때 블럭의 이름은 a.b.c 형식으로 붙여지는데, 여기서 a 는 파일이 정의 되어있는 노드의 인덱스를, b는 해당 노드에서 파일이 저장된 fnode_array에서의 인덱스를 나타낸다. 그리고 c는 데이터 블럭의 순차적인 인덱스를 나타낸다. 예를 들면, 노드 1에서 인덱스 5를 부여 받은 파일이 블럭 별로 저장될 때는 1.5.0, 1.5.1 등의 이름을 갖게 되며 각 노드에 골고루 분산된다. 예를 들어 첫 번째 블럭 1.5.0 파일은 노드 1에 저장되지만, 두 번째 블럭 1.5.1 파일은 노드 0에 저장될 수 있다.

4.1.2 f-node array

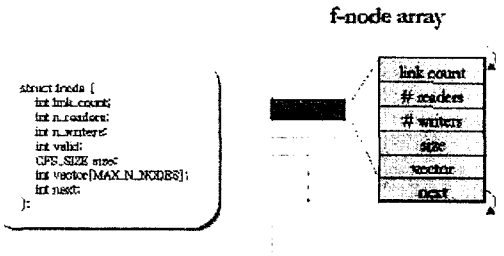

그림 3: f-node array

CFS 상에 존재하는 파일의 정보를 저장하기 위한 자료 구조이다. Size 필드는 해당 파일의 크기를 나타내며, vector는 해당 파일이 어느 노드에 분산 되어 있는지에 대한 정보를 갖는다. link_count 필드는 해당 파일에 대한 링크 수를 나타낸다. n_readers 와 n_writers 필드는 각 파일을 읽기 모드나 쓰기 모드로 오픈한 수를 나타낸다(그림 3).

4.1.3 d-node array와 handle array

d-node array는 CFS 상의 디렉토리 정보를 저장하기 위해 사용된다. d-node array의 각각의 엔트리는 해당 디렉토리의 링크 카운트, 해당 디렉토리 안에 들어있는 서브 디렉토리와 파일들의 정보 등을 가지고 있다.

handle array는 CFS 상의 오픈된 파일의 정보를 저장하기 위해 사용한다. 해당 파일이 저장되어 있는 위치 정보와 read/write시 파일의 오프셋 정보 등이 저장된다.

4.2 CFS 데몬

CFS를 구성하는 각 노드에서는 CFS daemon(cfsd)이 수행된다. cfsd는 CFS를 제어하기 위해 리눅스 파일 시스템 위에서 구현된 프로그램이다.

cfsd는 같은 노드의 응용 프로그램이나 쉘로부터 API를 통하여 서비스 요구를 받는다. 이때 요구를 자신이 수행 할 수 없을 경우는, 네트워크를 통하여 연결된 CFS를 구성하는 다른 노드에 수행을 요구하게 된다. 다른 노드로부터 요구를 받은 노드도 역시 자신이 수행할 수 없을 경우는 다른 노드에게 서비스를 요구하고 자신이 수행할 수 있을 경우는 요구를 수행한 후 요구를 보낸 노드에 답신을 보낸다. 즉, API를 통하여 요구를 받은 cfsd는 응용 프로그램에게 요구에 대한 응답을 보내게 되고, 다른 노드로부터 서비스 요구를 받았을 경우는 요구 받은 노드에 응답을 보낸다. 응용 프로그램과 cfsd 사이의 통신과 노드 사이의 통신 모듈은 라이브러리 형태로 구현되어있다.

한편 cfsd는 file/directory manager, communication manager, buffer manager 등의 기능을 제공한다.

4.2.1 File/directory manager

CFS상에 파일이나 디렉토리를 생성하거나 삭제하는 등과 같이 파일과 디렉토리에 관련된 작업들을 수행한다.

CFS에서는 파일/디렉토리의 정보는 한 노드에 저장, 관리되지 않고 여러 노드에 분산되어 저장된다. 이때, 정보는 파일이나 디렉토리의 생성 요청을 받은 노드에 저장된다.

여러 노드에 분산되어 저장된 파일/디렉토리를 이용하여 전체 파일 시스템을 구성하기 위하여 각 디렉토리는 하위에 존재하는 파일/디렉토리의 리스트와 파일/디렉토리의 정보가 저장된 노드 번호를 유지한다. 루트 디렉토리는 항상 0번 노드에 저장되고, 루트로부터 시작해서 하위 파일/디렉토리 리스트를 이용하여 전체 파일시스템의 구조를 구성 할 수 있다.

이는 기존의 유닉스 시스템과 유사한 디렉토리 구조를 제공하여 파일/디렉토리 관리를 용이하게 해주며, 분산되어 저장되는 파일/디렉토리 정보와 파일의 블럭들을 하나의 이미지로 보이도록한다. cfsd는 이러한 내부 구조가 사용자에게 투명(transparency)하게 보이도록 한다.

4.2.2 Communication manager

통신 라이브러리를 사용하여 동일 노드 내에서 응용 프로그램 프로세스와 데몬 프로세스 사이의 데이터 교환(Inner-node communication), 그리고 노드 간 데몬 사이의 데이터 교환(Inter-node communication)을 한다.

응용 프로그램과 데몬 사이의 데이터 교환은 대용량의 멀티미디어 데이터를 빠르게 주고받기 위해 공유 메모리를 사용하며, 공유 메모리의 동기화를 위해서 세마포어를 사용한다. 또한, 응용 프로그램이 데몬에게 서비스 요구를 하기 위해 Named pipe(FIFO 파일)를 이용한다.

각 노드에 존재하는 데몬은 상호간에 서비스 요구 및 요구에 대한 응답을 하며, 이 때 데이터 교환을 위해서 TCP/IP 소켓을 사용한다.

이는 CFS 시스템의 단일 시스템 이미지 제공을 가능하게 한다. 예를 들어 사용자는 파일/디렉토리의 위치에 상관없이 자신이 접속한 노드의 데몬에게 서비스 요청을 한다. 이때, CFS 데몬은 사용자의 요청을 자신이 처리할 수 없을 경우(예를 들어, 파일의 정보가 다른 노드에 저장된 경우) 다른 노드의 데몬에게 서비스를 요청하고 요청에 대한 응답을 받으면 이를 사용자에게 재전송한다. 이러한 방법으로, 사용자는 정보가 저장된 위치를 알 필요가 없이 CFS에 접근 할 수 있다. 이는 사용자에게 사용의 단순성, 용이성 등을 제공한다.

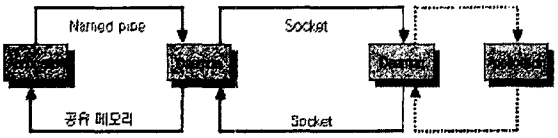

그림 4: CFS 통신 모듈

4.2.3 Buffer manager

읽기를 수행하는 사용자의 특성을 살펴보면 공간적 지역성(spatial locality)과 시간적 지역성(temporal locality)을 나타낸다. 즉 읽기 수행 중 대부분이 연속적인 읽기 이거나 읽기를 수행하는 위치가 반복된다. 따라서, 디스크에서 데이터를 읽어올 때 블럭의 단위로 데이터를 읽어 메모리에 저장하여 성능을 향상시키는 것과 같이, 본 시스템에서도 버퍼를 정의하여 파일로부터 일정크기의 데이터를 읽어 버퍼에 저장한다.

쓰기를 수행할 경우도 마찬가지로 버퍼를 사용한다. 사용자의 쓰기 요청이 있을 경우마다 파일의 내용을 갱신하려면 디스크 접근 지연시간이 너무 길어진다. 그 대신 사용자의 쓰기 요청에 대한 데이터를 버퍼에 저장한 후 버퍼가 찼을 경우에 버퍼의 내용을 일괄적으로 파일에 저장함으로써 디스크 접근 횟수를 줄일 수 있다.

4.3 CFS 라이브러리

CFS와 상위의 응용 프로그램의 인터페이스 역할을 하는 것으로 CFS에서 제공되는 기능들을 사용자 수준에서 응용 프로그램 구현이 용이하도록 API 형태로 제공된다. 주로 CFS 파일/디렉토리 관리와 시스템 관리를 위한 함수 등으로 구성되며 응용 프로그램 생성과 종료를 위한 함수와 파일 시스템 체크를 위한 함수 등도 구현되어 있다(표 1).

표 1: CFS 라이브러리

분류	API 함수
파일/디렉토리 생성/삭제	cfs_create(), cfs_delete()
파일 관련	cfs_open(), cfs_close(), cfs_read(), cfs_write(), cfs_seek(), cfs_copy(), cfs_compare(), cfs_link(), cfs_unlink()
파일 시스템 관리	cfs_reset(), cfs_init(), cfs_flush(), cfs_import(), cfs_export()
기타	cfs_begin(), cfs_end(), cfs_fsck()

4.4 CFS 지원 쉘

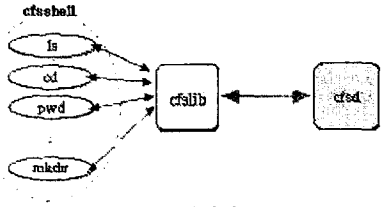

그림 5: CFS 명령어 개요

쉘 명령어들은 CFS의 디렉토리와 파일을 관리하기 위한 도구로 CFS 라이브러리에서 제공하는 API 함수를 사용하여 구현된 프로그램이다. 각각의 명령어는 다른 이름으로 링크되어 있으며, 본질적으로는 하나의 응용 프로그램으로 이루어져 있다.

본 시스템에서 제공되는 쉘 명령어는 디렉토리 관리를 위한 ls, cd, pwd, mkdir, rmdir 등과 파일 관리를 위한 import, export, link, unlink, cmp, cp, cat 등이 제공되며 일반 유닉스 쉘 명령어와 유사한 방법으로 작동한다. 그밖에도 파일 시스템의 일관성을 확인하기 위한 fsck 등의 명령어도 제공된다.

5. 결론 및 향후 과제

본 논문에서 제시된 CFS는 대용량의 멀티미디어 데이터 서비스를 하는데 있어서 기존의 단일 서버의 문제점을 해결해 줄 수 있을 것이다. 즉 리눅스 환경에서 구현된 클러스터 시스템에 적합한 CFS를 통하여 비용, 확장성 그리고 안전성 측면에서 고가의 고성능 서버를 대치할 수 있고 이를 통하여 저렴한 비용으로 다양한 서비스를 제공할 수 있을 것으로 예상된다.

현재 cfsd의 일부 기능을 확장하는 작업이 진행중이다. 블럭 단위의 스트라이핑 기능 기반에 결함 허용 기능과 다른 노드에 저장된 데이터에 대한 빠른 접근을 보장하기 위한 캐쉬 기능을 추가할 예정이다. 또한 성능 평가를 통한 시스템 최적화 작업이 필요하다.

참고 문헌

[1] D. H. Spector, Building Linux Clusters, O'Reilly and Associates, 2000.
[2] R. Buyya, High Performance Cluster Computing, Volume 1, Architectures and Sysetms, Prentice-Hall, 1999.
[3] G. F. Pfister, In Search of Clusters, Prentice-Hall, 1998.