SAN기반 공유 저장 장치 시스템에서 고성능 소프트웨어 RAID를 위한 기법

김 경호 황 주영 안 철우 박규호
한국과학기술원 전자전산학과
(kyhkim, jyhwang, cwahn, kpark)@core.kaist.ac.kr

Efficient Techniques for Software RAID of SAN based on shared Storage systems

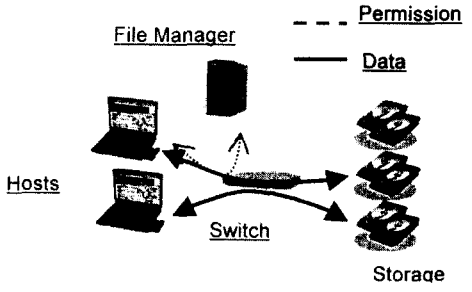
Kyung Ho Kim Joo Young Hwang chull Woo Ahn and Kyu Ho Park
Dept. of EECS, Korea Advanced Institute of Science and Technology

요 약

본 논문에서는 최근 이슈가 되고 있는 Fibre Channel과 같은 네트워크를 이용한 공유 디스크 시스템에서 다수의 호스트가 소프트웨어 RAID 4 혹은 5를 이용할 때, 디스크 공유에 의해 발생하는 패리티 블록 일관성 문제를 다룬다. 본 논문에서는 XDWRITE을 이용해 Software RAID에서의 일관성 문제 해결과 디스크 I/O수 감소에 대해 설명한다. 또한, 버퍼캐쉬에서 파일 블록 쓰기 발생시 원본의 복사본을 이용하여 패리티 계산을 위한 디스크 읽기의 수를 감소 시키는 방법에 대해 제안한다. 본 논문에서 제안한 방식은 기존의 방식인 보다 1.5배에서 2배 디스크 I/O의 수를 감소시키는 것을 알 수 있다.

1. 서 론

같은 전통적인 I/O 서브 시스템들은 다수 저장 장치를 사용하려 할 때, 그 다중의 저장 장치를 조절하기위한 단일의 중앙 집중화 된 구성요소를 사용한다. 이러한 단일 저장 제어기는 어플리케이션들로부터 읽기/쓰기 요청을 받고 그것들을 조절하여 단일의 저장 장치로 보일 수 있도록 한다. 따라서 이러한 저장장치의 주요 한계는 여러 호스트가 저장장치에 접근하려 할 때, 다수의 저장 장치를 관리하는 단일 집중화 된 서버의 저장 장치 제어기로 모든 데이터가 지나가야 하기 때문에, 부하 집중으로 인한 확장성이 떨어진다는 것이다. 최근에는 그림 1에서 보는 바와 같이, Fibre Channel과 같이 스위치로 연결된 스토리지 영역 네트워크, SAN의 도움으로 수천개의 저장 장치들이 호스트들에 의해 직접 연결되어 공유되고 있으며, 이것은 호스트들의 제어기의 공유 문제를 해결함으로 써, 잠재적으로 수천 개 이상의 디바이스 및 호스트가 직접 연결 가능하게 되어 뛰어난 확장성을 제시하고 있다[1][2][3]. 이러한 장점으로 이해, 최근에는 GFS(Global File System)[4][5] 나 SFS(Shared File System)[6]과 같이 공유 저장 장치 시스템 환경에서 발생할 수 있는 파일 일관성을 유지하는 파일 시스템들이 개발되어 사용되고 있다.



그러나, 그러한 공유 저장 장치 시스템에서는 분산 형태에 의한 중앙 조정 역할의 부재를 초래하며, 이것은 동시 저장 장치 접속 시 망가진 저장장치 복구를 위한 잉여 데이터의 갱신을 위해 두 호스트가 잉여 데이터를 읽고 쓰려 할 때 잉여 데이터의 일관성이 유지를 필요로 한다. 기존의 연구[7]에서는 같은 잉여 데이터 블록을 구성하는 저장 장치들의 블록들 단위, 즉 스트라입 단위로 락을 두어 관리하여, 다중 호스트의 접속 시 발생하는 잉여데이터의 일관성을 유지하고 있으며, 성능 향상을 위해 같은 스트라입에 대한 요청들을 구분하여 스트라입을 구성하는 저장 장치 개수의 반수 이상의 쓰기에 대해서 잉여 데이터를 저장장치에서 읽지 않고 다시 계산해서 쓰는 방식을 이용하여 저장 장치의 I/O를 줄이는 방식을 이용하고 있다. 본 연구는 호스트간 블록 쓰기 공유가 방지되어 있는 파일시스템 환경의 공유디스크 시스템에서 다중 호스트가 RAID 5의 디스크 쓰기를 할 때, 디스크 내부의 Lock을 이용한 방식을 이용하여 잉여데이터의 일관성을 유지하며 파일 시스템에서의 데이터 중복을 이용하여 성능을 높이는 방식을 제안한다.

2. 기존 연구

2.1 공유 디스크 상황에서 소프트웨어 RAID 4 혹은 5의 패리티 블록 일관성

RAID는 디스크의 병렬화를 통해 I/O 성능을 높이기 위한 것으로 패리티 데이터를 두어 고장 감내성을 높인 것이다. RAID는 하드웨어 및 소프트웨어 RAID로 나뉘며, 하드웨어 RAID는 특정한 하드웨어를 이용해 호스트의 OS 및 어플리케이션에서 디스크가 단일 이미지로 보이게 하는 것이며, 소프트웨어 RAID는 특별한 하드웨어 없이 OS내의 Device Driver와 같은 특정 계층에서 상위 어플리케이션에 단일 이미지를 보이기 위해 병렬 디스크를 제어하는 방식이다. RAID 4 혹은 5는 데이터들을 스트라이핑 방식으로 디스크에 저장을 하며 데이터 복구를 위해 하나의 디스크에 데이터들의 패리티를 저장하는 방식이다. RAID 5에서 단일 데이터 쓰기에 대해서는 패리티 갱신을 위해 2reads-2writes의 작업이 필요하다[8]. 이는 4번의 디스크 I/O를 불러 일으킨다. 스트라입 크기가 8인 그림 2에서 A, B, C, D, E, F, G의 블록과 P의 패리티블록을 형성하고 있을 때 호스트가 A'를 갱신하려면 디스크에서 A와 P를 읽어서 A'와 갱신된 P'를 디스크에 써야한다. 만약 이런 상황이 공유 디스크 시스템에 연결된 호스트 1과 호스트 2가 각각 A'와 A"를 차례로 갱신하려 한다면, 최종 패리티, P"=A⊕A"⊕P가 되어야 한다. 그러나, 만약 호스트2에서 패리티를 읽어가는 시간이 호스트 1이 업데이트 한 P'를 읽어 가기 전이었다면, P"=A'⊕A"⊕P가 되어 패리티의 일관성이 유지되지 못한다.

2.2 기존 연구

이와 같은 문제를 해결하기 위한 기존의 연구[6]에서는 하나의 서버가 각 스트라입 단위로 락을 두어 관리를 하며, 성능 향상을 위해 스트라

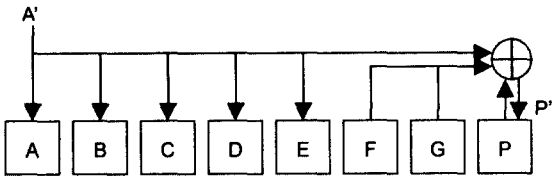


그림 2 RAID 5

입 크기의 반 이상의 데이터 쓰기를 LargeWRite이라는 정의 하고 이러한 경우에는 디스크에서 패리티를 읽지 않고 새로 계산된 패리티를 쓰는 방식으로 디스크 I/O를 줄이고 있다. 예를 들면 스트라입 크기가 8의 그림 3에서 그 스트라입의 락을 잡은 호스트가 A, B, C, D, E의 블록을 쓰려 할 때, 10reads-10writes의 20번의 디스크 I/O의 수를 2read-6write인 8로 줄인 것을 볼 수 있다.

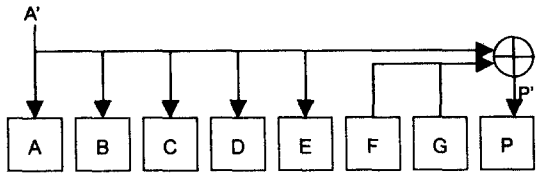


그림 3 기존 연구 : 성능 향상을 위해 LargeWrite 기법

3. 실용적 디스크 모델 기반 새로운 락 기법 및 버퍼 캐쉬를 이용한 성능 향상 기법

3.1 XDWRITE을 이용한 락 기법

디스크 I/O에서 대부분의 요청들은 읽기/쓰기로 구성된다. 그러나, 이는 최근의 실제 디스크 모델의 섬세함이 떨어지는 것으로 일로서, 최근의 디스크는 많은 종류의 명령을 정의 및 사용하고 있다. 특히, 하드웨어 RAID 가 보편화 되면서, 하드웨어 RAID에 최적화된 명령들이 표준화 되어 사용되고 있으며, XDWRITE[9]이라는 명령도 그 중의 하나로서, SCSI의 표준으로 정의되어 있다. 이것은 호스트에서 A블록을 이용해 P가 쓰여져 있는 디스크에 XDWRITE 요청을 하게 되면 디스크는 호스트에서 A블록을 가져가서 디스크 내부에서 새로운 데이터 P'=P⊕A를 생성하게 된다. 본 논문에서는 XDWRITE 요청을 이용하여 공유디스크 시스템에서 패리티 갱신을 위한 락으로 사용되는 방식을 제안한다. 만약 스트라입에 존재하는 블록 A'를 A의 디스크 블록에 쓸 때, 호스트에서 A를 읽고 A'와의 패리티를 계산한 결과를 XDWRITE을 이용해 스트라입의 패리티 디스크에 XDWRITE을 하는 것이다. 예를 들면, 그림 3에서 블록 A'의 쓰기에 의한 패리티는 A⊕A'의 XDWRITE 에 의해 이루어 진다. 실제로 XDWRITE에 의한 패리티 갱신은 디스크 내부에서 P를 읽어내는 과정이 포함이 되지만, 헤드의 찾기가 감소하게 된다. 본 논문에서는 간단한 성능 비교를 위해 XDWRITE에 의한 I/O 시간의 감소를 절반의 I/O라고 표현을 하며, 그 I/O의 수를 0.5로 대표 한다. 그러나, 최근의 SCSI 디스크에서 성능 병목현상은 헤드의 찾기에서 발생한다. 그렇기 때문에, I/O수를 0.5로 나타내는 것이 설득력을 가진다. 예를 들어, 최근의 디스크인 ST336737FC에서 10Kbyte 블록을 읽기 위한 I/O 시간은 평균적으로 보면, 찾기와 회전, 그리고 플래터에서 데이터를 읽는 시간은 각각 8.7ms, 4.1m 그리고 250us이다.

이 방식을 이용하게 되면, I/O 수가 줄어들 수 있다는 장점 뿐만 아니라 디스크 I/O버스에서의 데이터 전송 수가 줄어들며, 락을 관리하는 서버가 필요 없어지게 된다. 또한 스트라입 전체에 대한 락이 아니므로 다른 디스크의 접근이 허용 된다는 장점이 있다.

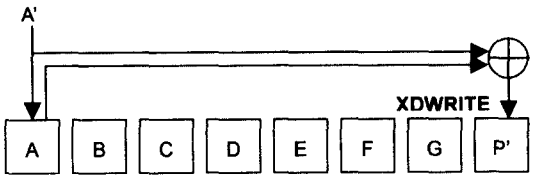


그림 4 XDWRITE을 이용한 락 방법

3.2 버퍼캐쉬를 이용한 성능 향상

본 절에서는 제안한 XDWRITE을 이용한 락 기법을 기반으로 하는 시스템에서 LargeWrite에 대한 성능 향상 기법에 대해 제안한다. 본 논문에서는 LargeWrite인 경우의 디스크 I/O 성능 향상을 위해 호스트의 버퍼 캐쉬를 이용하는 방법에 대해서 논한다. 버퍼 캐쉬란 호스트 내에서 관리되는 파일의 블록 캐쉬이다. 파일 시스템이 관리하는 파일 블록은 호스트 내에서는 버퍼 캐쉬에서 관리되며, 새로운 블록을 요할 경우, 수정이 일어난 dirty 블록중에서 LRU방식에 의해 디스크에 쓰여 지게 된다. 본 논문에서 제안하는 바는 버퍼캐쉬에서 사용되는 블록의 내용에 수정이

가해지면, 그때 그 블록에 대한 원본 복사본을 버퍼캐쉬 내에 가지고 있도록 하여, 실제 LRU에 의해 디스크로 쓰기가 이루어 질 때, 디스크에서 원본을 읽지 않고 패리티 계산 내용을 XDWRITE를 이용하여 패리티를 갱신 하는 방식이다. 예를 들면, 그림 5에서 블록 A, B, C, D, E를 쓸 때, 복사본에 의해 5번의 read를 줄일 수 있게 된다. 기존의 방식과 비교하면, 기존의 방식인 2reads-6writes에서 0.5read-6writes만이 이루어지는 것을 알 수가 있다.

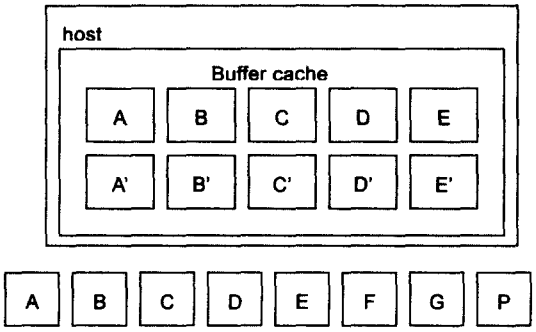
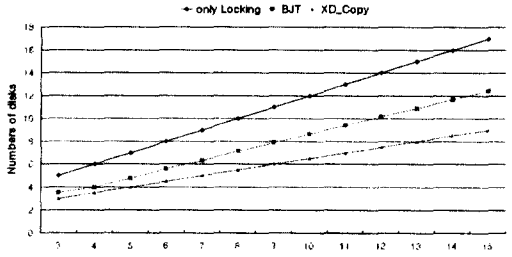


그림 5 버퍼캐쉬를 이용한 성능 향상 기법

4. 성능 비교

그림 6은 성능 비교를 나타낸 것이다. 그림에서 Only-locking 기법은 스트라입 락만을 사용한 것이고, 디스크 요청들의 분류 방식인 BST를 이용한 2.2절에서 설명한 기존의 방식으로 LargeWrite시 쓰기 블록이 개수가 스트라입의 크기의 반 이상인 경우 패리티를 읽지 않고 다시 계산하는 방식이다. 그리고 XD는 본 논문에서 제안한 방식을 나타낸다. 그림에서 X축은 디스크의 수, 즉 스트라입 크기를 나타내며, 예를 들어 그 숫자가 8이면 7개의 데이터 디스크와 1개의 패리티 디스크를 사용한 것이다. 그리고 비교가 되고 있는 값은 가능한 쓰기 요청을 동일 분포로 가정한 평균 디스크 I/O를 계산한 것이다. 결과로 볼 때, 본 논문에서 제안한 방식이 Only-locking 기법보다 2배, BST방식보다는 1.5배 정도 I/O의 개수가 줄어드는 것을 알 수 있다.



5. 결론 및 추후 과제

본 논문에서는 공유 저장장치 시스템에서 디스크들이 스트라입을 이루는 소프트 웨어 RAID4 혹은 5에서 다수의 호스트가 동시에 같은 스트라입의 블록을 쓰려 할 때, 패리티 일관성에 관한 문제를 다루고 있다. 본 논문에서는 실용적 디스크의 모델을 기반으로 새로운 락 기법과 버퍼캐쉬의 메모리를 이용한 디스크 I/O 수를 줄이는 방식을 제안한다. 락 기법은 디스크의 XDWRITE을 이용하는 방식으로 서버의 락 기능을 디스크로 옮김으로 써 I/O수를 줄일 수 있으며, 버퍼 캐쉬 내에서 블록 수정이 있을 때, 블록의 원본 복사본을 이용 LargeWrite의 경우에 디스크의 읽기 I/O수를 줄이는 방식이다. 성능에서는 다양한 스트라입 크기에 따른 평균 I/O수를 계산한 결과 기존의 방식인 록 기능만 있는 경우보다 2배, BST보다 1.5배 I/O수가 줄어드는 것을 알 수 있었다.

버퍼캐쉬에서 쓰기 블록의 원본 복사본을 유지하는 것은 호스트 내에서의 메모리의 낭비로서 성능 향상과 상관관계에 있다. 그러나, 기존의 방식 또한 패리티의 계산이 호스트에서 이루어 지는 관계로 호스트 내에서의 메모리 이중성이 불필요 해진다. 메모리 낭비에 대한 문제는 버퍼캐쉬 내에서 수정 블록의 life time과 많은 상관 관계를 갖고 있다. 따라서 메모리 낭비에 의한 성능 감소와 제안된 방식에 의한 성능 향상에 대해 실제 상황에서 기존의 방식과 비교해 보는 것을 추후과제로 남긴다.

참고문헌

[1] E. K. Lee, " Highly-Available, Scalable Network Storage" , in Compcon' 95, pp.397-402, March 1995
[2] J. R. Heath and P. J. Yokutis, " High-Speed Storage Area Networks using a Fibre Channel Arbitrated Loop Interconnect" IEEE Networks, vol. 14, pp.51-56, March-April 2000.
[3] D. A. Menasce, O. I. Pentakalos, and Y. Yesha, " An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices" , in the Proceedings of the ACM SIGMETRICS conference on measuring and modeling of computer systems, pp 180-189, May 1996.
[4] K. W. Preslan, S. R. Soltis C. J. Sabol, and M. T. O' Keefe, " Device Locks" Mutual Eclusion for Storage Area Networks" in the Proceedings of 16th IEEE symposium on Mass storage Systems, pp. 262-274, March 1999.
[5] K. W. Preslan, A. P. Barry, and J. E. Brassow, " A 64-bit, shared disk file system for Linux" , Mass Storage Systems, 1999. 16th IEEE Symposium on ,1999 pages 22-41
[6] L. M. Lee, K. K. H, S. S. Lim and K. H. Park, " Shared-disk Distributed File system for based on SAN" , Proceedings of the 26th KISS Fall Conference, Vol. 2, pp.786-788, 1999.
[7] K. Amiri, G. A. Gibson and R. Golding, " Highly concurrent Shared Storage" , Proceedings of the International Conference on Distributed Computing Systems, April 2000.
[8] RAB, The RAID Book. The RAID Advisory Board, 1993.
[9]http://www.seagate.com/support/disc/manuals/scsi/75789521a.pdf