RAID를 위한 패리티 디클러스터링 방식의 새로운 스페어링 기법

김경훈, 윤희용, 추현승
성균관대학교 정보통신 공학부
imfor, youn, choo@ece.skku.ac.kr

A New Distributed Sparing Approach with Parity Declustering for RAID

Kyung-Hoon Kim, Hee Yong Youn, Hyunseung Choo
School of Information and Communications
SungKyunKwan University

요 약

RAID 저장장치는 병렬성과 고 가용성을 추구하는 대용량 저장장치를 구축하는 방법이다. 특히 패리티 디클러스터링을 사용한 RAID 저장장치는 고장이 발생하였을 경우 재구축 작업중에도 사용자의 요청을 처리함에 있어서 성능 저하가 적고 높은 fault tolerance 와 고 가용성을 지원한다. 본 논문에서는 기존의 패리티 디클러스터링 기법과 분산 스페어링 기법을 적용한 RAID 구조에서 올 수 있는 단점을 보완할 수 있는 새로운 분산 스페어 기법을 제안한다. 본 논문에서 제안된 기법은 시뮬레이션을 통해 그 유효성이 입증되었고, 전반적으로 기존의 방식보다 1%~11%정도 성능이 더 좋음을 알 수 있다.

1. 서론

최근 들어 대용량의 디스크 저장 장치와 대용량 데이터의 압축기술 등의 컴퓨터 기술과 기가비트의 전송률을 갖는 초고속 통신망 등의 발전으로 사용자에게 대용량의 음성 자료, 동화상 자료 등의 멀티미디어 데이터 서비스가 가능하게 되었다. 이러한 대용량 데이터를 서비스하기 위해서는 데이터의 안정된 운용과 고속 서비스를 위한 고성능의 보조기억장치를 필요로 한다. 그러나, 대표적인 보조 기억 장치인 디스크는 기계적인 요인으로 인하여 성능향상에 많은 애로점이 있는 실정이다. 이를 해결하기 위해서 여러 개의 디스크를 배열로 구성하고, 에러복구 코드를 추가 시키는 디스크 시스템 구조인 RAID(Redundant Arrays of Independent / Inexpensive Disks)가 출현하게 되었다[1-5].

RAID 는 에러복구 코드와 데이터의 분산방법에 따라 여러 레벨로 구분할 수 있다. RAID 의 단계는 0, 1, 2, 3, 4, 5, 6, 7, 10, 53, 0+1 로 구분되어 있으며, 이들은 성능, 신뢰도 그리고 구축비용 면에서 차이를 인다. 현재 많이 사용하고 있는 RAID 는 0, 1, 2, 3, 4, 5, 7, 10 등 이다.

패리티 디클러스터링(Parity Declustering) 기법[1]은 RAID 구조에서 에러 발생시에 발생되는 성능 저하를 줄이고 가용성을 향상시키기 위하여 제안된 방법이다. 이 방법을 이용하면 디스크 배열에서 한 디스크에 고장이 발생하여 그 디스크의 데이터를 다시 만들어 다른 고장나지 않은 디스크로 옮기는 재구축(reconstruction) 작업이 진행중일 때도 사용자 요청에 대한 처리시간이 RAID5 보다 우수하다.

또한 RAID 에서는 하나 이상의 디스크 고장을 대비한 재구축 작업을 안전하게 수행하기 위해 스페어 디스크를 두게 된다. 이 스페어 디스크를 구축하는 방법은 독자적인 디스크를 두거나, 스페어 부분을 여러 디스크에 분산시키는 방법 등이 있다. 이러한 방법의 장점은 고장발생 후 재구축 작업이 진행 중일 때 자료를 분산시켜 저장하기 때문에 부하가 분산되어 재구성 작업 시간을 단축시킬 수 있다.

본 논문에서 제안한 방법은 패리티 디클러스터링 방법의 배치를 가지고 있으면서 기존의 분산 스페어링 기법[1]에서의 분산 스페어링의 배치를 효율적으로 구성하는 방법을 제시하였다. 본 논문에서는 효율적인

분산 스페어링의 배치로 인하여 기존의 디스크와 스페어부분과 고장이 날수 있는 부분간의 엑세스 시간을 줄이는데 목적을 두었다.

2. 패리티 디클러스터링과 분산 스페어링

2.1 패리티 디클러스터링

패리티 디클러스터링은 하나뿐만 아니라 그 이상의 디스크에서 고장이 발생하였을 때와 재구축 작업이 진행중일 때도 정상적인 사용자 요청에 대해 성능 저하없이 처리하기 위하여 패리티 스트라입(parity stripe)의 크기를 디스크 배열의 크기보다 작게 하는 배치 방법이다.

하지만 패리티 디클러스터링 배치 구조는 효율적인 자료 배치면에서 문제점을 가지고 있고, 이러한 문제를 해결하기 위해 6 가지 조건이 제시되었다.[1]

1) 단일 디스크 고장 : 하나의 스트라입의 어떠한 두 유닛도 같은 디스크에 배치하지 않아야 한다.

2) 효율적인 매핑 : 사용자 주소 공간을 디스크 주소로 매핑할 때 수행 시간 및 공간면에서 효율적으로 할 수 있어야 한다.

3) 분산 패리티 : 쓰기 동작의 성능이 떨어지지 않기 위하여 패리티가 모든 디스크에 분산되어야 한다.

4) 분산 재구축 : 재구축 부하를 균등하게 하려면 임의의 디스크 쌍에 대하여 동일한 갯수의 스트라입이 두 디스크에 유닛을 두어야 한다.

5) 큰 쓰기 최적화 : 크기가 한 스트라입이 되는 큰 크기의 쓰기 동작을 처리할 때 연속된 사용자 주소 공간이 디스크 상에서도 연속된 한 스트라입에 매핑이 되어야 한다.

6) 최대 읽기 병렬화 : 연속된 사용자 주소 공간을 읽을 때 최대한 많은 디스크들에서 읽을 수 있도록 매핑되어야 한다.

패리티 디클러스터링 RAID 에 있어서 패리티 스트라입(parity stripe)의 크기를 G, 디스크 배열을 C 라고 정의하고, (G-1)/(C-1)를 디클러스터링 비율(α)이라 정의한다. α 는 디스크 고장시 재구축하는 동안에 각 디스크에서 살아남은 디스크에 대한 비율이다. <그림 1>은 G = 4, C = 5, α = 0.75 인 경우를 나타낸 예이다.

Offset	Disk0	Disk1	Disk2	Disk3	Disk4
0	D0.0	D0.1	D0.2	P0	P1
1	D1.0	D1.1	D1.2	D2.2	P2
2	D2.0	D2.1	D3.1	D3.2	P3
3	D3.0	D4.0	D4.1	D4.2	P4

<그림 1> 패리티 디클러스터링의 예

위의 <그림 1>에서는 Dn,m 은 n 번째 스트라입의 m 번째 유닛(unit)을 의미하고, Pn 은 n 번째 스트라입에 대한 패리티를 의미한다.

하지만 디클러스터 패리티 기법에 있어 <그림 1>처럼 패리티 유닛이 한 디스크에 몰릴 수 있기 때문에 다음과 같은 패리티 디클러스터링 배치를 위한 블록 설계방법을 만들어 내었다[1]. 이 방법에 의하면 v 개의 구별되는 개체를 k 개의 원소를 가진 b 개의 튜플(우리는 이것을 block 로 볼 수 있다.)로 만들고, 각 개체는 정확히 r 개의 튜플에 나타나게 하고, 각 개체의 쌍은 λ p 개의 튜플에 나타나게 하며 b 는 최소값이 되어야 한다. <표 1>은 1 = 5, v=5, k=4, r=4 그리고 λ p = 3 인 경우 블록 설계의 예를 들어 보일 수 있다.

튜플 번호	튜플 내용
Tuple 0	0, 1, 2, 3
Tuple 1	0, 1, 2, 4
Tuple 2	0, 1, 3, 4
Tuple 3	0, 2, 3, 4
Tuple 4	1, 2, 3, 4

<표 1> 블록 디자인 예

<표 1>에서 튜플을 스트라입, 튜플 내의 원소를 디스크로 대응시켜 만든 배치도가 <그림 1>이다. <그림 1>의 배치를 만든, 총 $\binom{5}{4}$ 개의 튜플을 만들어 내는 방법을 완전 블록 설계(complete Block design)라 한다. <그림 2>는 <표 1>의 블록 설계 예를 응용하여 디스크 배치도를 작성한 예이다.

Offset	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	
0	D0.0	D0.1	D0.2	P0	P1	0, 1, 2, 3
1	D1.0	D1.1	D1.2	D2.2	P2	0, 1, 2, 4
2	D2.0	D2.1	D3.1	D3.2	P3	0, 1, 3, 4
3	D3.0	D4.0	D4.1	D4.2	P4	0, 2, 3, 4
4	D5.0	D5.1	P5	D5.2	D6.2	1, 2, 3, 4
5	D6.0	D6.1	P6	P7	D7.2	
6	D7.0	D7.1	D8.1	P8	D8.2	
7	D8.0	D9.0	D9.1	P9	D9.2	
8	D10.0	P10	D10.1	D10.2	D11.2	0, 1, 2, 3
9	D11.0	P11	D11.1	D12.1	D12.2	0, 1, 2, 4
10	D12.0	P12	P13	D13.1	D13.2	0, 1, 3, 4
11	D13.0	D14.0	P14	D14.1	D14.2	0, 2, 3, 4
12	P15	D15.0	D15.1	D15.2	D16.2	1, 2, 3, 4
13	P16	D16.0	D16.1	D17.1	D17.2	
14	P17	D17.0	D18.0	D18.1	D18.2	
15	P18	P19	D19.0	D19.1	D19.2	0, 1, 2, 3
16						0, 1, 2, 4
17						0, 1, 3, 4
18						0, 2, 3, 4
19						1, 2, 3, 4

<그림 2> 블록 설계를 응용한 디스크 배치

2.2 분산 스페어링

분산 스페어링 기법은 독자적인 디스크를 사용하는 스페어링을 RAID 5 나 패리티 디클러스터링 기법에 적용하였을 때 발생하는 문제를 해결하기 위한 방법이다. 우선 분산 스페어링 기법을 설명하기 위해 독자적인 디스크를 사용하는 RAID 5 에 적용하였을 때 발생하는 문제점은 다음과 같다. 재구축 작업 시 독자적인 스페어 디스크에 과도한 부하가 걸리므로 재구축 작업 시간이 길어질 수 있다. 재구축 작업이 진행될 때 사용자 요청을 처리하는 시간이 길어 지기 때문에 입출력이 빈번한 작업에서는 단점을 가질 수 있다.

이러한 문제를 해결하기 위해 패리티 디클러스터링에 분산 스페어링을 적용하여 적절히 스페어 유닛을 디스크에 할당하면 위의 문제를 완화시킬 수 있다. 이때 한 디스크에 고장이 발생하면 고장이 발생한 부분의 자료와 패리티를 복원하여 스페어 유닛으로 지정하고 재구성을 하면 된다. 분산 스페어링을 위한 조건은 아래의 5 개와 같다[1].

1)분산된 스페어 공간 : 스페어 유닛들은 각 디스크에 골고루 분산되어야 한다.

2)재구성 후 fault tolerance : 디스크 고장이 발생하여 스페어 유닛들에 고장이 난 디스크 부분이 복사되어 재구성이 된 상태 후에 fault tolerance 가 있어야 한다.

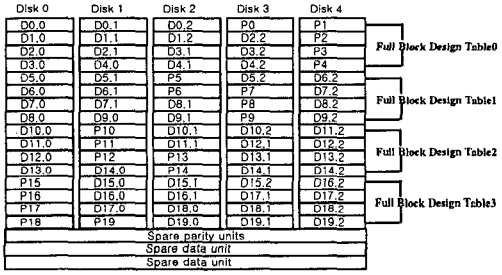
3)재구성 후 패리티의 균등 : 재구성이 된 상태에서도 패리티 유닛들이 골고루 분산되어 있어야 한다.

4)비공간 낭비 : 디스크 공간의 낭비가 없어야 한다.

5)고장난 유닛에 대한 스페어 유닛의 물리적 근접

성 : 재구성 작업 시 디스크 탐색 거리를 줄이기 위하여 스페어 부분이 고장이 난 유닛들에 물리적으로 가까워야 한다.

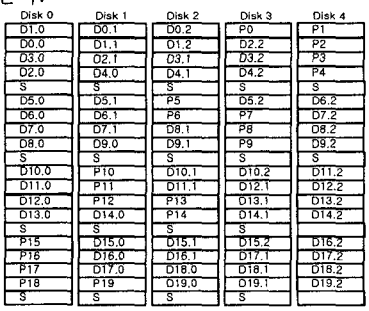
<그림 3>은 패리티 디클러스터링 기법에 분산 스페어링을 응용한 예를 보여준다. <그림 3>에서 스페어 부분은 하나 이상의 전체 블록 디자인 테이블(Full Block Design Table)마다 하나씩 존재한다. 여기서 패리티 유닛을 중심으로 생각하면 한 디스크에 고장이 발생하면 한 전체 블록 디자인 테이블에서 설정되어있는 튜플(r)개 만큼의 패리티 유닛이 손실되고 이들을 v-1 개의 다른 디스크에 분산하여 데이터를 저장한다. 여기서 각 디스크별로 스페어
부분을 두었으므로 데이터에 대한 손실은 없다는 조건 4 를 만족하게 된다. 이 방법은 고장이 발생할 수 있는 부분과 스페어 부분의 물리적 거리가 블록 디자인 테이블에 의해 커질 수 있다는 단점이 발생할 수 있다.



<그림 3> 분산 스페어링을 응용한 예

3. 제안된 방식

본 논문에서는 패리티 디클러스터링 기법을 기본으로 하여 분산 스페어링에서 발생할 수 있는 데이터 부분과 스페어 부분의 물리적 거리가 생길 수 있는 단점을 보완하는 방식을 제안한다. 이를 위하여 분산 스페어부분을 전체 블록 디자인 테이블에 두는 방식을 채택한다.



<그림 4> 개선된 분산 스페어링의 예

<그림 4>는 본 논문에서 제안된 방법에 의한 예를 보여 준다. 이 기법을 사용하면 스페어 유닛이 고장난 디스크의 모든 데이터를 저장할 수 있기 때문에 디스크 공간의 손실은 없다. 그리고 스페어 유닛의 중요한 목적은 고장난 디스크의 모든 데이터와 패리티 데이터를 기록하여야 한다는 것이다. 안전하게 데이터를 스페어 유닛에 저장하기 위해서는 스페어 유닛이 각 디스크에 골고루 분산 되어야 한다는 조건을 <그림 4>는 충분히 만족하고 있다. 전체의 스페어 유닛의 수는 한 디스크의 전체 유닛의 수와 동일하기 때문에 데이터를 안전하게 저장하면서 재구축 작업을 수행할 수 있다.

<그림 5>는 한 디스크가 고장 난 경우 재구축과정을 보여주고 있다. 굵은 상자안의 디스크는 고장난 디스크를 나타낸다. 이 과정을 통해 2.2 장에서 제시한 조건을 만족한다는 것을 알 수 있다.

1) 분산된 스페어 공간 : <그림 5>에서 볼수 있듯이 각각의 디스크에 모두 스페어 유닛을 두어 스페어 공간이 잘 분산되어 있음을 알 수 있다.
2) 재구성 후 fault tolerance : 고장난 디스크의 데이터 유닛과 스페어 유닛들은 고장난 디스크를 제외한 모든 디스크에 저장되어 지기 때문에 fault tolerance 를 보장한다.
3) 재구성 후 패리티의 균등 : 각각의 디스크에 모든 패리티 정보가 저장되어 있으므로 패리티 정보는 균등하게 저장된다.
4) 비공간 낭비 : <그림 4>의 설명에서 볼 수 있듯이 스페어 유닛들은 공간의 낭비없이 효율적으로 사용되고 있다.
5) 고장난 유닛에 대한 스페어 유닛의 물리적 근접성 : 기존의 분산 스페어링 기법에서 처럼 큰 블록 디자인에서 물리적으로 거리가 멀어질 수 있지만 여기서는 각 블록 디자인에 두었기 때문에 물리적 근접성을 가질 수 있다.

<그림 5> 개선된 기법을 사용한 재구축 예

4. 성능 평가

본 논문에서는 3 장에서 제안된 기법을 측정하기 위하여 소프트웨어 RAID 시스템으로 RAID 를 구성하였고 시뮬레이션은 Iozone[7]을 사용하여 디스크에 대한 엑세스 시간을 각각의 다른 크기의 파일에 대해 측정하였다. 시뮬레이션을 하기위한 실험 환경은 <표 2>과 같다.

CPU	Inter-Pentium MMX 200Mhz CPU
Memory	32M RAM
Hard Disk	Fujitsu MPE3136AT(13G) - (1)
	Fujitsu MPE3084AT(8G) - (2)
OS	NetBSD 1.52
RAID card	Iwill side RAID 100(IDE 방식)

<표 2> 실험 환경

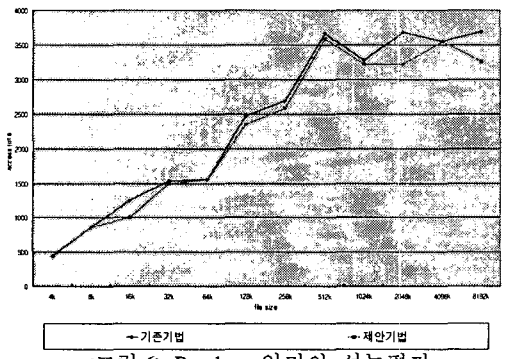
소프트웨어 RAID 는 RAIDframe[6]을 사용하여 패리티 디클러스터링과 분산 스페어링 기법을 구성하였으며 본 논문에서 제안한 기법은 RAIDframe 에서의 rf_decluster.c 파일을 수정하여 RAID 시스템을 구성하였다. 블록 디자인은 4x4 크기의 블록을 사용하여 구성하였다.

3 장에서 제안된 방법에 의한 블록 배치를 위해 다음의 함수들이 변경되었다. 이 변경된 함수들은 분산 스페어링 유닛들을 각각의 전체 블록 디자인 테이블에 넣기 위해 변경되었다.

```
int rf_gcd(int m, int n, int r){
    int t, temp;
    while (m>0) {
      if(m != r){ //블록이 각 블록 테이블의 위치에
                  맞는지 검사하는 부분
        t = n % m;
        n = m;
        m = t;
      }
      else //스페어 유닛 위치이면 1 개를 증가 시켜
      다시 계산
        m = m + 1;
    }
    return(n);
}
```

Iozone 은 디스크의 입출력의 성능을 측정하는 유닉스 기반 툴이다. 본 논문의 시뮬레이션에서는 64K byte 부터 8192K byte 까지 순차적으로 파일의 크기에 대한 엑세스 시간을 얻어 내었다. 본 시뮬레이션은 소프트웨어 RAID 를 사용하여 성능을 측정하였기 때문에 실제 물리적인 RAID 시스템보다는 늦을 수 있음을 밝혀둔다.

<그림 6>은 파일을 random 하게 읽기 작업을 수행할 때의 성능을 나타낸 그래프이다.

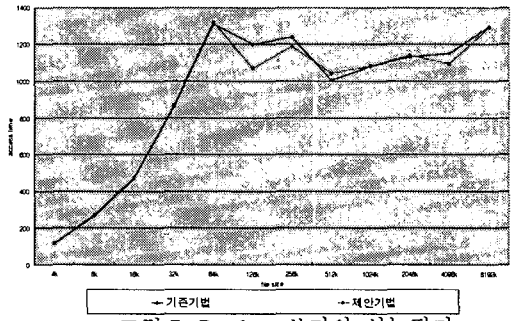


<그림 6> Random 읽기의 성능평가

여기서 볼 수 있듯이 기존에 제안되었던 기법[1]에서 보다 3 장에서 제안되었던 기법이 전반적으로 좋은 성능을 보임을 알 수 있다. 이러한 이유는 [1]에서 밝혀진 스페어 배치 조건을 본 논문에서 제안된 기법이 좀더 잘 만족하기 때문이다.

<그림 7>은 파일을 random 하게 쓰기 작업을 수행할 때의 성능을 평가한 그래프이다. 이 결과에서 부분적으로는 좋은 성능을 가질 경우도 있지만 전반적인 성능은 기존의 기법과 이 제안된 기법간의 차이가 별로 없다. 이는 물리적인 거리는 줄어들지만 연산이 증가되기 때문이다.

본 시뮬레이션에서의 결과에서 읽기 작업에 있어 큰 차이는 아니지만 1%에서 11%까지 각각의 파일 크기에 대해 더 좋은 성능을 보임을 확인할 수 있다. 이것은 스페어 유닛과 각 블록디자인의 물리적 거리를 줄임으로서 나타낼 수 있는 결과이다.



<그림 7> Random 쓰기의 성능평가

5. 결론

본 논문에서는 RAID 시스템의 가용성을 높이기 위한 패리티 디클러스터링 구조에 스페어 유닛을 골고루 분산시키는 개선된 방법을 제안하였다.

이로서 패리티 디클러스터링 기법과 분산 스페어링 RAID 구조에서 재구축 작업시 성능이 떨어지는 문제점을 해결할 수있게 되었다. Iozone 을 이용한 시뮬레이션의 결과에서 전반적으로 읽기작업에 대해 더 좋은 성능을 보임을 확인하였다.

향후작업으로 본 논문에서 생긴 쓰기 작업에서 발생한 문제점을 수정 보완하고 본 논문의 제안을 수학적 분석 모델을 만들어 다른 여러가지 모델과 비교해 보는 방법을 연구 중이다.

참고문헌

[1]M. C. Holland, "On-line Data Reconstruction in Redundant Disk Arrays" PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1994

[2]G. A. Gibson "Tutorial on Storage Technology : RAID and Beyond", Proceeding of the ACM SIGMOD International Conference on Management of Data, 1995.

[3]P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz and D. A. Patterson. "RAID:High-Performance, Reliable Secondary Storage," ACM Computing Survey, Vol.26 No.2 ,pp.145-185, Jun 1994

[4]A. Thomasian and J. Menon, "RAID 5 Performance with Distributed Sparing" IEEE Trans. On Parallel and Distributed Systems, Vol.8 No6, pp. 640-657

[5]G. A. Gibson "Tutorial on Storage Technology : RAID and Beyond", Proceeding of the ACM SIGMOD International Conference on Management of Data, 1995.

[6]G. Gibson, W. Courtright II, M. Holland, and J. Zelenka, "RAIDframe: Rapid prototyping for disk arrays", Computer Science Technical Report CMU-CS-95-200, Carnegie Mellon University, 1995.

[7]"Iozone filesystem benchmark" – (http://www.iozone.org)