http://dx.doi.org/10.14400/JDPM.2014.12.1.379

고성능 저장장치를 위한 SAN최적화기법

이인선
신구대학교 IT미디어과

A SAN Optimization Scheme for High-Performance Storage System

In-Seon Lee
Dept. of IT Media, Shingu College

요 약 본 논문에서는 SAN(Storage Area Network)에 장착된 하드디스크를 단순히 고성능 저장장치로 교체하면 고성능 저장장치의 뛰어난 성능을 제대로 활용할 수 없음을 확인하고 원인 분석을 하여 고성능 저장장치를 위한 SAN의 성능 최적화기법들을 제안하였다. 먼저 고성능 저장장치에 맞지 않는 기존의 SAN 입출력경로에 존재하던 지연을 없애고, 저장장치 서버에서 입출력 요청들을 병렬 처리할 수 있게 하고, 소형 랜덤 입출력처리의 성능 향상을 위해 SAN에 연결된 초고속 네트워크에 사용되는 기존의 전송 프로토콜에 시간적 병합 기법을 추가하였다. 제안한 기법들의 우수성을 입증하는 방법으로 실제로 고성능 저장장치를 장착한 SAN에 최적화기법들을 구현하였으며, 다양한 입출력 데이터로 실험한 결과 30%이상의 입출력 지연시간 절감과 200%이상의 성능 향상을 확인하였다.

주제어 : 고성능저장장치, 저장장치분야 네트워크, 성능 최적화, 시스템 구현, 초고속네트워크 프로토콜

Abstract We noted that substituting hard disk with high-performance storage device on SAN did not immediately result in getting high performance. Investigating the reason behind this leaded us to propose optimization schemes for high-performance storage system. We first got rid of the latency in the I/O process which is unsuitable for the high-performance storage device, added parallelism on the storage server, and applied temporal merge to Superhigh speed network protocol for improving the performance with small random I/O. The proposed scheme was implemented on the SAN with high-performance storage device and we verified that there were about 30% reduction on the I/O delay latency and 200% improvement on the storage bandwidth.

Key Words : High-Performance Storage, Fast Storage Device, Storage Area Network, Performance Optimization, System Implementation, Superhigh Speed Network protocol

1. 서론

오늘날 서버 시스템 환경은 급속도로 증가하는 데이터를 효율적으로 처리 및 저장하기 위해 네트워크로 연결된 다수의 컴퓨터들을 클러스터로 구성하거나, 분산시스템 또는 SAN(Storage Area Network : 저장장치분야

Received 1 December 2013, Revised 2 January 2014
Accepted 20 January 2014
Corresponding Author: In-Seon Lee(Shingu College)
Email: inseon@shingu.ac.kr

ISSN: 1738-1916

© The Society of Digital Policy & Management. All rights reserved. This is an open-access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.otg/licenses/by-nc/3.0), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

네트워크) 환경을 구축하여 보다 효율적인 서버 시스템을 구현하고 있다.

CPU 코어 수가 점점 늘어나면서 컴퓨팅파워가 높아지고 초고속 네트워크의 성능이 빠르게 발전하고 있는데 비해 저장장치 성능의 발전 속도는 그에 훨씬 미치지 못하고 있다. 그로 인해 전체 시스템에서 저장장치시스템

은 항상 성능상의 병목이 되고 있으며, 이러한 문제를 해결하기 위해 고성능 저장 장치(High-Performance Storage, Fast Storage Device) 성능 개선에 대한 시장의 요구가 증가하고 있는 추세이다. HDD(Hard Disk Drive) 기반의 저장장치 시스템은 기계적 움직임으로 동작하는 하드웨어의 속성상 성능의 한계가 존재하며, 고성능 저장장치시스템에 대한 시장의 요구를 만족 시킬 수 없다. 그 대안으로 플래시기반의 고성능 SSD(Solid State Drive)나 DRAM기반의 SSD 혹은 메모리를 랜덤 저장장치로 사용하는 기술 등이 대안으로 떠오르고 있으며, PCRAM(Phase Change Random Access Memory,상변화 메모리), FeRAM(Ferroelectric Random Access Memory, 강유전체 램), MRAM(Magnetic Random Access Memory, 자기메모리)등의 차세대 저장 장치에 대한 연구도 활발히 진행되고 있다[3,8].

하지만 단순히 기존의 HDD기반의 저장장치 시스템 환경에 고성능 저장장치를 교체하는 것만으로는 고성능 저장장치 시스템의 성능을 제대로 활용할 수 없다는 것은 자명한 사실이며, 그에 따른 성능을 위한 최적화 방안이 필요하여 이미 이러한 고성능 저장장치에 대한 연구가 다양한 시스템 레벨에서 이루어지고 있다[4,5,18,22].

SAN은 호스트 컴퓨터와 저장장치간의 데이터버스를 초고속 네트워크로 구성한 시스템으로 저장 장치는 네트워크에 직접 연결되며, 표준 네트워크 프로토콜을 통해서 여러 호스트 컴퓨터들과 통신한다. 일반적으로 SAN 환경에서의 호스트 컴퓨터를 *Initiator*, 저장장치 서버를 *Target* 이라 부른다. 이러한 SAN은 보통 대형 및 중소형 기업환경에서 효율적인 서버시스템 구축을 위해 널리 사용되고 있으며, 기가비트 이더넷, 광 채널, 인피니밴드 등의 초고속 네트워크를 이용하여 구축할 수 있다[10,13,14].

본 논문에서는 먼저 하드디스크를 장착한 SAN환경을 단순히 고성능 저장 장치로 교체한 경우 고성능 저장장치의 뛰어난 성능을 제대로 활용할 수 없다는 사실을 확인하였다. 고성능 저장장치는 PCI-E 인터페이스를 이용하는 DRAM기반의 SSD를 이용하였다[21]. 기존 SAN 시스템을 구축하기 위해 SAN 환경을 구축해주는 개방형 프로그램인 SCST를 사용하였고[20], 네트워크는 최근에 가장 많이 쓰이는 초고속네트워크 프로토콜인 RDMA를 이용하여 데이터 전송을 수행하는 인피니밴드 초고속 네트워크 장비를 사용하였다[15].

다양한 입출력 패턴에 대해 실험한 결과 소형 랜덤 입출력의 경우 '읽기'와 '쓰기' 모두 초고속 네트워크 장비를 사용했음에도 불구하고 고성능 저장 장치의 성능이 현저하게 떨어짐을 확인할 수 있었다.

성능 저하의 첫 번째 원인은 기존의 블록 입출력 경로가 가지는 긴 지연시간이다. 기존의 SAN시스템에서는 보통 SCSI 레벨에서 저장장치 가상화를 수행한다[1]. SCST 역시 SCSI 중간 레벨에서 수행되는 커널 프로그램이며, 인피니밴드의 경우 SRP(SCSI RDMA Protocol)을 이용하여 RDMA를 통한 데이터 전송을 수행하게 된다. 실제로 낮은 지연시간을 가지는 고성능 저장장치 장비에 대해서 SCSI 레이어는 입출력 경로를 불필요하게 길게 만들어 그 자체가 오버헤드가 될 수 있음을 증명하는 연구도 있다[4]. 두 번째 원인은 병렬처리의 부재이다. 일반적으로 고성능 저장장치의 경우 장치 레벨에서 입출력에 대한 병렬성을 높이고 있다. 하지만 기존의 SAN 환경을 분석해보면 이러한 처리 과정에서 대부분 순차 수행, 특히 장치 입출력의 경우 순차적으로 처리되는 경우가 많았다.

본 논문에서는 위에서 분석한 기존 하드디스크 기반 SAN 환경의 문제점분석을 바탕으로 이를 개선할 수 있는 새로운 SAN시스템 성능을 위한 최적화기법들을 제안한다.

먼저 SCSI계층으로 인해 입출력 경로가 길어지는 문제를 해결하기 위해 SCSI계층을 완전히 제거하는 작업을 수행하였다. 그리고 기존의 입출력 스케줄러를 없애고 단순한 형태의 새로운 입출력 스케쥴러를 구현하였다. 또한 *Target*쪽에서 입출력 요청을 처리하는 일련의 모든 과정들을 스레드-풀을 이용한 멀티 스레드 환경으로 처리되도록 하여 병렬성을 높였으며, 특히 실제 고성능 저장장치에서 수행되는 장치 입출력이 동시에 여러 개 수행되도록 하여 장치의 대역폭을 최대한 활용하게 구현하였다. 마지막으로 RDMA를 이용한 *Initiator*와 *Target*

간의 데이터 전송방식을 소형 입출력에도 최적화되도록 하였다. RDMA를 이용한 데이터 전송 프로토콜은 분명히 뛰어난 성능을 보여준다. 하지만 소형 랜덤 입출력의 경우 기존의 SAN환경에서 사용되는 RDMA전송 방식은 비효율적인 측면이 있다. 상대적으로 느린 저장 장비를 사용했을 경우 보이지 않았던 소형 입출력에 대한 처리 부하들이 고성능 저장 장비를 사용하면서 드러나는 것이다. 이에 본 논문에서는 소형 랜덤 입출력의 성능을 높이기 위한 시간적 병합 기술[22]을 적용하였다.

2장에서는 오늘날 사용되고 있는 고성능 저장장치들에 대해서 살펴보고, 하드디스크 기반의 블록 입출력 경로와 인피니밴드를 이용한 기존 SAN환경을 분석한다. 3장에서는 제안하는 최적화기법들의 동작상의 원리에 대해 상세하게 기술하며, 4장에서 구현의 관점에서 *Initiator*와 *Target* 각각의 세부적인 구현사항에 대해 설명한다. 5장에서는 고성능저장장치를 장착하고 제안한 최적화기법들을 구현한 SAN시스템에 다양한 실험을 실시하여 어느 정도 성능이 향상되는지 보인 후 6장에서 본 논문의 성과에 대해 정리하고 향후 연구계획에 대해서도 기술한다.

2. 관련 연구

2.1 고성능 저장장치 고찰

오늘날 고성능 저장장치시스템에 대한 요구가 증가하고 있지만, 기존의 저장장치시스템의 주류를 이루는 하드디스크 기반의 저장장치들은 이러한 시장의 요구를 만족시키지 못한다. 그 개선책으로 플래시기반의 SSD나 DRAM 기반의 솔루션들이 대두되고 있다.

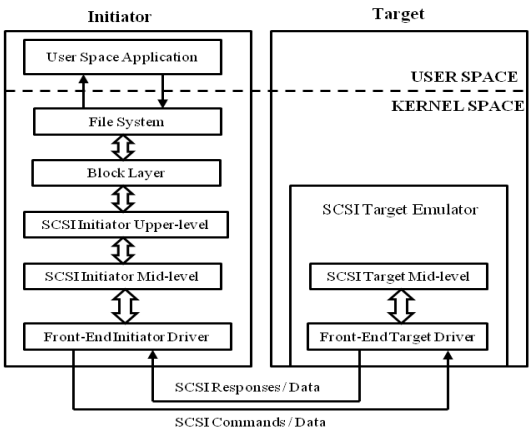
플래시기반의 SSD 중 Fusion-io 사의 ioDrive 시리즈는 대표적인 고성능 저장장치로[9] 플래시의 비휘발성 속성을 가지면서 단축된 지연시간 및 향상된 대역폭을 가지고 있다. 하지만 플래시자체가 가지는 문제점인 짧은 수명이라든지 '쓰기' 성능 저하 현상 등은 여전히 남아 있다.

DRAM 기반의 해결책으로는 먼저 메인메모리 자체를 저장장치로 사용하는 방식들이 있다. 수천 개의 일반 서버 컴퓨터들을 초고속 네트워크로 연결하여 각 컴퓨터들의 메인 메모리에 모든 데이터 정보를 저장하거나[12,17], 또는 하나의 컴퓨터에 다량의 DRAM을 꽂아 사용하기도 한다[19]. 이 방식들은 DRAM의 휘발성이라는 물리적인 한계 때문에 추가적으로 이중화나 백업등의 기술이 강조되고 있다.

DRAM 기반의 또 다른 해결책으로 DRAM 기반의 SSD가 있다. 이는 PCI-E 채널을 통해 호스트 컴퓨터에 연결되고, 배터리 지원시스템으로 전원을 공급받고 고장이 발생하면 추가적인 배터리 전력을 이용하여 데이터가 비휘발성 저장장치에 백업되도록 하여 DRAM의 구조적 문제를 해결하고 있다. 또한 뛰어난 대역폭과 단축된 지연시간을 보여 주고 있으며, 순차 또는 랜덤 방식에 상관없이 일정한 입출력성능을 유지한다. 따라서, 본 논문에서도 제안한 최적화기법들을 구현하는 SAN의 고성능 저장장치로 DRAM기반의 SSD를 장착하였다.

2.2 하드디스크 기반 SAN 입출력경로

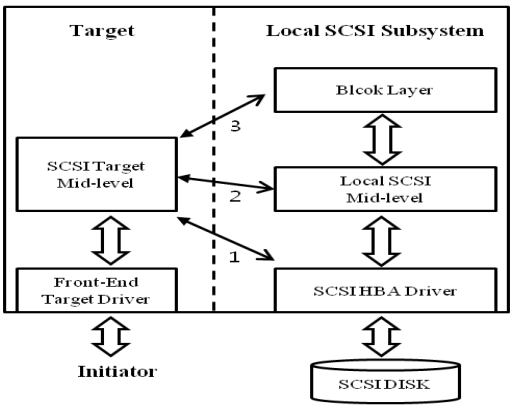
[Fig. 1]은 기존의 하드디스크 기반의 SAN환경에서의 입출력경로를 개념적으로 도식화하여 보여준다.



[Fig. 1] Traditional SAN I/O Path

이 그림에서 알 수 있듯이 SAN 환경의 *Initiator*에서 수행되는 입출력은 일반적인 로컬 SCSI 저장장치에 대한 리눅스 입출력 서브시스템의 경로와 상당히 유사하다. 어플리케이션계층에서 시작된 입출력요청은 파일시스템을 거쳐 블록계층으로 전송된다. 파일시스템에서 블록계층으로의 입출력수행 요청은 블록구조단위로 전송된다. 이 블록은 엘리베이터 알고리즘 등을 수행하여 이전에

이미 들어와 대기하고 있는 다른 입출력 요청과 합쳐질 수 있다. 블록은 요청 구조를 구성하게 되고 요청 큐에 삽입되어 입출력 스케줄러에 의해 처리되기를 기다린다. 이후 입출력 스케줄러에 의해 인출된 요청은 SCSI계층으로 전송되고, SCSI 중간레벨에서는 SCSI 장치에 대한 입출력수행을 위해 요청 구조를 SCSI명령어로 바꾼다. 이후 일반적인 로컬 SCSI 장치에 대한 입출력이라면 구성된 SCSI 명령어를 해당 SCSI 장치의 호스트 버스 어댑터(HBA) 드라이버에 전송하고, SAN입출력인 경우에는 SCSI명령어는 네트워크를 통해 *Target*으로의 데이터 전송을 수행할 Front-End Target *Initiator* 드라이버로 전달된다.



[Fig. 2] Target options to interact with local SCSI subsystem

[Fig. 2]는 *Initiator*로부터 전달받은 명령어와 데이터를 *Target*에서 입출력 처리하는 흐름을 나타낸 것이다. *Target*에서 입출력 수행을 위해 리눅스 입출력 서브시스템과 상호작용을 할 때 3가지 옵션이 존재한다. 첫 번째 옵션은 SCSI HBA드라이버와 직접적으로 입출력을 수행하는 방식으로 완성된 SCSI명령어를 구성하여 전달해야 하며, SCSI 디스크에 대한 입출력을 수행할 때 지연시간이 가장 짧다. 두 번째 옵션은 SCSI 중간레벨과의 상호작용으로 해당 레벨에서 SCSI 명령어를 구성할 수 있도록 그에 대한 프로세싱을 미리 수행해야 한다. 실제로 가장 많이 쓰이는 방식은 세 번째 옵션으로 블록계층에 대한 장치 입출력을 수행하며, 커널의 페이지 캐시를 활용하여 비동기적인 입출력을 수행하는 파일 입출력방식과 페이지 캐시를 사용하지 않고 블록을 구성하여 직접 입출력을 수행하는 블록 입출력방식으로 나뉜다.

일반적으로 기존의 하드디스크기반의 저장장치는 파일 입출력 방식을 주로 사용하지만 고성능 저장장치의 경우 페이지 캐시를 사용하는 것이 오히려 오버헤드가 되어 성능이 떨어지는 경우도 있다.

2.3 인피니밴드를 이용한 SAN구축

현재 리눅스시스템에서 인피니밴드 장비의 RDMA전송 기능을 이용하여 SAN을 구축하는 방법은 두 가지가 있다. 첫 번째는 SCSI RDMA Protocol (SRP)를 이용하는 것이고[20], 또 다른 하나는 iSCSI extensions for RDMA (iSER) 프로토콜을 이용하는 것이다[16]. 이 둘은 모두 인피니밴드를 지원하고 SCSI계층에서 RDMA 데이터 전송 방식을 이용하여 SAN 환경을 구축할 수 있다는 점에서는 동일하지만 실제로 어떤 방식을 사용할 지 결정하기에 앞서 장단점을 먼저 따져보아야 할 것이다.

우선 SRP를 통한 SAN구축은 SCST를 통해서 가능하며, iSER의 경우 *Target* 프로젝트에 의해서 구현 가능하다. SRP는 iSER에 비해 SAN 환경을 구축하기 수월하며, SCST의 SRP *Target*의 경우 커널영역에 구현되어 있어 사용자영역에 구현되어 있는 iSER *Target*보다 높은 대역폭과 짧은 지연시간을 가진다. 반면 iSER은 iSCSI 인터페이스를 사용하기 때문에 *Target*-발견기능을 이용하여 특정 *Target*에 로그인한다거나 패스워드기반의 사용권한을 준다거나 하는 다양한 운영 기능들을 이용할 수 있다[16,20].

본 논문에서는 성능의 최적화에 초점을 맞추고 있기 때문에 높은 대역폭과 짧은 지연시간을 가지는 SRP를 이용하여 성능비교를 위한 기존 SAN시스템을 구축하는데 사용하기로 한다.

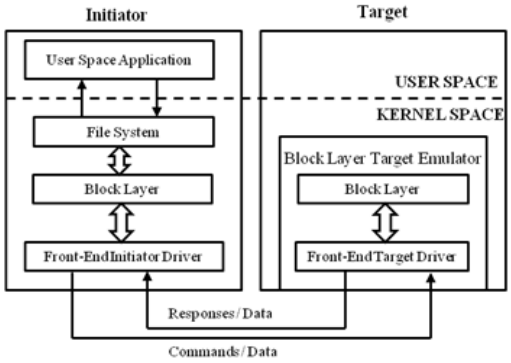
3. 최적화기법 설계

3.1 입출력경로에서 지연 제거

1장에서 분석한 바와 같이 고성능 저장장치에서 SCSI계층은 성능의 병목으로 작용하나 기존의 SAN 최적화 방안들은 보통 SCSI계층에서 저장장치 가상화를 수행하였다. 본 논문에서는 이러한 하드 디스크방식의 SCSI계

층을 제거하고 [Fig. 3]과 같이 블록계층에서 저장장치 가상화를 구현하였다. 리눅스 입출력 서브시스템의 입출력 스케줄러는 기본적으로 하드 디스크를 가정한 스케줄링 방식을 사용하거나, 또는 병합을 위해 입출력을 바로 처리하지 않고 지연시켜 고성능 저장장치시스템에서는 오버헤드로 작용할 수 있는 불필요하고 복잡한 작업들을 수행하고 있다. 스케줄러 정책을 "NOOP"방식으로 변경함으로써 오버헤드를 어느 정도 감소시킬 수 있으나 여전히 병합을 위한 불필요한 대기를 수행하거나 복잡한 스케줄링과정을 거치는 등 고성능 저장장치에 적합하지 않은 작업들을 수행하고 있다[7].

이에 본 논문에서는 기존의 입출력 스케줄러를 사용하지 않는다. 이는 기존 입출력 스케줄링에서 발생하는 오버헤드를 제거할 뿐만 아니라 최적화를 위해 블록 구조를 기반으로 하는 전송 프로토콜을 사용하기 위해 요청구조기반으로 동작하는 입출력 스케줄러를 사용하는 것은 그 자체가 오버헤드가 된다. 그 대신 간단한 형태의 요청 큐를 구현하고 3.3절에서 설명할 시간적 병합기법을 이용한 인출과정을 수행함으로써 최대한 신속하게 입출력을 처리할 수 있게 구현한다.

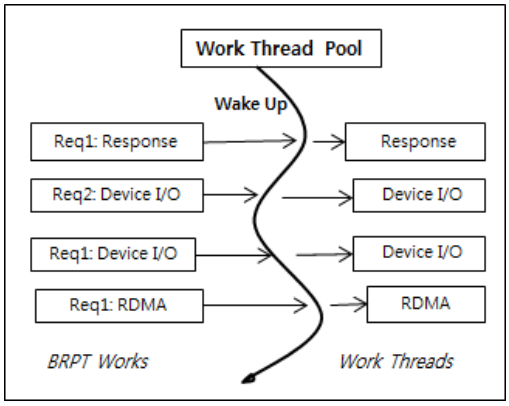


[Fig. 3] New SAN I/O Path without SCSI Layer

3.2 Target에서의 병렬처리 극대화

고성능 저장장치를 장착한 SAN 시스템의 성능 최적화에 적용될 두 번째 아이디어는 *Target*에서 입출력요청을 처리하는 과정에 병렬성을 최대로 증대시키는 것이다. 일반적으로 고성능 저장장치의 경우 장치 레벨에서 입출력에 대한 병렬성을 높이고 있다. 이는 여러 개의 입출력을 효율적으로 처리하여 뛰어난 대역폭 성능을 구현하기 위함이다. 하지만 기존의 SAN최적화 방안들은 이러한 고성능 저장장치의 특성을 잘 이용하지 못하고 있다. RDMA전송방식을 이용하는 SAN 환경에서 *Target*이 수행하는 입출력요청에 대한 처리 과정은 다음과 같이 크게 4단계로 분류할 수 있다. 1) *Initiator*로부터 도착한 명령어분석 2) RDMA 수행 3) 장치 입출력 수행 4) *Initiator*에게 응답 전송.

또한 서로 다른 입출력 요청에 대해서 수행하는 이러한 일련의 처리 과정들은 서로 독립적이므로 동시에 처리될 수 있는 작업들이다. 특히 장치 입출력의 경우 동시에 처리될 경우 고성능 저장장치의 뛰어난 동시 입출력 처리 능력을 이용할 수 있어 성능이 향상된다. 따라서 본 논문에서 제안하는 고성능 저장장치 시스템을 위한 SAN 최적화설계에서는 [Fig. 4]와 같이 위에서 언급한 *Target*에서 수행되는 입출력 처리 과정들을 BRPT 워크라 정하고 스레드-풀 형태로 관리 및 동작하는 워크-스레드들을 통해서 동시에 처리과정을 수행하게 설계한다.



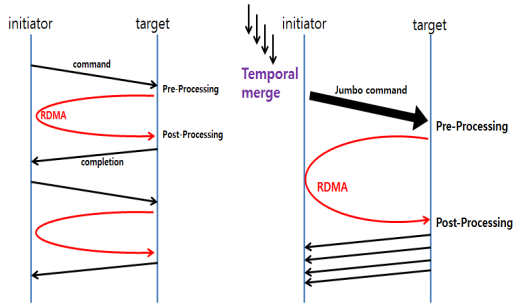
[Fig. 4] Parallel Processing multiple I/O requests with Work-Threads

3.3 시간적 병합을 이용한 전송방식

마지막으로 RDMA를 사용하여 데이터 전송을 하는 *Initiator*와 *Target*간의 데이터 전송방식을 고성능저장장치의 특성을 고려한 성능 최적화에 초점을 맞춰 개선한다. RDMA를 이용한 데이터 전송은 뛰어난 성능을 보여주나, 소형 랜덤 입출력이 많은 경우에는 기존의 RDMA 전송방식을 고성능 저장장치에 그대로 적용하는

데는 성능에 문제가 생긴다. [Fig. 5]의 왼쪽은 '쓰기'요청에 대한 기존의 RDMA 전송 방식을 도식화한 것이다. *Target*에서는 *Initiator* 로부터 전달받은 명령어를 바탕으로 사전-프로세싱과정을 거쳐 RDMA 데이터 전송을 수행하고, 이후 RDMA 전송이 완료되면 사후-프로세싱 과정을 거친다. 대형 입출력의 경우 데이터전송시간이 입출력대기시간에 비해 현저하게 길기 때문에 이러한 프로세싱 오버헤드가 충분히 가려지게 되지만, 소형 입출력의 경우 요청당 데이터전송시간이 작기 때문에 프로세싱 오버헤드가 두드러지게 된다.

이를 개선하기 위해 시간적 병합을 도입한다. 시간적 병합은 특정 시간 동안 들어오는 입출력 요청들을 공간적 연속성에 상관없이 병합하여 전송하는 방법으로 소형 입출력이 많은 경우 저장장치 장비의 성능을 최대한 활용할 수 있는 기법으로 소개된 바 있다[22]. 본 논문에서는 이 기법을 *Initiator*와 *Target*간의 RDMA 전송 프로토콜에 적용하였다.



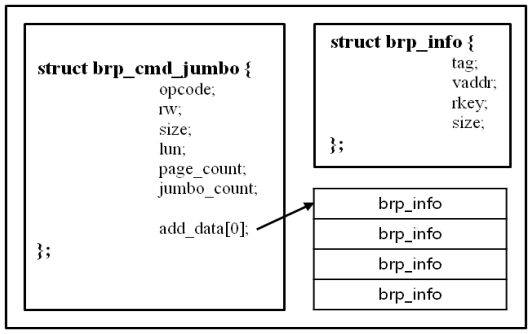
[Fig. 5] Existing RDMA transfer vs Temporal Merge applied RDMA transfer

[Fig. 5]의 오른쪽 그림은 시간적 병합기법을 적용한 RDMA 전송 방법을 도식화한 것으로 *Initiator*쪽에서는 *Target*으로 전송하기 위한 명령어를 구성할 때 서로 다른 입출력 요청들을 공간적 연속성에 상관없이 하나로 병합하여 점보명령어를 생성하여 전송한다. 이후 점보명령어에 통합되어 저장된 다수의 입출력 데이터 정보를 바탕으로 사전-프로세싱을 수행한다. 또한 RDMA에서는 scatter/gather DMA를 지원하고 있으며, 이 방식을 통해 서로 연속적이지 않은 메모리 영역에 존재하는 데이터들을 한번의 RDMA 수행으로 전송할 수 있다. RDMA 전송이 완료되면 그에 대한 사후-프로세싱과정을 수행하게 된다. 이후 장치 입출력을 수행하고 이에 대한 완료 응답은 완료되는 순서대로 *Initiator*에 즉각적으로 전송하여 입출력요청에 대한 응답성을 높인다.

4. 구현

4.1 Block-RDMA Protocol(BRP) 구현

하드디스크기반의 SAN 시스템에서 사용하던 SCSI계층과 SCSI명령어 기반의 RDMA 전송 프로토콜대신에 보다 빠른 입출력경로를 구현하고 *Initiator*에서 입출력요청들을 시간적으로 병합하여 전송하기위해 블록계층과 블록구조 기반의 시간적 병합을 수행하는 전송 프로토콜을 구현하고 이 프로토콜을 Block-RDMA Protocol(BRP)이라 명명한다.



[Fig. 6] Block RDMA Protocol structure

[Fig. 6]은 BRP구조를 보여주고 있다. *Initiator*에서는 블록구조의 정보들을 바탕으로 'brp_cmd_jumbo' 구조를 구성하여 *Target*에 전송한다. 'brp_cmd_jumbo'의 구성을 살펴보면 먼저 *Target* 쪽에서 새로운 블록구조를 생성하는데 필요한 필수적인 'rw', 'size', 'page_count' 등의 정보들을 가지고 있다. 또한 'brp_cmd_jumbo'에서는 *Target*쪽에서 RDMA 데이터 전송을 수행하기 위해 필요한 정보들도 가져야 한다. 이것들은 'add_data'멤버에 저장되며, RDMA 수행을 위한 'dma address'와 'remote key' 등을 담고 있는 'brp_info'구조를 'add_data'에 동적 배열 멤버로 저장한다. 그리고 'jumbo_count'멤버 변수는 'brp_cmd_jumbo'에 몇 개의 서로 다른 입출력 요청 정보가 병합되었는지를 나타내는 병합된 입출력 요청 블록의

개수이다. 개수만큼 'brp_info' 구조가 'add_data' 멤버에 저장되고, 각각은 'brp_info' 구조의 태그 변수에 의해 구분되어진다. 이후 *Target* 쪽에서의 입출력 수행이 완료되면 *Initiator*는 입출력 요청에 대한 응답은 각각 따로 받게 되며, *Target*으로부터 전달받은 응답에 저장된 태그 정보를 이용하여 그에 대응하는 입출력 요청블록을 완료시킨다.

시간적 병합은 기본적으로 소형 랜덤 입출력에 대한 성능 향상을 위해 적용된 최적화기법이다. 대형 데이터의 경우 병합을 수행하지 않아도 네트워크 대역폭을 충분히 활용할 수 있으므로 소형 입출력에 대해서만 시간적 병합을 수행하도록 하고, 본 논문에서는 소형 입출력의 기준을 하나의 페이지 크기 데이터로 정한다. 이 외에 인피니밴드와 관련하여 RDMA를 수행하기 위해 필요한 자원 할당 및 데이터 전송 수행은 관련 API들을 이용하였고[2], 기존의 SRP 소스를 참조하여 구현하였다.

4.2 BRP을 위한 Target 시스템 커널 수정

BRPT(BRP *Target*)은 먼저 [Fig. 4]에서 설명한 대로 *Target*에서 입출력요청을 병렬처리하기 위해 각각의 입출력 요청은 'brp_work'라는 구조체 형태로 저장되며, 이러한 'brp_work'들은 대기-리스트에 삽입되어 이후 스레드-풀 형태로 관리되는 워크-스레드에 의해 인출되어 병렬 처리되도록 수정하였다. 'brp_work'는 3가지 워크타입을 가지게 구현하였으며 각각의 타입에서 수행되는 연산은 '읽기/쓰기'타입에 따라 조금씩 차이가 있다. 3 가지 타입의 워크들은 스레드-풀에서 관리되는 다수의 워크 스레드에 의해 병렬로 수행되며, 해당 워크 스레드가 깨어나 수행되는 과정은 'leader-follower model' 방식을 따라 수행되도록 구현하였다[6].

그다음에는 소규모 랜덤 입출력처리 성능 최적화를 위해 시간적으로 병합되어 *Initiator*로부터 전송되어 온 점보 명령어안에 있는 입출력요청들을 병렬처리하기 위해 필요한 부분을 수정하였다. 하나의 점보명령어에는 다수의 입출력 요청들이 병합되어 저장되어 있으므로 이를 기존의 SAN에서 사용하던 모듈을 그대로 적용하면 각각에 대한 입출력 요청이 순차적으로 처리되어 오히려 병렬성을 저하시키게 된다. 입출력 요청이 시간적으로 병합되는 경우는 요청 데이터의 크기가 한 개 페이지 크기일 때이므로 이런 점보명령어에 대비하여 필요한 자원들을 사전에 미리 할당해 두었다가 점보명령어가 도착하면 필요한 자원들을 바로 할당하여 병렬처리가 가능하게 수정하였다.

5. 성능평가

본 논문에서는 두 개의 컴퓨터를 *Initiator*와 *Target*으로 구성하여 SAN을 구축하였으며, 구축한 SAN의 H/W 스펙은 <Table 1>과 같다.

〈Table 1〉 Implemented SAN H/W spec.

parameter	value
CPU	Intel Xeon E5630 2.53GHz quad core CPUs (total 8 cores)
Main Memory	*Initiator* 8GB, *Target* 16GB
kernel version	Linux 2.6.32
Target High-performance storage device	64GB(=8GB of DDR2*8) DRAM-Based SSD
Superhigh speed network device	MHQH18B-XTR (40Gb/s, ConnectX-2 Infiniband device, Mellanox Co.)

먼저 입출력 지연시간이 얼마나 단축되는지를 확인하기 위해 기존의 SAN을 구축하는 방식인 SRP-SRPT와 본 논문에서 제안한 최적화기법들을 적용하여 구현한 BRP-BRPT의 '읽기/쓰기' 입출력 지연시간을 측정하여 그 결과를 <Table 2>에 정리하였다.

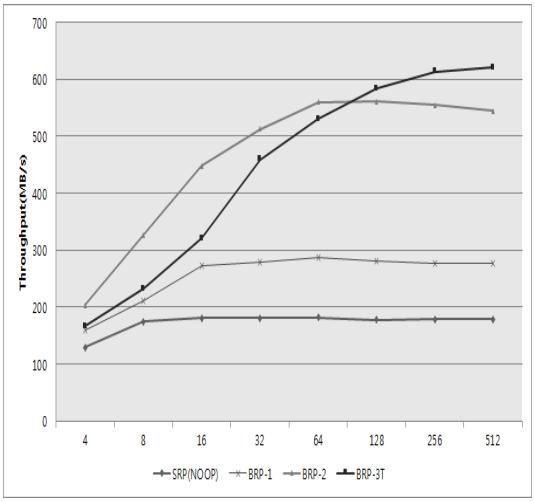
〈Table 2〉 I/O path Latency Comparison

I/O Type	SRP-SRPT (usec)	BRP-BRPT (usec)	Latency Reduction Ratio
Read	63 (51)	43 (31)	-31.7 (-39.2)%
Write	75 (62)	54 (41)	-28 (-33.8)%

괄호 안의 값은 전체 지연 시간중에서 디바이스 입출력지연시간을 뺀 값으로 SAN 입출력경로에서의 지연시간만을 비교할 수 있다. <Table 2>를 보면 '읽기'는 약 31.7(39.2)%의 지연시간이 줄었고 '쓰기'의 경우는 28(33.8)%정도 줄었다. 이는 SCSI레이어의 제거와 스케

줄러간소화 작업을 통해 입출력경로가 30%이상 감소되었음을 보여준다.

다음에는 구현한 최적화기법들의 성능분석을 위해 FIO 마이크로-벤치마크 툴을 이용하여[11], 다양한 입출력 패턴에 대해 실험을 실시하였다. 사전실험을 통해 고성능 저장장치 SAN에 리눅스 시스템 디폴트 입출력 스케줄러인 CFQ방식을 사용하는 경우 고성능저장장치에 큰 오버헤드로 작용하여 심각한 성능 저하가 생기므로 기존 SAN구축 솔루션인 SRP-SRPT방식에 NOOP 입출력 방식을 적용한 SRP(NOOP)방식을 본 논문에서 제안한 방식들과 성능비교하기로 하였다. 본 논문의 최적화 방안들은 구현한 정도에 따라 'BRP-1', 'BRP-2', 'BRP-3T'로 구분한다. 'BRP-1'은 SCSI계층을 없애고 입출력스케줄러를 수정한 1단계 최적화만을 적용한 것이고, 'BRP-2'는 'BRP-1'에 추가적으로 *Target*의 입출력과정에 병렬성을 추가한 것이며, 'BRP-3T'는 'BRP-2'에 입출력상황에 무관하게 항상 시간적 병합을 적용한 것이다.



[Fig. 7] Performance analysis with # of threads on random write pattern

[Fig. 7]은 스레드 수에 따른 각 최적화기법들의 성능 변화를 측정한 그래프이다. 실험은 소형 랜덤 쓰기 패턴에 대해 진행하였으며, 정확성을 위해 메모리 버퍼에 의한 이득을 반영하지 않는 랜덤 입출력 방식으로 수행하였다. 예상한 대로 기존 방식인 'SRP(NOOP)'보다 입출력 지연을 없앤 'BRP-1'이 성능이 향상되었음을 알 수 있으며, 입출력 양에 따른 처리는 동일하므로 스레드수가 16이상에서는 둘 다 성능의 변화가 없음을 알 수 있다. 그러나 'BRP-2'은 입출력 량이 많은 경우 스레드 수에 따라 병렬성이 증가하므로 스레드수가 증가함에 따라 성능이 대폭 증가하는 것을 확인할 수 있다. 시간적 병합의 성능을 분석하기 위해 'BRP-2'와 'BRP-3T'를 비교한 결과 집중적인 입출력이 수행되지 않는다면 오히려 'BRP-3T'의 성능이 낮을 수 있다. 이는 시간적 병합에 의해 합쳐진 여러 개의 요청들이 *Target*쪽에서 순차적으로 처리되어 병렬성을 떨어트리기 때문이다. 그러나 [Fig. 7]에서 보다시피 입출력 양이 많아져 스레드가 128 이상인 경우부터 시간적 병합을 하는 'BRP-3T'방식이 'BRP-2'보다 성능이 나아짐을 알 수 있다. 이는 입출력 량이 많아지면 많은 수의 스레드가 시간적으로 병합된 요청들을 동시에 수행하여 성능을 향상시키기 때문이다.

6. 결론 및 향후 연구계획

본 논문은 고성능 저장장치시스템의 뛰어난 성능이 최대한으로 발휘될 수 있도록 SAN최적화기법에 대한 연구를 진행하였다. 먼저 고성능 저장장치 장비를 기존의 SAN 솔루션을 이용하여 구축하는 경우 심각한 성능 저하가 발생하는 것을 확인하였다. 이에 본 논문에서는 그에 대한 원인을 분석하고 문제를 해결할 수 있는 새로운 SAN 최적화기법들을 제안하였다.

고성능 저장장치시스템을 기반으로 하는 SAN환경을 위한 첫 번째 최적화는 고성능 저장장치의 단축된 지연시간 특성에 맞는 짧은 입출력 경로를 갖게 한 것이다. 두 번째 최적화는 고성능 저장장치 서버의 내부 병렬성을 최대한 이용하는 병렬 입출력 처리 방안을 개발하여 서버에서 입출력요청을 처리하는 모든 과정들을 워크-스레드에서 동시에 처리할 수 있도록 하였다. 세번째 최적화는 *Initiator*와 *Target*을 연결하는 초고속 네트워크 환경에서 많이 쓰이는 RDMA 데이터 전송 방식에 시간적 병합기법을 추가하여 많은 소형 입출력 요청들을 공간적 연속성에 상관없이 병합시켜 점보 명령어를 구성한 후 RDMA를 수행하도록 하는 최적화를 구현하였다. 이러한 시간적 병합기술을 통해서 RDMA 전송 횟수를 감소시키고 네트워크 오버헤드를 줄여 전체 입출력 성능이

향상될 수 있도록 한 것이다.

모든 최적화 기법들을 실제 고성능 저장장치를 장착한 SAN에 구현하여 다양한 입출력에 대해 실험을 진행하였으며, '읽기'의 경우 약 39.2%, '쓰기'는 33.8%의 입출력지연시간을 단축시킬 수 있었고, 저장장치 대역폭 측면에서는 시간적 병합의 효과를 최대한으로 가지는 소형 입출력의 경우 200%이상의 성능 향상을 확인할 수 있었다.

본 논문에서는 성능 측면만을 고려한 고성능 저장 장치 SAN 최적화기법들을 구현하였으나, 향후에는 SAN 환경 구성에 필수적인 신뢰성과 운영 측면까지도 고려한 기술들을 추가적으로 연구할 계획이다. 또한 본 논문에서 제안한 SAN 최적화 기법들을 SAN뿐만 아니라 네트워크를 통한 이중화와 같은 분야에도 적용할 수 있을 지 연구할 계획이다. 장기적으로는 개발한 최적화기법들을 발전시켜 이후 등장할 차세대 저장장치 장비들에 대해서도 적용할 수 있도록 연구할 계획이다.

REFERENCES

[1] Ashish Palekar, Design and Implementation of ALINUX SCSI Target for Storage Area Networks, NETWORKS, In ALS' 01

[2] Bob Woodruff, Introduction to the InfiniBand Core Software, Linux Symposium, Vol. 2, pp. 271-282, July 2005.

[3] Burr, G. W., Overview of candidate device technologies for storage-class memory, IBM Journal of Research and Development, Vol. 52, Issue 4.5, pp. 449-464, July 2008.

[4] Caufield, A. M., Et Al., Moneta: A high-performance storage array architecture for next-generation, non-volatile memories, In MICRO'10 (2010), pp. 385 - 395.

[5] Charawi., S., Using a shared storage class memory device to improve the reliability of RAID arrays, In PDSW 2010, pp.1-5.

[6] Jose B. Cruz, JR, Leader-follower strategies for multilevel systems, IEEE Transactions on Automatic Control, Vol. 23, No. 2, pp. 244 - 255, Apr. 1978.

[7] DP Bovet, M Cesati, Understanding the Linux Kernel, 3rd Edition, O'REiLLY, 2005.

[8] Freitas, R. F., storage-class memory: The next storage system technology, IBM Journal of Research and Development, Vol. 52, Issue 4.5, pp.439-447, July 2008.

[9] Fusion-io, ioDrive, http://www.fusionio.com/products/iodrive/

[10] IBM, Introduction to storage Area Networks, http://www.redbooks.ibm.com/redbooks/pdfs/sg245470.pdf

[11] Jens Axboe, Flexible IO Tester, http://git.kernel.dk/?p=fio.git

[12] John Ousterhout, The Case for RAMClouds: Scalable High-Performance storage Entirely in DRAM, ACM SIGOPS Operating Systems Review, Vol. 43, No. 4, pp.92-105, January 2010.

[13] Mellanox, Building a Scalable Storage with InfiniBand, http://www.mellanox.com/relateddocs/whitepapers/WP_Scalable_Storage_InfiniBand_Final.pdf

[14]Mellanox, InfiniBand storage, http://www.mellanox.com/pdf/whitepapers/I-nfiniBand_storage_WP_050.pdf

[15]Mellanox, Introduction to InfiniBand, http://www.mellanox.com/pdf/whitep-apers/IB_Intro_WP_190.pdf

[16] Mike Ko, Technical Overview of iSCSI Extensions for RDMA (iSER) & Datamover Architecture for iSCSI (DA), RDMA Consortium, 2003.

[17] RAMCloud Project, https://ramcloud.stanford.edu/wiki/display/ramcloud/R-AMCloud

[18] Ru Fang, High Performance Database Logging using Storage Class Memory, In ICDE 2011, pp.1221-1231.

[19]SAP, SAP HANA, http://www.sap.com/solutions/technology/in-memory-computing-platform/index.epx

[20] SCST, Generic SCSI Target Subsystem for Linux, http://scst.sourceforge.net/

[21] TAEJININFOTECH, HHA 3804, http://www.taejin.co.kr

[22] Young Jin Yu, el., Exploiting Peak Device Throughput from Random Access Workload, In Hot storage'12, USENIX.

이 인 선(Lee, In Seon)

・1987년 2월 : 서울대학교 계산통계학과(이학사)
・1996년 8월 : 한국과학기술원 정보및통신공학과(공학석사)
・2003년 8월 : 서울대학교 대학원 전기.컴퓨터공학부(공학박사)
・1998년 3월 ~ 현재 : 신구대학교 IT미디어과 교수
・관심분야 : 분산시스템, 그리드시스템, 클라우드 컴퓨팅
・E-Mail : inseon@shingu.ac.kr