VIA기반의 병렬파일시스템 구현 방법

이윤영° 서대화
경북대학교 전자공학과
yylee@palgong.knu.ac.kr dwseo@ee.knu.ac.kr

A method of implementing parallel file system in base VIA

Yoon-Young Lee° Dae-Wha Seo
Department of Electronics, Kyungpook National University

요 약

클러스터링(clustering)은 병렬 처리를 위한 기술로 비교적 값이 싼 컴퓨터들을 네트웍으로 연결하여 전체가 하나의 고성능 수퍼 컴퓨터처럼 동작하게 하는 기술이다. 이 클러스터 시스템의 성능을 최대한 활용하기 위해서는 디스크 입출력에 생기는 병목현상을 해결하여야 하는데, 그 해결책의 하나로 병렬파일시스템을 들 수 있다. 기존의 병렬파일시스템은 TCP/IP기반의 소켓으로 메시지를 주고받았다. 그러나 TCP/IP는 프로토콜 오버헤드가 크고 처리 속도가 느리다. 본 논문에서는 이런 오버헤드를 줄이기 위해 도입된 Lightweight 메시징 기법인 VIA(Virtual Interface Architecture)를 이용하여 병렬파일시스템을 구현하기 위한 구체적인 방안을 제시하고 있다.

1. 서 론

프로세서와 네트웍 장비의 급속히 발전한데 반하여, 입출력 장치의 발전 속도는 이를 따라잡지 못하고 있다. 이 때문에 입출력 장치가 전체 컴퓨터 시스템의 병목이 되어 버렸다. 이러한 병목을 해소하는 하나의 방법으로 입출력 장치에 병렬성을 도입하였으며 이 결과로 현재 많은 클러스터링 컴퓨팅 환경에서 병렬입출력기법(Parallel I/O)과 병렬파일시스템(Parallel File System)이 사용되고 있다.

그런데 대부분의 병렬파일시스템에서는 WAN (Wide-Area Network) 환경에서 양극단간의 신뢰성있는 패킷 전송을 위해 고안된 TCP/IP 프로토콜을 기반으로 개발되어 있다. 이러한 TCP/IP는 동질적이며, 신뢰성이 매우 높아 전송 에러를 무시할 수 있는 클러스터를 위한 SAN(System Area Network) 환경에서는 불필요한 기능들을 많이 포함하고 있으며, 그림 1과 같이 커널 내부에 TCP/IP 프로토콜이 존재하여 패킷 전송 시마다 문맥교환과 커널 영역으로의 데이터 복사가 발생한다. 이런 한 통신 오버헤드는 네트웍을 통한 데이터 통신 량이 많은 병렬파일시스템의 성능을 저하시키는 큰 요인으로 작용하고 있다.

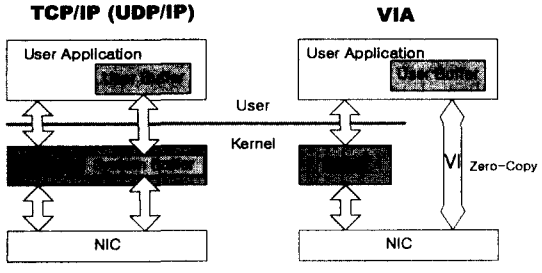

그림 1 TCP/IP와 VIA의 비교

이러한 단점들 때문에 AM, FM, BIP, U-NET, VIA 등의 다양한 Lightweight Messaging 기법들이 고안되었으며, VIA[1]는 Intel, Compaq, Microsoft 등의 많은 업체가 규격 정의에 참여하고 있으며 산업계의 표준으로 정착할 확률이 가장 높다.

본 논문에서는 리눅스 클러스터 환경에서 다양한 멀티미디어 및 과학계산용 프로그램의 병렬 파일 I/O를 담당할 수 있는 병렬 파일 시스템인 PFSL(Parallel File System for Linux clusters)[2]의 통신 프로토콜로서 VIA를 적용하기 위한 구체적인 방안을 제안한다.

2. PFSL

본 논문은 PFSL을 기반으로 하고있다. PFSL은 크게 클러스터 파일 매니저, 블록 매니저 그리고 메타데이터 매니저로 나뉘어지며, 그림 2와 같이 구성된다. 클러스터 파일 매니저는 실행 노드에서 응용프로그램이 요청하는 파일 서비스를 담당하고, 블록 매니저는 입출력 노드에서 분산 저장된 파일에 대한 데이터 블록의 입출력 서비스를 담당한다. 메타데이터 매니저는 분산 저장된 파일에 대한 메타데이터를 관리하는 서버이다.

응용프로그램이 PFSL 라이브러리를 통해 클러스터 파일 매니저에게 파일에 대한 서비스를 요청하면, 클러스터 파일 매니저는 응용프로그램의 요구를 수행하기 위해 메타데이터 매니저에 있는 메타데이터를 이용하여 필요한 데이터 블록의 논리적 주소를 연산한 후, 입출력 노드의 블록 매니저에게 데이터 블록에 대한 서비스를 요구한다. 블록 매니저는 클러스터 파일 매니저로부터 데이터 블록에 대한 서비스 요청이 있을 경우, 디스크의 입출력을 통해 데이터 블록의 읽기/쓰기 요청을 서비스한다.

클러스터 파일 매니저에서 멀티쓰레드를 이용해서 동시에 여러 블록 매니저로 데이터 블록을 요구하여, 블록 매니저에서의 데이터 블록 서비스가 병렬로 진행될 수

있다. 이러한 병렬 처리의 장점은 입출력 노드가 많아질 수록 데이터의 입출력에 대한 성능 향상을 가져온다.

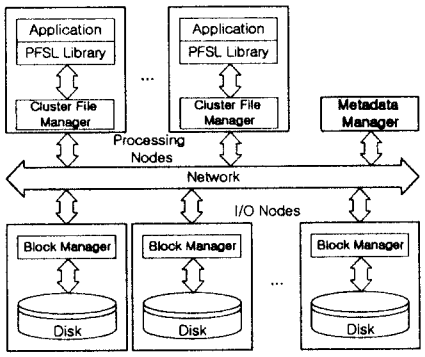

그림 2 PFSL의 구성

3. VIA기반의 PFSL

3.1 일반 응용프로그램에 VIA의 적용

VIA를 통신 프로토콜을 이용하는 응용프로그램을 작성하는 방법은 일반적으로 두 가지로 생각할 수 있다.

하나는 기존의 응용프로그램을 수정하지 않고, 네트웍 장치를 하드웨어적으로 VIA 지원하는 장치로 설정해 주는 방식이다. 이때는 Emulex사에서 개발한 cLAN 장비와 같이 VIA위에서 IP계층을 에뮬레이트해주는 LANE(LAN Emulation)[3]와 같은 디바이스 드라이버가 제공되어야 한다. 이때는 디바이스 드라이버가 커널 수준에서 동작하므로 문맥 교환이 빈번하게 이루어지며, 커널 영역으로의 데이터 복사 또한 막을 수 없어 Lightweight 메시징의 장점을 살리기 힘들다는 단점이 존재한다.

다른 한가지 방법은 기존의 socket과 같은 통신 프로토콜을 VIA를 이용하는 라이브러리로 1:1 대응시켜 대체하는 방식이다. 이때, VIA의 기본 라이브러리인 VIPL (Virtual Interface Provider Library)을 직접 사용하는 경우와 VIPL위에 흐름제어 등을 담당하는 계층을 하나 더 두어 보다 안정적인 통신을 하도록 하는 라이브러리를 제작하여 사용하는 방법이 있을 수 있다. 이때는 기존 응용프로그램의 코드를 수정해주는 것이 불가피하며, 때에 따라서는 VIPL 사용의 난이성 때문에 응용프로그램의 복잡성이 기존의 프로그램보다 훨씬 증가할 수 있다.

3.2 병렬파일시스템의 특성

일반적인 응용일 경우 3.1에서 언급한 두 가지 방식으로 응용프로그램에 VIA를 적용할 수 있다. 그러나 병렬파일시스템에는 이러한 방식을 그대로 적용할 경우 좋은 성능을 기대하기 힘들다.

PFSL에서는 응용프로그램의 요청을 네트웍을 통하여 파일매니저에 보내고, 이 요청을 파일 매니저는 자신이 처리할 수 있는 요청은 자신이 처리하고, 그렇지 못한 요청은 블록 매니저에 다시 요청을 하게된다. 이렇게 파일시스템을 이루고 있는 컴포넌트들 간의 빈번한 요청이 이루어지는데, 이러한 요청을 처리하는 메커니즘이 VIA를 그대로 적용하기에 적합하지 못하다.

PFSL의 요청 처리 메커니즘은 그림 3과 같다. 서비스를 제공하는 컴포넌트를 서버로 표현하였으며, 서비스를 요청하는 컴포넌트를 클라이언트로 나타내었다. 각 컴포넌트는 초기화 시에 채널을 열어두고(OPEN) 다른 컴포넌트로부터 연결을 기다린다(WAIT). 이 때, 클라이언트가 요청을 위해 연결(CONNECT)을 시도하고, 요청에 대한 정보를 보내면(SEND), 서버는 이 정보를 받아(RECV) 서비스를 수행하고(PROCESS), 서비스 결과를 클라이언트에 돌려주고(TRANSFER), 모든 서비스가 끝났다는 Ack을 클라이언트에 보낸(SEND) 다음 연결을 끊고(CLOSE) 다음 서비스 요청을 대기한다(WAIT).

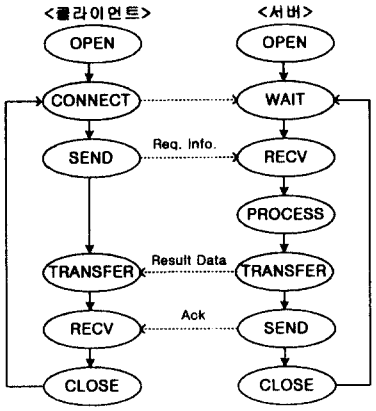

그림 3 PFSL의 요청처리 메커니즘

그림 3에서 보듯이 PFSL의 각 컴포넌트들은 요청이 발생할 때마다 연결을 새롭게 설정하고, 요청을 처리하고 난 다음에는 연결을 끊어버린다. 따라서 하나의 프로세스를 수행하는 동안에도 수없이 많은 연결이 수립되고 해제된다.

M. Banikazemi의 논문에서 표 1과 같은 성능 측정 결과를 얻을 수 있었다[4]. 표에서 보는 것처럼 VIA의 구현 방식에 따라 차이는 있지만 대부분의 VIA의 구현에서 연결을 설정하는데 엄청난 시간이 소요되며, 잦은 연결 성정과 해제는 성능 저하의 요인이 된다. 따라서, 병렬파일 시스템에서는 단순히 TCP/IP로 구현된 통신 방식을 VIPL을 이용하여 대체해서는 좋은 성능을 얻을 수 없다.

Operation	M-VIA	BVIA	cLAN
Creating VI	93	28	3
Destroying VI	0.19	0.19	0.11
Establishing Connection	6465	496	2949914
Tearing Down Connection	3	9	155

표 1 VIA 구현의 마이크로 벤치마크 결과

3.3 병렬파일시스템을 위한 하이브리드 연결 모델

3.2에서 본 바와 같이 병렬파일시스템에서는 단순히 TCP/IP를 VIA로 대체하는 것으로는 좋은 성능을 낼 수 없다. 따라서 본 논문에서는 HCMP(Hybrid Connection Model for PFSL)를 제안한다.

HCMP는 VIPL을 기반에 흐름제어를 추가하여 자체 제작한 VIA 라이브러리를 이용하면서, 디바이스 드라이버 수준에서 TCP/IP를 에뮬레이트해주는 LANE[3]를 병행하여 사용한다. 이것은 그림 4와 같이 데이터 전송을 위한 채널과 제어 신호를 위한 채널을 분리하여 둠으로써 가능해진다. 데이터 전송을 위한 채널은 VIA 라이브러리를 이용하여 작성하게되며, 이때 각 병렬파일시스템들의 연결은 각 컴퍼넌트들이 초기화 될 때 단 한번 수립되어, 파일시스템 종료 시까지 계속 유지한다. 따라서 VIA의 연결 오버헤드를 없애면서도 Lightweight 메시징으로 인한 성능향상을 얻을 수 있다.

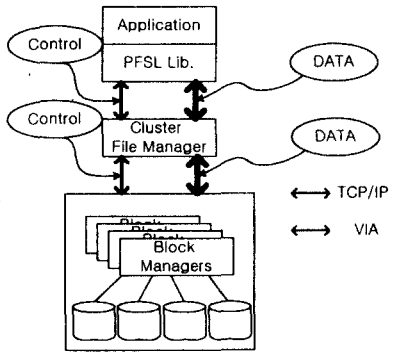

그림 4 HCMP의 구성

그림 5에서는 병렬파일시스템에서 가장 많이 사용되며, 가장 중요한 성능 평가 척도인 READ 요청이 HCMP를 적용하여 VIA를 사용하였을 때, 어떻게 수행되는지를 나타내었다. 먼저 클라이언트는 서버에 READ 요청임을 알리고, 읽어야할 블록의 리스트를 TCP/IP 채널을 통해 전송한다. 서버는 요청 받은 블록 리스트를 읽어서 VIA 채널을 통해 전송하고, 전송이 끝난 후 실제로 읽은 블록의 수를 Ack을 대신하여 전송한다. 이 과정에서 TCP/IP 연결과 해제만이 한번 발생하며, VIA 연결은 전혀 발생하지 않는다.

4. 결론

병렬파일시스템은 클러스터 환경의 과도한 파일 입출력 상황에서 병목현상을 해결하기 위한 수단으로 고안되었다. 그러나 현재 사용되고 있는 통신 프로토콜인 TCP/IP는 WAN환경을 위해 고안되었으므로 신뢰성이 높은 클러스터 환경에서는 많은 오버헤드를 포함하고 있어 적합하지 않다. 따라서 병렬파일시스템에 Lightweight 메시징 기법을 적용하는 것은 타당하며, VIA는 Lightweight 메시징 기법 중 가장 전망이 밝다.

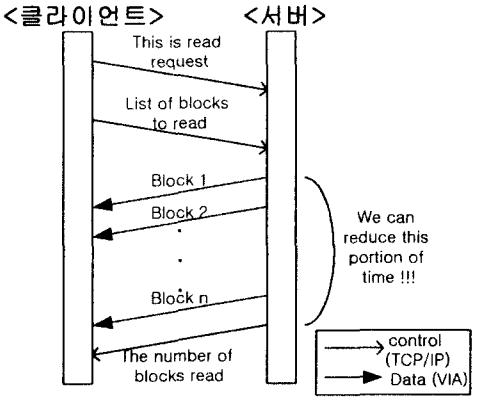

그림 5 HCMP를 적용한 PFSL READ

본 논문에서는 VIA를 병렬파일시스템에 적용하기 위해서는 병렬파일시스템의 특성상 TCP/IP의 일반적인 대체 수준이어서는 곤란함을 보였다. 즉, VIA의 연결 오버헤드가 큰 점을 고려하여 VIA 연결이 가능하면 발생하지 않도록 구성하여야 함을 본 논문을 통해 알 수 있다.

본 논문에서는 해결책으로 HCMP를 제안하여, VIA 연결이 병렬파일 시스템의 초기화 시에 단 한번만 이루어지도록 해서 VIA의 연결 오버헤드가 전체 파일시스템의 성능에 크게 영향을 주지 않도록 설계하였다.

참고문헌

[1] "Virtual Interface Architectrue Specification", draft version 1.0, 1997.

[2] J. Cho, C. Kim, and D. Seo, "A Parallel File System Using Dual Cache Scheme and Prefetching", *The 2000 International Conference on Parallel/Distributed Processing Techniques and Application (PDPTA2000)*, June, 2000

[3] "cLan for Linux, Software User's Guide", Emulex, 2001

[4] M. Banikazemi, J. Liu, S. Kutlug, A.,Ramakrishna, P. Sadayappan, H. Sah, and D. K. Panda, "VIBe: A Micro-benchmark Suite for Evaluating Virtual Interface Architecture (VIA) Implementations", *Int'l Parallel and Distributed Processing Symposium (IPDPS)*, April 2001.

[5] "VI Architecture Software Developer's Guide", Giganet, 1999

[6] M. Banikaze, B. Abali, and D. K. Panda, "Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA)", *Fourth Int'l Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC '00)*, Jan 2000, pp. 145-161.