클러스터 웹서버를 위한 UDP/IP기반 SAN의 구현 및 성능 분석

이주평° 박규호
한국과학기술원 전자전산학과 전기및전자공학전공 컴퓨터공학연구실
{jplee°, kpark}@core.kaist.ac.kr

Implementation & Analysis of SAN based on UDP for Cluster Web Server

Jupyung Lee°, Kyu-Ho Park
EECS Dept. Korea Advanced Institute of Science and Technolohy

요 약

본 논문에서는 UDP/IP기반 SAN의 구현을 소개하였고 TCP/IP기반 SAN 및 로컬디스크와의 성능 비교를 통해 UDP/IP기반 SAN이 클러스터 웹서버에서 사용될 수 있는 가능성을 보였다. 실험결과 UDP/IP기반 SAN은 TCP/IP기반 SAN의 경우보다 약 20%정도 성능이 우수함을 볼 수 있다. 이는 UDP의 경우 TCP의 프로토콜 오버헤드가 없고 slow start의 영향을 받지 않으며 ACK를 기다릴 필요가 없기 때문이다.

1. 서 론

폭발적인 인터넷 사용자의 증가에 의한 웹서버의 병목현상을 해결하기 위해 그동안 PC기반 클러스터 웹서버 시스템에 대한 연구가 꾸준히 진행되어 왔다. PC기반 클러스터 웹서버는 전용 하드웨어를 사용한 고가의 웹서버보다 가격이 저렴하고 확장성이 우수하다.

PC기반 클러스터 웹서버에서 각각의 노드가 같은 디스크를 공유하면 디스크의 용량을 효율적으로 이용할 수 있으며 전위(Front End)의 웹 요청 분배기(dispatcher)의 웹 요청의 분배 문제가 간단해진다. 디스크는 파일서버를 통해서 공유될 수도 있지만, 이 경우 파일서버에서 병목현상이 발생할 수 있다. 이를 해결하기 위해 노드와 공유디스크를 SAN(Storage Area Network)을 통해 직접 연결하는 것이 효과적인 해결방안이 될 수 있다.

SAN은 Fibre Channel, Infiniband, TCP/IP 등의 프로토콜을 통해 구성할 수 있다.[1] Fibre Channel이나 Infiniband는 높은 대역폭을 가지므로 SAN을 구성하는데 적절하지만, 새로운 네트워크을 설치해야 하는 것이 단점이다. 반면에 TCP/IP를 이용해 SAN을 구성하는 경우에는 기존의 IP 네트워크를 그대로 이용하므로 대역폭은 상대적으로 낮지만, 새로운 네트워크를 필요로 하지 않는다.

TCP/IP를 이용해 SAN을 구성하기 위한 프로토콜은 iSCSI라는 이름으로 현재 개발중이다.[2] 그런데 LAN환경에서 SAN을 구성할 때는 protocol overhead가 낮은 UDP/IP를 이용하는 것이 더 효율적일 수 있다.

본 논문에서는 UDP/IP기반 SAN을 구현하였고 LAN 환경에서 TCP/IP기반 SAN 및 로컬디스크와 성능을 비교분석 하였다.

2. 관련연구

University of New Hampshire에서는 iSCSI프로토콜에 호환되는 initiator 및 target 드라이버를 개발하였다.[3] SAN을 구성하는 각 프로토콜을 비교분석하는 연구도 계속되고 있다.[1][4]

3. UDP/IP기반 SAN의 설계 및 구현

3.1. 전체구조

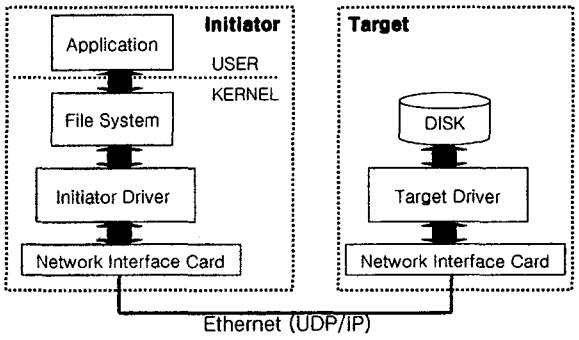

그림 1 UDP/IP기반 SAN의 전체 구조

UDP/IP를 기반으로 SAN을 구성하기 위해서는 노드와

공유디스크가 UDP/IP를 통해 SCSI command와 data를 주고받을 수 있어야 한다. 이를 위해서 노드는 공유디스크를 로컬디스크처럼 마운트(mount)할 수 있어야 한다. 마운트한 후에는 노드는 UDP/IP를 통해 공유디스크에 접근할 수 있다.

노드(initiator)와 공유디스크(target)의 전체구조는 그림1과 같다. 공유디스크에 대한 디바이스 드라이버가 initiator에 삽입되고 initiator의 디바이스 드라이버와 target의 디바이스 드라이버가 ethernet을 통해 연결된다. target을 initiator에 마운트한 뒤에 파일시스템을 설치하면 initiator의 사용자 영역 프로그램은 파일시스템, initiator 디바이스 드라이버를 거쳐 target에 접근할 수 있다.

모든 드라이버는 Linux 2.4.13에서 구현하였다.

3.2. initiator와 target의 구조

initiator와 target은 그림2와 같은 구조로 되어있다. tx_thread와 rx_thread는 ethernet으로부터 패킷을 보내거나 받는 역할을 한다. struct connection에는 UDP socket, 목적지 IP주소, 송신 버퍼, 수신 버퍼, SCSI command queue 등이 포함된다.

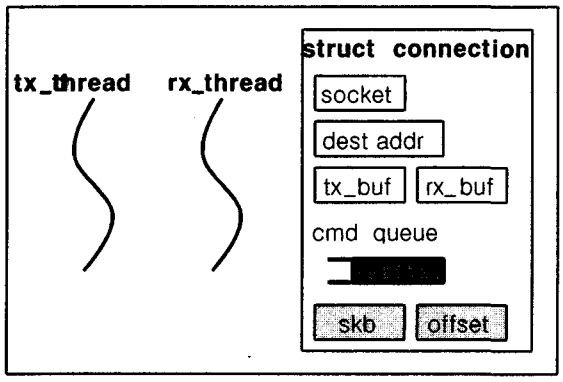

그림 2 initiator와 target의 구조

3.3. initiator와 target의 SCSI command/data 전송

initiator의 사용자 영역 프로그램에서 write 시스템콜이나 read 시스템콜을 호출하면 Initiator Driver와 Target Driver는 그림2와 같은 SCSI command/data를 전송하게 된다.

file system은 SCSI command를 command queue에 넣기 위해서 queuecommand()를 호출한다. queuecommand()는 tx_thread를 통해 SCSI command를 target으로 보낸다.

read command의 경우 target은 요청받은 block data를 보내고 response 메시지를 보낸다. initiator는 block data를 file system으로부터 넘겨받은 request_buffer 포인터로 카피해서 사용자 영역 프로그램이 block data를 넘겨받을 수 있게 한다.

write command의 경우 target은 block data를 받을 준비를 하고 rdy_to_xfer 메시지를 보낸다. 그러면 initiator는 사용자 영역 프로그램으로부터 넘겨받은 block data를 보내고 target은 response 메시지를 보낸다.

initiator가 response 메시지를 받은 후에는 command queue에서 해당 command를 삭제한다.

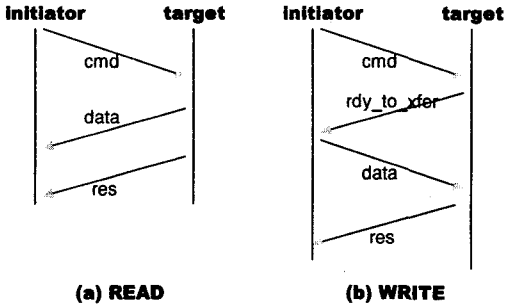

그림 3 Initiator - Target의 ISCSI command/data 전송

3.4. 에러 복구

UDP/IP는 TCP/IP와는 달리 패킷이 없어져도 재전송을 요청하지 않는다. 따라서 UDP/IP를 사용할 때는 상위 계층에서 패킷이 없어졌을 때 재전송을 요청할 수 있는 에러 복구 함수를 추가해야 한다.

본 구현에서는 SCSI 모듈에 포함된 에러 복구 함수를 사용하였다. SCSI command를 보내기 위해 queuecommand()를 호출할 때 SCSI 모듈에서는 해당 command에 대해 타이머를 켠다. 패킷의 손실 없이 모든 메시지 교환을 끝내고 command queue에서 command를 삭제할 때 해당 타이머를 끈다. 만일 패킷의 손실이 있어서 타임아웃이 되면 SCSI 모듈의 scsi_error_handler()가 호출되는데 이는 현재의 command에 대한 작업을 모두 중단하고 해당 command를 재전송하게 된다.

3.5. 다수의 initiator 연결

target은 다수의 initiator에 의해 공유되어야 하므로 하나의 target은 다수의 initiator에 연결 가능해야 한다.

본 구현에서는 initiator의 개수만큼 그림2의 target구조를 할당해 두고 서로 다른 포트번호를 통해 다수의 initiator와 연결하게 하였다.

3.6. 새로운 udp_recvmsg() 구현

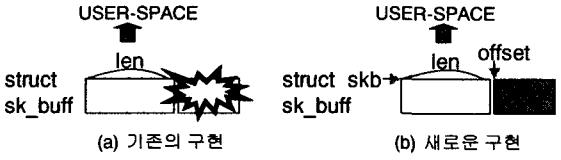

그림 4 udp_recvmsg()의 구현

기존의 udp_recvmsg()로 패킷을 받는 경우의 문제를 그림4(a)에 나타내었다. udp_recvmsg()가 호출되었을 때 받은 패킷의 길이보다 작은 데이터가 요청되었을 때는 해당영역을 사용자영역으로 카피한 다음 나머지 영역을 포함해서 전체 struct sk_buff에 대해 free()를 수행한다. 따라서 이 경우 나머지 영역의 데이터를 잃게 된다. 현재 udp_recvmsg()가 이렇게 구현되어 있는 이유는 UDP 계층에서는 버퍼링을 수행하지 않기 때문이다. 기존의 udp_recvmsg()를 사용하면 데이터를 잃지 않기 위해서 언제나 전체 패킷을 카피해와야 하기 때문에 구현하기에 불편하다.

새롭게 구현한 udp_recvmsg()를 그림4(b)에 나타내었다. 기존의 구현에 struct sk_buff의 포인터인 skb와 offset을 추가하였다. skb는 현재의 struct sk_buff를 가리키며 offset은 다음번에 udp_recvmsg()가 호출될 때 카피해야 하는 지점을 가리키고 있다. 사용자 영역으로 데이터를 카피한 후에는 항상 offset을 이동시킨다. offset이 struct sk_buff의 끝에 도달하면 현재의 struct sk_buff에 대해 free()를 수행하고 offset을 0으로 만들고 skb는 다음번 struct sk_buff를 가리키게 한다.

skb와 offset은 connection당 하나씩 필요하며 그림2의 initiator/target 구조에 추가해 두었다.

4. 실험결과

initiator는 세 대의 PC로, target은 한 대의 PC로 구성되어있다. 각 PC는 128MB의 메모리와 두 개의 450MHz Pentium III CPU를 가지고 있다. 각 PC는 100Mbps 이더넷으로 연결되어있다.

Benchmark는 PostMark[5]를 이용했다. PostMark는 정해진 범위의 파일을 정해진 갯수만큼 생성한 다음, 'Create file or Delete file', 'Read file or Append file' transaction을 정해진 횟수만큼 수행한다.

여기에서는 10Kbytes ~ 20Kbytes의 작은파일을 1000개를 생성한다음 50000번의 transactions을 수행하였다.

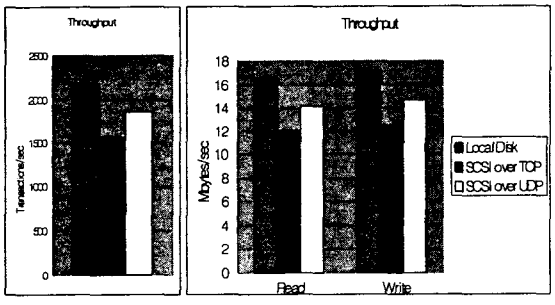

그림 5 Throughput 비교

UDP/IP기반 SAN의 경우 로컬디스크보다 성능은 떨어지지만, TCP/IP기반 SAN의 경우보다 약 20%정도 성능이 우수함을 볼 수 있다. 이는 UDP의 경우 TCP의 프로토콜 오버헤드가 없고 slow start의 영향을 받지 않으며 ACK를 기다릴 필요가 없기 때문으로 생각된다.

5. 결론 및 추후과제

본 논문에서는 UDP/IP기반 SAN의 구현을 소개하였고 TCP/IP기반 SAN 및 로컬디스크와의 성능 비교를 통해 UDP/IP기반 SAN이 클러스터 웹서버에서 사용될 수 있는 가능성을 보였다.

앞으로 공유파일시스템을 UDP/IP기반 SAN에 설치하고 실제로 클러스터 웹서버에서 공유디스크로 사용하기 위해 시스템을 안정화할 예정이다.

참고문헌

[1] K. Voruganti, P. Sarkar. An Analysis of Three Gigabit Networking Protocols for Storage Area Networks. IEEE International Conference on Performance, Computing and Communications, 2001

[2] J. Satran and et al. iSCSI. In http://search.ietf.org/internet-drafts/draft-ietf-ips-iscsi-04.txt. IETF, 2000

[3] A. Palekar, N. Ganaphthy, A. Chadda, R. Russell. Design and Implementation of a Linux SCSI Target for Storage Area Networks. Proceedings of the 5th Annual Linux Showcase & Conference, 2001

[4] H. Simitci, C. Malakapalli, V. Gunturu. Evaluation of SCSI over TCP/IP and SCSI over Fibre Channel Connections. Hot Interconnects 9, 2001

[5] J. Katcher, "PostMark: A Net File System Benchmark," Technical Report TR3022, Network Appliance, www.netapp.com/tech_library/3022.html