제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

스트라이핑 시스템에서 쓰기 연산에 의한 연쇄적 재구성 방법

박유현°, 김창수, 김영호, 강동재, 김학영
한국전자통신연구원 컴퓨터·소프트웨어연구소*

The cascading reorganization method by write operation in striping system

BAK,Yuhyeon°* KIM,Changsoo* KIM, Youngho* KANG, Dongjae* KIM, Hagyoung*
ETRI Computer & Software Laboratory

요약

데이터의 병행 읽기 성능을 높이기 위해서 데이터를 스트라이핑 방법으로 저장하는 시스템은 흔하게 볼 수 있다. 하지만, 시스템을 운영하다 저장장치를 확장해야 할 경우가 종종 발생하게 되는데, 이때 기존의 데이터에 대해서 재구성을 수행한 후에만 시스템을 정상적으로 사용할 수 있다. 하지만 데이터의 양이 급속하게 증가하고 있는 상황에서 재구성 연산을 수행할 때 그 오버헤드로 인하여 서비스를 중단해야 하는 상황이 발생한다.

이 논문에서는 스트라이핑으로 데이터를 저장하는 시스템에서 저장장치가 확장되었을 때, 모든 데이터를 대상으로 재구성을 수행하는 것이 아니라, 갱신 연산이 요청된 블록에 대해서만 재구성을 수행하는 연쇄적 재구성 기법을 제안한다. 사용자로부터 갱신이 요청된 블록이나 새로 저장될 블록은 추가된 디스크를 포함한 모든 디스크 수에 의해서 저장될 위치가 결정되고, 만일 이 위치가 새로운 디스크에 존재하면 연산이 종료된다. 하지만, 결정된 위치가 기존의 디스크라면 이미 이 위치에 존재하는 블록 또한 이동하게 된다. 이러한 현상으로 제안하는 방법은 연쇄적 재구성 방법이라 한다. 연쇄적 재구성 방법은 재구성 오버헤드를 분산시키는 효과를 가지면서 서비스 중단 시간을 줄일 수 있다. 하지만, 추가되는 디스크의 수가 기존 디스크의 수보다 적을 경우에는 사용자의 응답대기 시간이 길어지는 단점을 가진다.

1. 서론

RAID[1]는 시스템의 입출력 성능 개선 및 신뢰성을 위해 제안된 개념으로, 응용 특성에 따라 여러 단계를 제공한다. 그 중에서 0, 4, 5 단계는 디스크 배열을 구성하는 장치들에 데이터를 분산하여 저장하는 방법이다. 이러한 단계들은 기본적으로 스트라이핑을 하는데, 스트라이핑을 하게 되면 디스크에 대한 I/O를 동시에 여러 디스크에 분산시켜 성능을 높일 수 있다.

RAID 0, 4, 5와 같이 스트라이핑을 하는 시스템에서 데이터를 디스크에 저장할 때는 디스크 수로 모듈러 연산을 수행하여 저장할 위치 디스크를 결정하는 것이 보편적인데, 사용하고 있는 도중에 새로운 디스크를 추가하고자 할 때는 디스크의 수가 변경되기 때문에, 저장되어 있는 데이터를 전체 디스크를 고려하여 균등하게 분배하는 재배치를 수행하여야 한다. 대부분의 경우 재배치를 할 경우에는 데이터를 백업하고 데이터를 새로 추가된 디스크를 고려하여 다시 분배하기 때문에 이러한 연산을 수행하고 있는 동안에는 다른 디스크 입출력 서비스를 제공하지 않는 경우가 많다.

따라서 이 논문에서는 재구성을 쓰기 요청이 들어온 블록에 대해서만 수행하는 방법을 제안한다. 쓰기 연산이 들어오면 그 블록에 대해서 추가된 디스크의 수를 포함한 전체 디스크 수에 의해서 저장될 위치가 결정된다. 만일 이 위치가 새로운 디스크에 존재한다면 연쇄 재구성 연산은 종료된다. 하지만, 새로 결정된 위치가 기존의 디스크라면 이미 그 위치에 저장되어 있는 블록 또한 재구성을 수행해야 한다. 이러한 반복적 과정으로 쓰기 요청된 블록에 대한 연쇄적 재구성이 수행된다.

2. 관련연구

2.1 재구성 방법

재구성 방법은 스트라이핑 방법으로 저장되어 있다는 것을 가정한다. 왜냐하면 다른 RAID 레벨로 저장되어 있는 것은 스트라이핑 방법을 그대로 또는 조금만 수정하면 되기 때문이다. 기본적인 재구성 방법은 재구성 속도가 다소 느리고, 재구성 후의 I/O 성능을 최대로 하는 것을 특징으로 한다. 이 방법은 어떤 블록을 어느 위치로 옮길 것인지를 결정해서 재구성 테이블을 작성한다. 재구성 테이블 작성이 완료되면 다음은 재구성 테이블을 보고 블록을 옮기게 된다. 이 방법을 개선하여 적당한 수준의 스트라이핑 성능을 유지하도록 이동 데이터를 전체 데이터로 하지 않는 방법 등의 연구가 진행되고 있다.

2.2 재구성을 지연시키는 방법

2.2.1 SZIT(Striping Zone Information Table) 기반의 매핑 방법[5]

SZIT를 이용한 매핑 방법은 디스크가 추가되는 시점 이후로는 추가된 디스크들에만 스트라이핑으로 쓰기 연산을 수행하여 기존 디스크들의 데이터 보유량까지 진행한다. 모든 디스크의 데이터 보유량이 동일해 지면 그 이후로는 모든 디스크를 대상으로 스트라이핑을 수행한다. SZIT의 구성은 다음과 같다.

[표 1] SZIT의 구성

zone	스트라이핑 지역 번호
tot_disk	전체 디스크 수
par_disk	참여 디스크 수
s_phy	첫 물리 블록 번호
e_pyh	끝 물리 블록 번호
s_log	첫 논리 블록 번호
e_log	끝 논리 블록 번호

초기에 3개의 디스크로 운영되는 시스템에서 논리블록 11이 써진 후에 새로운 디스크 2개가 추가된 경우의 SZIT와 이때 디스크에 저장되는 데이터의 모습은 다음과 같다.

[표 2] 저장매체 추가 이후의 SZIT

zone	tot_disk	par_disk	s_phy	e_phy	s_log	e_log
0	3	3	0	3	0	11
1	5	2	0	3	12	19
2	5	5	4	14	20	74

2.2.2 테이블 기반의 매핑 방법[3]

이러한 디스크 수의 변화에 유연하게 적응하기 위해 [표 3]과 같이 논리주소와 물리주소를 테이블의 형태로 저장하여 매핑 할 수 있다.
하지만 이 방법은 논리블록의 수만큼의 테이블을 유지해야 하기 때문에 많은 저장공간이 필요하다. 세 개의 디스크가 각각 40GB이고, 블록 크기가 1K일 경우, 이러한 테이블을 유지하기 위해서는 (40GB/1K) × 한 튜플에 저장되는 데이터 크기 = 40MB × 한 튜플에 저장되는 데이터 크기가 된다.

[표 3] SZIT를 이용한 매핑 계산식

targetDevice = ((logBlkNo) % parDevice) + (totalDevice-parDevice) targetPhyBlk = (logBlkNo-logStartNo) / parDevice + phyStartNo	
targetDevice	논리블록 n을 저장하는 디스크
targetPhyBlk	논리블록 n에 매핑 되는 디스크 내의 물리 블록 번호
logBlkNo	논리블록 n의 주소
phyStartNo	logBlkNo의 물리블록 번호
parDevice	참여하고 있는 디스크의 수
totalDevice	전체 디스크 수

[표 4] 매핑 테이블의 구조

논리주소	디스크 번호	물리주소
0K	1	0K
1K	2	0K
2K	3	0K
3K	1	1K
4K	2	1K
5K	3	1K
:	:	:

3.쓰기 연산에 의한 연쇄적 재구성 방법

이 논문에서 제안하는 쓰기 연산에 의한 연쇄적 재구성 방법은 기존의 재구성 방법과 같이 재구성을 수행하고 있는 중에 일반 연산을 금지시키지 않고 계속해서 서비스를 제공할 수 있다. 대신 사용자의 요청 연산 중에서 데이터를 디스크에 저장하게 하는 연산(쓰기, 갱신 등)에 대해서 요청된 블록만을 대상으로 재구성을 수행하는 방법이다.

제안하는 방법은 요청된 블록으로 인해 재구성이

연쇄적으로 발생한다. 즉, 사용자로부터 요청된 디스크 저장 블록에 대해서 저장될 위치를 결정한다. 저장될 디스크는 블록번호를 전체 디스크 수로 모듈러 연산을 수행하여 결정하고, 결정된 디스크 내의 물리주소는 블록번호를 디스크 수로 나누면 알 수 있다. 만일 이렇게 결정된 위치가 새로운 디스크에 존재하면 사용자로부터 시작된 연쇄 재구성이 끝나게 된다. 하지만, 결정된 위치가 기존의 디스크라면 이미 그 위치에 저장되어 있는 블록에 대해서도 같은 과정을 수행해야 한다.

쓰기 연산에 의한 연쇄적 재구성을 수행하고 있는 시스템에서 특정 논리블록에 대한 디스크 접근 요청(읽기, 쓰기)이 들어오면 이 논리블록의 위치를 기존의 디스크 수를 적용해서 결정해야 할지, 디스크가 추가된 후의 디스크 수를 적용해서 결정해야 할지 알 수가 없다. 또한 시스템을 다시 시작한 경우에 재구성을 수행한 블록과 그렇지 않은 블록을 계속해서 유지해야 한다.

이러한 이유로 쓰기 연산에 의한 연쇄적 재구성을 수행하려면 다음과 같은 재구성 테이블이 필요하다.

[표 5] 초기 재구성 테이블

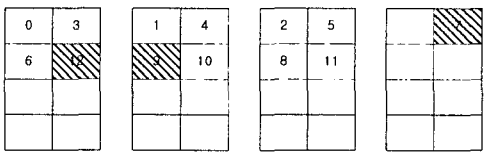
0	1	2	3	4	5	6	7	8	9	10	11	12	13
F	F	F	F	F	F	F	F	F	F	F	F	F	F

첫 행의 숫자들은 논리블록 번호를 나타낸다. 여기에서는 편의상 13번 블록까지만 표현하였다. 또한 아래쪽의 'F'로 표현된 부분은 물리주소가 저장되며, 물리주소는 (디스크 번호, 물리블록 번호)의 쌍으로 표현된다. 'F'로 되어 있는 블록은 현재 재구성이 수행되지 않은 블록을 나타낸다.

쓰기 연산에 의한 연쇄적 재구성 방법의 수행과정을 살펴보면 다음과 같다.

디스크가 추가되기 전에 논리블록에서 물리블록으로의 매핑은 수식을 사용한다. 저장된 블록의 디스크 번호는 (블록번호 % 디스크 번호)로, 저장될 블록의 물리 블록 번호는 (블록번호 / 디스크 번호)로 결정된다. 디스크가 추가되면 [표 ??]가 생성되고 모든 디스크 I/O 연산은 이를 참고하게 된다. 즉, 읽기 연산이 요청된 논리블록에 대해서는 재구성 테이블의 물리주소 필드의 값을 읽어서 그 값이 'F'(재구성되지 않았음)이면 추가되기 전의 디스크 수를 이용하여 물리주소를 결정하고, (디스크 번호, 물리블록 번호)의 값이 있으면 이를 참조하여 디스크에 접근한다.

사용자로부터 요청된 디스크 I/O가 쓰기 연산일 때는 먼저 해당 논리블록에 대한 물리주소를 재구성 테이블을 참조하여 얻는다. 물리주소가 초기값(F)이 아니라면 해당 위치에 데이터를 갱신한다. 만일, 물리주소가 초기값이라면 추가된 이후에 갱신이 한번도 일어나지 않은 것을 나타내므로 연쇄 재구성을 수행한다. 즉, [그림 1]과 같이 3개로 구성된 스트라이핑 시스템에서 논리블록 11까지 데이터를 쓴 후에 새로운 디스크 3이 추가되면 그 시점에서 [표 5]가 생성된다. 논리블록 12에 대해 쓰기 연산이 요청되면, 먼저 재구성 테이블에서 논리블록 12번의 물리주소를 읽는다. 한번도 쓰기 연산이 들어오지 않았기 때문에 초기값으로 되어 있다. 따라서 물리주소는 전체 디스크 수인 4를 기준으로 결정한다. 논리블록 12의 물리주소는(0, 3)이다. 디스크 번호 0은 논리블록(12)을 디스크 수(4)로 모듈러 연산을 하였고, 물리블록 3은 논리블록을 디스크 수로 나누기 연산을 하여 구한 것이다. 그런데, 논리블록 12번의 저장될 위치는 새로 추가된 3번 디스크가 아니라 0번 디스크이다. 그 위치에는 이미 논리블록 9의 정보가 저장되어 있다. 따라서 논리블록 9에 대해서도 재구성을 수행하여야 한다. 이렇게 반복적으로 수행하면 논리블록 7이 새로운 디스크에 결정되는 시점까지 따라가게 된다. 이러한 모든 과정이 끝난 후의 재구성 테이블은 [표 6]과 같다.



[그림 1] 논리블록 12에 의한 연쇄 재구성 과정

[표 6] 논리블록 12에 의한 연쇄 재구성 이후의 재구성 테이블

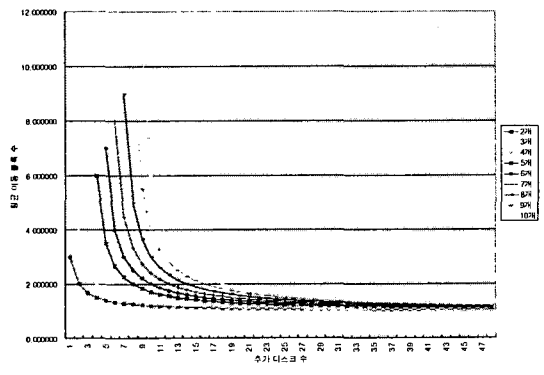
0	1	2	3	4	5	6	7	8	9	10	11	12	13
F	F	F	F	F	F	F	(3,1)	F	(1,2)	F	F	(0,3)	F

4. 연쇄적 재구성 방법의 성능평가

[그림 2]는 스트라이핑 시스템에서 연쇄적 재구성을 수행할 때의 평균 이동 블록 수를 나타낸다. 즉, 사용자로부터 쓰기 요청된 하나의 블록으로 인해 연쇄적으로 이동하는 블록들의 평균이다. 초기 디스크의 수는 2개부터 10개까지로 하였고, 각각의 경우 디스크를 1개 추가하는 상황에서부터 47개 추가하는 경우까지를 표현하였다.

그래프를 살펴보면 전반적으로 디스크를 많이 추가하면 추가할수록 연쇄 재구성 대상이 되는 블록의 수는 적어진다. 이것은 디스크가 많이 추가되면 처음 쓰기 연산이 일어난 후에 새로 추가된 디스크로 저장 위치가 결정될 확률이 높기 때문이다. 또한 디스크가 1개 추가된 경우에는 쓰기 블록 하나에 대해 대략적으로 디스크 수만큼의 이동이 발생한다. 응용에 따라 다르겠지만 적어도 현재 서비스를 제공하고 있는 디스크의 수만큼을 추가하는 경우에 쓰기 연산에 의한 연쇄적 재구성 방법이 효용성을 가질 수 있다. 운용중인 디스크의 수 이상을 추가한 경우에는 정상 시스템의 디스크 I/O 수에 비해 2배 이하의 성능을 보일 수 있기 때문이다.

만일 연쇄 재구성 중에 시스템이 다시 시작하는 경우, 모든 재구성 정보가 사라지게 될 수도 있다. 이를 방지하기 위하여 재구성 테이블도 디스크에 저장될 필요가 있는데, 이러한 경우에는 하나의 블록을 옮기는데, 추가 2번의 디스크 I/O가 발생하기 때문에 성능은 더욱 떨어질 수 있다.



[그림 2] 추가 디스크 수의 변화에 따른 평균 데이터 이동 수

하지만, [그림 2]는 첫 번째 블록부터 차례대로 쓰기 연산이 들어온 경우를 가정하였기 때문에 이미 재구성되어 있는 블록에 대한 중복 수도 포함되어 있다. 또한 일부 데이터에 집중되는 서비스 패턴을 볼 때, 제안하는 방법은 실효성을 가질 수 있다.

5. 결론

이 논문에서는 운영 중인 스트라이핑 시스템에 디스크가 추가되었을 때, 모든 데이터를 대상으로 재구성을 수행하는 것이 아니라, 디스크 추가 시점 이후에 요청되는 쓰기 연산에 의해 관련된 블록들만을 재구성하는 방법을 제안하였다. 이 방법은 쓰기 연산이 요청된 블록을 전체 디스크 수를 고려하여 위치를 결정하고, 만일 결정된 위치가 새로 추가된 디스크라면 재구성을 종료하고 새로운 사용자 요청을 기다린다. 하지만, 결정된 위치가 기존의 디스크 상에 존재한다면, 이미 그 위치에 저장된 데이터를 먼저 이동시켜야 한다. 이러한 과정을 반복적으로 수행하여 요청된 블록으로 인한 모든 블록들이 연쇄적으로 재구성이 된다.

만일 디스크에 새로 쓰기 연산이 자주 들어오지 않는 시스템에서는 이러한 방법을 사용하는 것이 유리하다. 또한 실험적으로 살펴보았을 때, 현재 운용되고 있는 디스크의 수와 같은 수 이상의 디스크를 추가하는 경우에는 참을만한 사용자 응답시간을 보인다. 하지만, 이 방법은 재구성 테이블이란 메타데이터가 필요하며 재구성 중간에 시스템이 재시작 하더라도 데이터의 일관성을 유지하기 위해서는 이를 디스크에 저장해야 하며, 이로 인한 성능이 떨어질 수 있다. 따라서 성능 개선을 위한 추가의 연구가 필요하다.

참고문헌

[1] D.A. Patterson, J.L.Hennesy, R.H.Katz, "A case for redundant arrays of inexpensive disk(RAID)", International Conference of Management of Data(SIGMOD), pp.109-116, 1988.

[2] Steven R. Soltis, Thomas M. Ruwart, Matthew T. O'Keefe, "The Global File System", Conference on Mass Storage Systems and Technologies, Sept 17-19, 1996.

[3] Edward K. Lee, Chandramohan A. Thekkath, "Petal:Distributed Virtual Disks", In Proceedings of International Conference on ASPLOS, 1996.

[4] "Features of veritas volume manager and veritas file system", http://www.veritas.com

[5] "스트라이핑 시스템에서 동적 디스크 수를 지원하는 계산에 의한 매핑 방법" 한국정보처리학회 춘계 학술 발표 논문집, 2002. 4.