OpenCFS 클러스터 파일 시스템의 구현 및 성능 평가

전 승협°, 차규일, 김 진미, 유혁
고려대학교 컴퓨터학과

The Implementation and Performance Analysis of a OpenCFS Cluster File system

Seung-Hyub Jeon°, Gyu-Il Cha, Jin-Mi Kim, Chuck Yoo
Dept. of Computer Science and Engineering, Korea University

요 약

본 논문에서는 멀티미디어나 데이터베이스 등 대용량 입출력을 효율적으로 지원하기 위하여 고속 네트웍으로 연결된 클러스터링 환경에서 동작하는 클러스터 파일 시스템인 OpenCFS를 설계하고 구현하여 성능을 평가한다. 구현된 클러스터 파일 시스템은 입출력 장치의 한계를 극복하기 위하여 스트라이핑(striping) 기법을 통한 병렬 입출력(parallel I/O)을 수행하고, 능동적으로 시스템 내부 정책 변경을 가능하게 하는 오픈 임플리멘테이션(Open Implementation) 방법론은 적용함으로써 응용프로그램의 시스템 내부 정책에 대한 접근 방법을 제공한다. 실험을 통하여 구현된 클러스터 파일 시스템의 성능을 분석한 결과, 사용자가 기존의 프로그래밍 환경을 유지하면서 시스템 내부 정책을 변경함으로써 개선된 성능의 입출력 서비스를 제공 받을 수 있다.

1. 서 론

과학 계산과 같이 대규모 데이터를 처리하는 많은 분야에서 고속의 계산 능력을 얻기 위한 방법으로 병렬적 특성을 이용하는 병렬 컴퓨터의 연구가 대학, 연구소 그리고 슈퍼컴퓨터 회사들을 중심으로 매우 활발히 이루어져 왔다. 그 결과, 병렬 컴퓨터는 상당한 기술적 진보가 이루어졌으며, 부분적으로 포트란과 같은 언어에서 병렬성을 나타내고 처리하는 기술은 상당 부분 안정화 되어 실제 실용화 단계까지 이르렀다. 그러나 지금까지 이런 병렬 컴퓨터의 주된 연구 분야는 주로 계산 중심(Computation intensive) 분야에 집중되어져 왔고, 이런 기술적 진보는 최근 멀티미디어나 대규모 데이터베이스와 같은 입출력 중심(I/O intensive) 작업이 증가함에 따라 한계에 부딪치게 되었다. 이러한 한계의 원인은 두 가지에 기인한다. 첫째로 입출력 장치의 속도가 프로세스나 메모리의 속도를 따라가지 못하는 하드웨어 발전 속도의 불균형으로 인하여 대규모의 입출력이 발생할 경우에 입출력 장치에서 병목현상이 발생하게 된다. 둘째로는 기존의 개발 방법론인 블랙 박스 방법론(BBA methodology)에서 원인을 찾을 수 있다. 이 방법론에서는 내부 정책을 고정시키고 사용자에게 인터페이스만을 제공한다. 하지만 응용프로그램이 다양해지고 그에 따라 입출력 방식이 서로 다름에 따라서 파일 시스템으로부터 제대로 서비스를 받지 못하는 응용 프로그램들이 존재하게 된다. 이 같은 두가지 문제점을 해결하기 위해서 스트라이핑을 통한 디스크 병목현상을 최소화하고 오픈 임플리멘테이션 방법론을 적용하여 다양한 정책을 지원할 수 있는 클러스터 파일 시스템인 OpenCFS를 설계하고 평가한다.

2. OpenCFS의 구현

2.1. 기본 구조

OpenCFS는 다중 노드로 연결된 클러스터 구조를 갖는다. 이 클러스터 환경은 100Mbps의 Fast Ethernet을 통해 고속 네트워크의 이점을 최대한 활용하고 분산되어진 로컬 디스크를 통한 입출력의 병렬성을 증가시킬 수 있다.

2.2. 상세 구조

OpenCFS는 서버의 역할을 담당하는 입출력 분산 서버(IODS)와 응용 프로그램에 라이브러리 형태로 결합되

어 클라이언트 역할을 하는 입출력 라이브러리(IOLC)의 두 부분으로 구성된다. 클라이언트와 서버는 1:n의 매핑구조가 가능하며 하나의 응용프로그램의 입출력 요청을 n개의 입출력 분산 서버로 분산 요청함으로써 인터리빙을 통한 입출력 장치의 지연 시간 단축 효과를 얻을 수 있다. 뿐만 아니라 클러스터 파일 시스템의 클러스터링에 참여하고 있는 각 노드의 입출력 분산 서버는 m개의 입출력 라이브러리와 통신 채널을 형성할 수 있다.

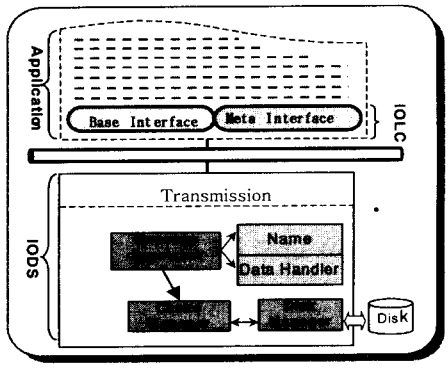


그림 1. OpenCFS의 기본 구조

2.2.1. 입출력 라이브러리

이 모듈은 응용 프로그램과 실제로 결합되어 사용자의 입출력 요청 및 개방 정보를 관리하고 처리하는 부분으로 사용자가 메타 인터페이스를 이용하여 개방 정보를 다루는 부분과 기본 인터페이스 서비스를 제공하는 부분 그리고 마지막으로 실제 데이터를 전송하기 위한 모듈로 구성되어 있다.

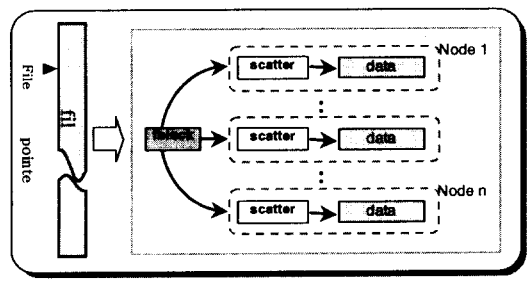
2.2.2 입출력 분산 서버

입출력 분산 서버는 스트라이핑된 데이터에 대한 실질적인 입출력을 담당하는 부분으로 각 노드의 입출력 분산 서버들과 메타 데이터를 분산 관리한다. 자료 관리자(data handler), 파일명 관리자(name handler), 캐시 관리자(cache manager)와 디스크 관리자(disk manager)등으로 구성되어 있다.

2.3. 파일과 메타데이터

OpenCFS의 구조는 각 노드의 디스크들을 마치 하나의 디스크처럼 사용할 수 있도록 해준다. 또한 리눅스 파일 시스템과 비슷한 선형 파일 모델(liner file model)과 유사한 구조를 가지도록 설계되어 있다. 그러므로 하나의 파일은 연속된 바이트의 스트림으로 보여지고 응용 애플리케이션들은 이런 바이트의 스트림 중에서 연속된 범위를 읽거나 쓸 수 있다.

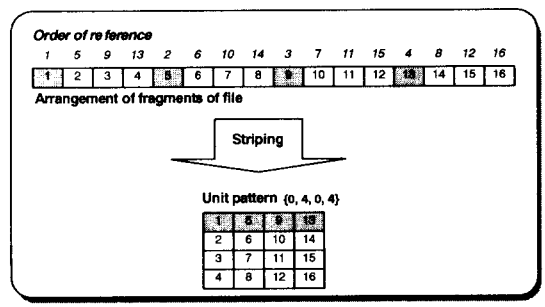
본 시스템에서 파일 구조는 [그림 2]와 같은 이차원적인 구조를 가진다. 시스템 전체적인 관점에서 보여지는 하나의 파일을 나타내는 메타 데이터 fblock과 노드 단위로 스트라이핑 된 데이터를 관리하기 위한 노드 단위의 메타데이터인 scatter로 구성된다. 하지만 이러한 이차원적인 구조는 내부에 숨겨지므로 유닉스처럼 파일이라는 논리적 구조에 집중하고 파일 포인터라는 가상적 위치에 사용해서 파일에 대한 읽기와 쓰기의 모든 동작을 처리한다.



[그림 2] 이차원적인 파일 구조

2.4. 단위 스트라이핑(cell striping) 표현법

입출력 장치의 지연 시간을 극복하고 입출력 성능을 향상시키기 위해서 스트라이핑을 사용하는 것은 필수적이다. 기존의 분산/병렬 파일 시스템은 이 스트라이핑 기법을 적용하기 위해 두 가지 유형을 제공한다. 동일한 단위 크기로 나누어 정적인 스케쥴링을 사용하는 방식과 다양한 스트라이핑 표현방법을 제공하는 두 가지 방법이 존재하는데 첫 번째 방법은 다양한 스트라이핑 유형을 사용하는 병렬 파일 시스템에는 적당하지 않고 두 번째 방법은 지나치게 복잡하다는 단점을 갖는다. 이런 문제점을 해결하기 위해서 OpenCFS에서는 유닉스 프로그래밍 환경을 유지하면서도 다양한 스트라이핑을 다룰 수 있는 '단위 스트라이핑(cell striping) 표현법'을 사용한다.



[그림 3] 스트라이핑 표현법의 단위 유형의 정의

단위 스트라이핑 표현법에서는 먼저 단위 표현 방법을 정의하고 이런 단위 유형(unit pattern)에 치환(replacement)과 재 수집(re-collection)을 적용해 다른

유형으로 변경하는 방법을 반복적으로 적용한다. 단위 스트라이핑 표현법은 유형의 단위 표현을 { 이웃 스트라이핑 단편의 배치 방향, 기본 유형의 크기, 다음 반복의 시작, 기본 반복 크기}의 네가지 쌍으로 표현한다.

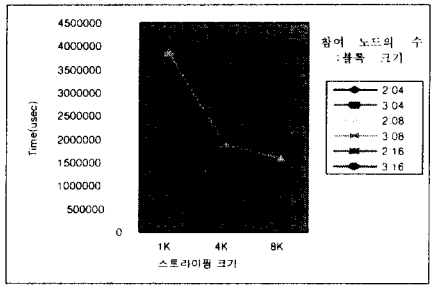
3. 실험 결과 및 고찰

3.1. 실험 환경

시스템의 성능을 분석하기 위하여 166Mhz의 CPU, 64Mbyte RAM, Fast Ethernet을 가지고 있는 범용 PC 4대를 통하여 클러스터링 환경에서 4Mbyte 크기의 파일을 연속적으로 읽기와 쓰기를 수행하면서 결과를 얻었다. 또한 사용자에게 개방되어 있는 정보인 스트라이핑의 참여 노드수와 스트라이핑 크기를 변화시키면서 10회 반복적으로 실험한 결과이다.

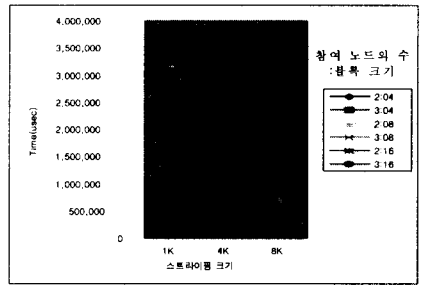
3.2. 읽기

[그림 4]에서 보듯이 읽기의 경우는 스트라이핑에 참여하는 노드의 수가 많을수록 입출력 병렬성이 높아져 성능이 향상된다. 또한 기본 디스크 블록의 크기가 클수록, 특히 블록의 정수배가 될수록 성능이 높아진다. 이것은 저장되는 디스크의 위치가 연속적으로 할당됨으로써 디스크 스케줄링의 이점을 얻은 것이다. 하지만 특이하게도 3개의 노드에 블록 크기가 4Kbyte로 한 경우 스트라이핑 크기가 8Kbyte가 된 경우는 다른 그래프와는 다른 결과를 나타내었는데 이것은 디스크 캐시 정책이 MRU가 필요한 상황에서 LRU 캐시를 사용했기 때문에 캐시의 효과를 얻지 못하고 극도의 부하만 증가시킨 결과로 분석된다.



3.3. 쓰기

스트라이핑의 크기가 클 때는 작은 경우에 비해 상대적으로 통신비용이 적게 들어 상당한 소요 시간 단축을 가져오게 된다. 그러나 [그림 5]의 2:08과 3:08의 경우를 제외하고는 그래프가 기울기가 완만해 짐을 알 수 있다. 이것은 스트라이핑의 크기가 디스크 블록 크기와 같아지는 점을 정점으로 통신에 따른 부하의 영향이 감소한다는 것을 의미한다. 그리고 같은 디스크 블록의 크기를 사용하는 경우에는 사용되는 노드의 수를 늘임에 따라 병렬성이 증가되어 성능이 향상되고 있음을 알 수 있다.



[그림 5] 쓰기의 성능 분석

4. 결론

본 논문은 대용량 파일의 처리를 원활히 하고 기존 분산/병렬 파일 시스템의 문제점을 해결하기 위한 OpenCFS의 구조와 성능에 대해서 다루고 있다. 스트라이핑을 통한 병렬성을 증가시키고 스트라이핑에 대한 오픈 임플리멘테이션 방법론을 적용하여 사용자가 알맞은 스트라이핑 정책을 사용할 수 있도록 하였다. 추후 계획으로는 다양한 캐시정책, 패리티 비트 유무등 다양한 정책을 지원함으로써 보다 효율적인 성능을 지원하는 클러스터링 파일 시스템을 구현하는 것이다.

참 고 문 헌

[1]유혁, "클러스터 파일 시스템", 병렬처리시스템 연구회지, 제 8 권 제 1 호, pp.22-31, 1997.2.

[2]차규일, 유혁 "OpenCFS : Open Implementation Approach to Cluster File System", 98년도 컴퓨터시스템 연구회 추계 학술발표회 논문집, pp.181-190, 1998.9.

[3]Chris Maeda, Hee-Woong Lee, Gail Murphy, and Gregor Kiczales, "Open Implementation Analysis and Design", In Proc. of International Symposium on Software Resuability, 1997.

[4]Hartman, J. and Ousterhout, J. "The Zebra Striped Network File System", ACM Transactions on Computer Systems, 1995.

[5]M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System", ACM Transactions on Computer Systems, 10(1):26-52, 1992

[6] Luis-Felipe Cabrera and Darrel D. E. Long. "Swift: Using Distributed Disk Striping to Provide High I/O Data Rates", Computing Systems, 4(4):405-436, 1991.