고성능 클라우드 스토리지 환경 구축을 위한 SAN 최적화

최재우*, 신동인**, 엄현상*, 염헌영*
*서울대학교 컴퓨터 공학과
**태진 인포텍
e-mail : jwchoi@dcslab.snu.ac.kr

SAN Optimization for Implementing High Performance Cloud Storage Environments

Jae Woo Choi*, Dong In Shin**, Hyeonsang Eom*, Heon Young Yeom*
*Dept. of Computer Science and Engineering, Seoul National University
**TAEJIN INFOTECH

요 약

오늘날 클라우드 서버 시스템 환경은 급속도로 증가하는 데이터를 효율적으로 처리 및 저장하기 위해 네트워크로 연결된 다수의 서버 머신들로 clustering 이나 분산 시스템 환경, 또는 SAN(Storage Area Network) 환경 등을 구성하여 보다 효율적인 서버시스템을 구현하고 있다. 이러한 서버시스템 환경에서의 병목현상은 주로 디스크기반의 스토리지에서 발생하며, 이를 극복하기 위해 고성능 스토리지에 대한 요구가 증가하고 있다. 그러나, 단순히 디바이스를 교체하는 것 만으로는 고성능 장비의 뛰어난 성능을 제대로 활용할 수 없으며 그에 맞는 최적화 작업이 요구된다.

본 논문에서는 기존의 SAN Solution 의 문제를 분석하고, 고성능 스토리지의 성능을 잘 활용할 수 있는 설정 및 최적화 방법을 제안한다.

1. 서론

CPU core 의 수가 점점 늘어나면서 computing capability 가 높아지고, 초고속 네트워크의 성능이 빠르게 발전하고 있는데 비해, 스토리지 성능의 발전속도는 그에 훨씬 미치지 못하고 있다. 그로 인해 전체 시스템 환경에서 스토리지 시스템의 항상 성능상의 병목이 되고 있으며, 이러한 문제를 해결하기 위해 고성능 디바이스에 대한 시장의 요구가 증가하고 있는 추세이다. HDD 기반의 스토리지는 기계적 움직임으로 동작하는 하드웨어의 속성상 성능상에 한계가 존재하며, 이러한 시장의 요구를 만족 시킬 수 없다. 그 대안으로 플래쉬 기반의 고성능 SSD 나, DRAM 기반의 SSD, 혹은 메인 메모리를 직접 스토리지로 활용하는 테크닉 등이 고려되고 있으며, PCM, FeRAM, STT RAM 등의 차세대 스토리지에 대한 연구도 활발히 진행되고 있다[1,2,9,12]. 하지만 단순히 기존의 스토리지 환경에 고성능 디바이스를 적용하는 것만으로는 디바이스의 뛰어난 성능을 활용할 수 없으며, 그에 맞는 최적화가 필요하다.

SAN(Storage Area Network)은 Host 컴퓨터와 스토리지 간의 data bus 를 high-speed network 로 대체한 시스템으로, 스토리지 디바이스는 네트워크에 directly attached 되며, standard network protocol 을 통해서 여러 Host 컴퓨터와 상호작용한다. 이러한 SAN 은 보통 enterprise 와 small 또는 medium sized business environments 에서 클라우드 스토리지 환경 등의 효율적인 서버시스템 구축을 위해 널리 사용되고 있으며, Gigabit Ethernet, Fibre Channel, Infiniband 등의 초고속 네트워크를 이용하여 구축할 수 있다[3,5,6].

본 연구에서는 먼저 SAN 환경에 단순히 고성능 storage 장비를 적용하여 기존 SAN solution 이 고성능 storage 장비의 뛰어난 성능을 제대로 활용할 수 없다는 사실을 확인하였다. Storage 장비는 PCI-E 인터페이스를 이용하는 DRAM 기반의 SSD 를 이용하였다. 이 장비는 latency 와 bandwidth 측면에서 뛰어난 성능을 자랑하며, sequential, random I/O 모두 uniform 한 성능을 보여 주고 있어, 본 연구의 목적에 아주 적합한 스토리지 장비라 할 수 있다[11]. 기존의 솔루션으로는 SAN 환경을 구축할 수 있도록 해주는 오픈 프로그램인 SCST 를 사용했고[10], 네트워크는 RDMA 를 이용하여 데이터 전송을 수행하는 초고속 네트워크 장비인 infiniband 를 사용하였다[7]. 다양한 I/O 타입에 대해 실험한 결과 small size random I/O 경우 read 와 write 모두 고성능 스토리지 장비에 대한 성능이 현저하게 떨이지는 것을 확인할 수 있었다.

이에 우리는 이러한 성능 저하의 원인을 분석하고 간단한 configuration 변경 및 코드 수정을 통해 이를

해결할 수 있는 최적화 방법을 제안한다.

2. Infiniband 를 지원하는 기존의 SAN 솔루션

현재 Linux 환경에서 Infiniband 의 RDMA 전송 기능을 활용해 간단히 SAN 을 구현할 수 있는 방법은 두 가지 이다. 첫 번째는 SCSI RDMA Protocol (SRP)를 이용하는 것이고[10], 다른 하나는 iSCSI extensions for RDMA (iSER) protocol 을 이용하는 것이다[8]. 이 둘은 모두 Infiniband 를 지원하고 SCSI layer 에서 RDMA 데이터 전송 방식을 이용하여 SAN 환경을 구축할 수 있다는 점에서는 동일하지만, 실제로 어떤 솔루션을 사용할 지 결정하기에 앞서 이 둘 간의 tradeoff 를 고려해야 할 것이다.

우선 SRP 를 통한 SAN 구현은 SCST 를 통해서 가능하며, iSER 의 경우 tgt projet 에 의해서 구현 가능하다. SRP 는 iSER 에 비해 SAN 환경을 구축하기 수월한 편이다. 또한 SCST 의 SRP Target 의 경우 kernel 영역에 구현되어 있어 user 영역에 구현되어 있는 iSER target 보다 높은 bandwidth 와 짧은 latency 를 보여준다. 하지만 iSER 은 iSCSI 인터페이스를 사용하기 때문에 target-discovery 를 수행하여 특정 Target 에 log in 한다거나 password 기반의 사용권한을 준다거나 하는 다양한 management 기능들을 이용할 수 있다[8,10]. 본 연구에서는 Performance 에 초점을 맞추고 있기 때문에 보다 뛰어난 성능을 보여주는 SRP 를 선택하였다.

3. 실험환경

우리는 두 개의 머신을 사용하였으며, 각각 Host 컴퓨터와 스토리지 서버를 구성해 실험을 진행하였다. 두 머신 모두 두 개의 Intel Xeon E5630 2.53GHz quad core CPUs (total 8 core)를 가지고 각각 8GB, 16GB size 의 main memory 를 장착하고 있으며, 모든 실험은 Linux 2.6.32 kernel 에서 수행하였다. 스토리지 서버에는 64GB(=8GB of DDR2*8) size 의 capacity 를 가지는 DRAM-Based SSD 가 존재하며, 이 장비를 본 연구의 고성능 storage system 으로 활용한다. SAN 환경 구축을 위한 초고속 네트워크 장비로는 Mellanox 사의 ConnectX-2 Infiniband 장비인 MHQH18B-XTR 을 사용했으며, 최대 40Gb/s 의 bandwidth 의 성능을 가진다.

우리는 먼저 고성능 디바이스에 대한 Local I/O 성능 테스트를 진행하였으며, 이 결과를 기준으로 기존의 SAN 과 본 논문에서 제안한 최적화를 적용한 SAN 과의 성능을 비교 분석한다.

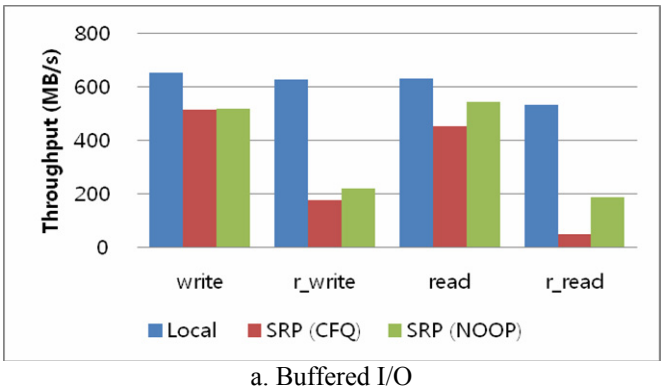
성능 실험을 위해서 FIO 와 같은 micro-benchmark tool 을 이용하였으며[4], 16 개의 multi-threaded I/O 하에서 Small Size 데이터(4KB)에 대해 다양한 I/O pattern 으로 실험을 진행 하였다. Large size I/O 의 경우에도 어느 정도 성능향상을 확인할 수 있었지만, 이미 괜찮은 성능을 보여주고 있었기 때문에 본 논문에서는 다루지 않았다.

4. I/O Scheduler Overhead 제거

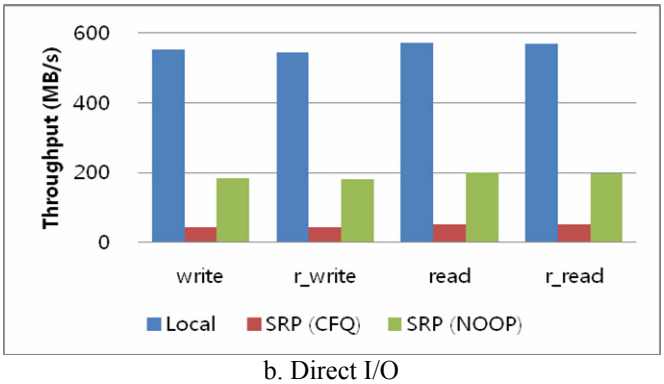
Linux I/O subsystem 의 I/O scheduler 는 기본적으로 Disk 를 가정한 scheduling policy 를 사용하거나, merge 를 위해 I/O 를 바로 처리하지 않고 delay 하는 등, 고성능 storage system 하에서 오버헤드로 작용할 수 있는 불필요하고 복잡한 작업들을 수행하고 있다. 예를 들어 linux 의 기본 scheduler policy 인 CFQ 는 가장 대표적인 Disk assumption 중 하나이다. 각 프로세스마다 작업큐를 가지고, time slice 를 할당 받으며, time slice 안에 작업을 모두 끝내더라도 바로 끝내지 않고 특정 시간동안 혹시나 있을 I/O 요청을 기다린 후 결국 없으면 다음 프로세스 큐로 이동하는 방식이다. 이는 고성능 스토리지에서는 불필요한 delay 를 발생시켜 치명적인 overhead 로 작용한다.

따라서 우리는 scheduler 의 정책을 NOOP 으로 변경함으로써 I/O scheduler 에 존재하는 overhead 를 감소시킬 수 있었다. 이는 간단한 configuration 변경만으로 가능하다.

그림 1 은 CFQ 정책과 NOOP 정책을 적용했을 때의 성능들을 Local I/O 의 성능과 함께 비교한 결과이다. 실험 결과에서 알 수 있듯이, 간단히 scheduler 정책을 변경하는 것 만으로도 모든 패턴에 대해서 큰 성능 향상을 확인할 수 있었다. Buffered I/O 에서 sequential-write 과 sequential-read 패턴의 경우, 각각 Merge 와 read-ahead 효과를 통해 SAN I/O path 의 bandwidth 를 잘 활용하면서 충분히 좋은 성능을 보여 주고 있다.



a. Buffered I/O



b. Direct I/O

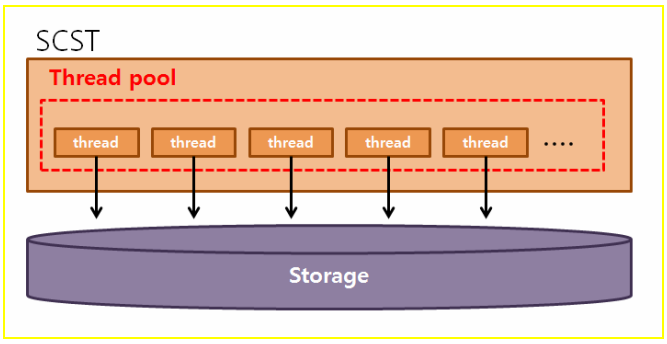
(그림 1) CFQ 정책과 NOOP 정책의 성능 비교

5. Parallelism 향상

고성능 storage 시스템을 위한 SAN solution 에 적용될 두 번째 optimization 기술은 Target 에서 I/O 요청을 처리하는 과정에 대해서 parallelism 을 증가 시키는 것이다. 일반적으로 고성능 storage 의 경우 디바이스 레벨에서 I/O 에 대한 parallelism 을 높이고 있다. 이는 multiple I/O 를 효율적으로 처리하여 뛰어난 bandwidth 성능을 구현할 수 있도록 한다. 하지만 기존의 SAN solution 들은 이러한 고성능 storage 의 특성을 잘 이용하지 못하고 있다.

실제로 SAN Storage 서버에서 수행되는 일련의 처리 과정들은 서로 거의 독립적이라고 볼 수 있으며 parallel 하게 처리될 수 있는 작업들이다. SCST 역시 이러한 처리과정을 thread-pool 의 multi thread 를 활용하여 어느 정도 parallel 하게 처리하고 있다. 하지만 실제 device 에 수행되는 I/O 의 경우 serial 하게 처리되는 경우가 많았으며, 이로 인해 고성능 storage 의 뛰어난 bandwidth 를 제대로 활용하지 못하여 심각한 성능 저하가 발생하고 있었다. 이는 small sized random I/O 의 경우 더욱 심각한 성능저하 현상을 초래하게 된다.

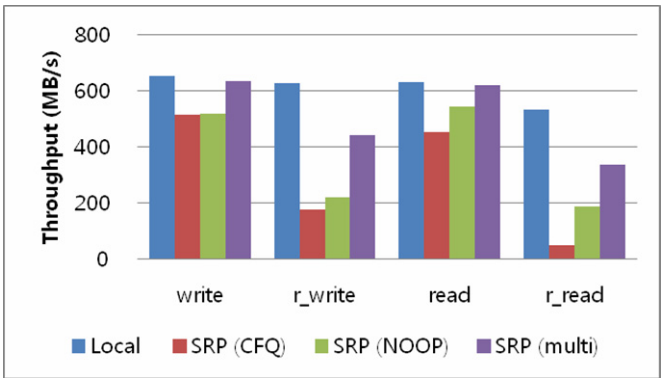
따라서 본 연구에서는 그림 2 와 같이 실제 device 에 대한 I/O 역시 multi thread 로 parallel 하게 처리할 수 있도록 기존의 코드를 약간 수정하였다. Thread-pool 에 존재하는 다수의 thread 들은 실제 device I/O 를 포함하여, I/O 를 처리하기 위해 필요한 모든 과정들을 parallel 하게 수행할 수 있다.



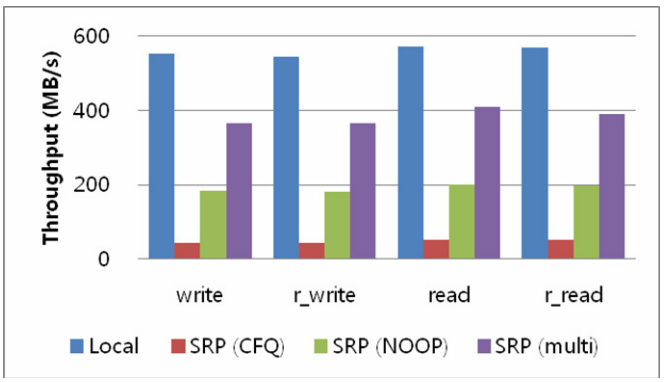
(그림 2) SAN Storage 서버에서 thread-pool 을 활용한 multiple device I/O

그림 3 은 parallel 한 device I/O 를 적용한 실험 결과이다. 실험환경은 앞의 경우와 동일하며, 마찬가지로 대부분의 I/O 패턴에 대해서 성능이 향상하는 것을 확인할 수 있었다. 기존에 충분히 좋은 성능을 보여주고 있었던 buffered I/O 의 sequential read/write 의 경우에도 parallelism 증가로 인해 어느 정도 성능상의 이득을 볼 수 있었으며, random I/O 패턴의 경우 상당한 성능 향상을 확인할 수 있었다.

이 결과는 앞으로 등장할 고성능 스토리지 환경에서는 디바이스의 특성을 잘 반영하는 I/O 처리 과정이 필수적으로 요구된다는 사실을 보여주고 있다.



a. Buffered I/O



b. Direct I/O

(그림 2) Multiple device I/O 최적화를 적용한 성능 비교 분석

6. 결론

우리는 고성능 스토리지 시스템의 뛰어난 성능을 최대한 잘 활용할 수 있는 클라우드 스토리지 환경을 구축하기 위해서 기존의 SAN 솔루션에 대한 최적화 연구를 진행하였다. 먼저 고성능 디바이스를 기존의 SAN solution 인 SCST 에 그대로 적용해 보았으며, 초고속 네트워크 장비인 Infiniband 를 사용했음에도 불구하고, 심각한 성능 저하 현상이 발생하는 것을 확인하였다. 이에 우리는 그에 대한 원인을 분석하고, 간단한 configuration 변경 및 코드 수정을 통해 문제를 해결할 수 있는 최적화 방안을 제시하였다.

첫 번째는 I/O scheduler 의 스케쥴링 정책을 CFQ 에서 NOOP 으로 바꾸는 것이다. CFQ 는 상대적으로 느린 장비인 HDD 를 가정하여 최적화된 정책으로, 고성능 디바이스 환경에서는 심각한 overhead 로 동작하게 되는 것을 확인 하였다. 이에 우리는 CFQ 를 NOOP 으로 변경함으로써 전체적인 성능향상을 확인할 수 있었다.

두번 째 최적화는 I/O 처리과정에서의 parallelism 을 높이는 것이다. 기존의 SAN 솔루션에서 디바이스 I/O 를 serial 하게 처리하는 것을 발견하였고, 이로 인한 성능 저하현상을 확인하였다. 이에 우리는 디바이스 I/O 역시 multi threads 를 활용하여 parallel 하게 처리할 수 있도록 코드를 수정하였으며, 이를 통해 상당한 성능 향상을 확인 하였다.

본 논문에서 제안하는 두 가지 방안은 고성능 디바

이스를 사용하는 클라우드 스토리지 환경에 꼭 필요한 최적화 과정이 될것이라 확신하며, 이는 앞으로 등장할 차세대 스토리지 환경에도 적용할 수 있을 것이라 믿는다.

7. 사사

본 연구는 방송통신위원회의 방송통신정책센터 운영 지원사업의 연구결과로 수행되었음 (KCA-2011-1194100004-110010100)

참고문헌

[1]CAULFIELD, A. M., ET AL. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In MICRO'10 (2010), pp. 385–395.

[2]Chaarawi., S. Using a shared storage class memory device to improve the reliability of RAID arrays, In PDSW 2010, p1-5

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

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

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

[6]Mellanox, InfiniBand Storage, http://www.mellanox.com/pdf/whitepapers/InfiniBand_Storage_WP_050.pdf

[7]Mellanox, Introduction to InfiniBand, http://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf

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

[9] Ru Fang, High Performance Database Logging using Storage Class Memory, In ICDE 2011, p1221-1231

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

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

[12]Young Jin Yu, Exploiting Peak Device Throughput from Random Access Workload, In Hotstorage'12