패리티 디클러스터링 RAID 시스템에서의 성능 개선 방안

장 태 무†

요 약

RAID는 고 병렬성과 고 가용성을 목표로 하는 대용량 저장 매체를 구축하는 방법이다. 패리티 디클러스터링을 이용한 RAID는 특히 고장이 발생한 경우에도 성능의 저하를 최소화하여 고 결함 허용도 및 가용성을 높일 수 있는 저장 장치를 구축할 수 있는 기법으로 널리 연구되어 왔다. 본 논문에서는 이러한 패리티 디클러스터링을 사용한 RAID에 스페어 유닛을 분산시킨 새로운 구성을 제안하고, 특히 이러한 분산 스페어링이 고장이 없는 정상 상태에서도 성능 개선에 유용함을 보인다. 본 논문에서 제안된 방법의 유효성은 시뮬레이션 방식으로 입증하였으며, 전반적으로 정상상태의 성능을 5-15% 정도 높일 수 있음을 알 수 있다.

Performance Improvement on RAID System with Parity Declustering

Tae-Mu Chang†

ABSTRACT

RAID systems have been used as a mass storage system with high parallelism and availability. Especially RAID systems with parity declustering are widely studied as a technique to provide high fault tolerancy and availability by reducing performance degradation in case of disk failures. In this paper, a new organization of parity declustering with distributed spare units is proposed. And in normal mode where there are no failures, it is shown that these organization can improve the performance of RAID systems. By simulation methods, it is proved that the performance of RAID system in normal mode is improved by 5 to 15%.

1. 서 론

RAID(Redundant Arrays of Inexpensive Disks)는 1980년대 이후 대표적인 대용량 저장 수단으로 많은 응용 분야에서 활용되고 있으며, 시장 규모도 급신장하고 있다[1, 2]. 반도체 기술의 급속한 발전으로 마이크로프로세서의 성능은 매년 25~30% 정도 증가하고 있으나, 기계적인 부품을 가진 자기 디스크의 성능 향상 속도는 5%에 불과하다[3]. 따라서 입출력 시스템이 상대적으로 느리기 때문에 발생하는 전체 컴퓨터 시스템의 성능 저하를 막기 위하여 병렬성과 고 가용성을 가진 RAID의 중요도는 날로 크게 부각되고 있다.

RAID 시스템의 구현 방법은 신뢰도를 높이기 위한 패리티 등의 추가정보(redundancy)의 배치 방법에 따라 여러 가지가 있다[4]. 흔히 RAID 단계(level)라고 부르는 것이 그것으로 단계 1에서 6까지가 많이 사용되고 있으며, 이들은 성능, 신뢰도, 그리고 구축비용 면에서

※ 이 논문은 동국대학교 교원 해외 연수 지원에 의하여 이루어졌음.
† 정 회 원 : 동국대학교 컴퓨터·멀티미디어공학과 교수
논문접수 : 1999년 9월 6일, 심사완료 : 1999년 12월 27일

차이가 있다.

패리티 디클러스터링(parity declustering) 방법은 RAID의 가용성을 높이는 방안의 하나로 많이 연구된 배치(layout) 방법이다. 즉 디스크 배열 중 한 디스크에 고장이 발생하였을 때 및 그 디스크에 있던 자료를 다시 만들어 다른 안전한 디스크로 옮기는 재구축(reconstruction) 작업이 진행 중일 때도 사용자 요청에 대한 처리 시간 면에서 RAID 5 보다 우수한 것으로 알려져 있다[1, 3, 5, 6, 7, 8]. 반면에 디클러스터링 방법은 적절한 디스크 배치를 위한 표를 기억하는 메모리 및 위치를 찾기 위한 시간적인 오버헤드가 필요하다. 또한 고장 상태가 아닌 정상 상태(normal mode)에서는 RAID 5보다 성능이 떨어 질 수 있는 약점이 있다[3].

RAID에서는 하나 이상의 디스크 고장을 대비하여 재구축 작업을 수행하기 위한 스페어 디스크를 두게 된다. 스페어 디스크의 구축 방법은 독자적인 디스크를 두거나, 스페어 부분을 여러 디스크에 분산시키거나, 패리티를 이중으로 두는 방법 등이 제시되어 있다[9]. 디스크 고장 발생 후 재구축 작업이 진행 중일 때 분산 스페어링 방법을 이용하게 되면 부하가 분산되므로 재구축 작업 시간을 단축할 수 있다[1, 3, 10, 11]. 물론 이 경우 두 번째 고장이 일어날 것을 대비하여 스페어 부분에 기록된 자료를 안전한 장소로 대피시키는 오버헤드가 따르지만 디스크 접근 자체가 한꺼번에 몰리는 경향이 있어 디스크 유휴 시간이 많고, 이러한 유휴 시간을 적절히 이용한다면 무시할 수 있다[1].

본 논문에서는 디클러스터링 방법의 배치를 갖는 디스크 배열에서 분산 스페어링을 도입하는 방안을 제시한다. 기존의 연구[3]에서도 이러한 방법이 제안되었으나, 본 논문에서는 고장이 없는 정상 상태에서 성능을 개선할 수 있는 새로운 분산 스페어링 방안을 제시하고, 분산된 스페어 부분을 정상 상태에서도 활용하여 디클러스터링의 단점 중의 하나인 정상 상태에서 성능이 떨어질 수 있는 점을 보완하고자 하였다.

2. 패리티 디클러스터링의 특성

패리티 디클러스터링은 기본적으로 하나 이상의 디스크에서 고장이 발생하였을 때와 재구축 작업이 진행 중일 때도 정상적인 사용자 요청을 성능의 큰 저하가 없이 처리하기 위하여 패리티 스트라입(stripe)의 크기(G)를 디스크 배열의 크기(C) 보다 작게 하는 배치 방법이다. 이때 $(G-1)/(C-1)$를 디클러스터링 비율(α)이라고 정의한다. (그림 1)은 $G=4$, $C=5$, $\alpha=0.75$인 경우의 예이다. (그림 1)에서 Dn,m은 n번째 스트라입의 m번째 유닛(unit)을 의미하고, Pn은 n번째 스트라입에 대한 패리티이다.

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

이러한 패리티 디클러스터링 배치 구조는 효율적인 자료의 배치라는 새로운 문제를 야기한다. 이상적인 자료의 배치에 대하여 [7]에서 다음과 같은 여섯 가지 조건을 제시되었다.

(조건 A1) 단일 디스크 고장에 대하여는 올바르게 고치는 것이 가능하도록 한 스트라입의 어떠한 두 유닛도 같은 디스크에 배정하지 않아야 한다.

(조건 A2) 쓰기 동작의 성능이 떨어지지 않기 위하여 패리티가 모든 디스크에 분산되어야한다.

(조건 A3) 재구축 부하를 균등하게 하려면 임의의 디스크 쌍에 대하여 동일한 개수의 스트라입이 양 디스크에 유닛을 두어야한다.

(조건 A4) 사용자 주소 공간을 디스크 주소로 매핑(mapping)할 때 수행 시간 및 공간 면에서 효율적으로 할 수 있어야한다.

(조건 A5) 크기가 한 스트라입이 되는 큰 크기의 쓰기 동작을 처리할 때 연속된 사용자 주소 공간이 디스크 상에서도 연속된 한 스트라입에 매핑이 되어야 효율적인 처리를 행할 수 있다.

(조건 A6) 연속된 사용자 주소 공간을 읽을 때 최대한 많은 디스크들에서 읽을 수 있도록 매핑되어야 한다.

그러나 위의 여섯 가지 조건을 모두 만족하는 이상적인 배치는 존재하지 않는다. 일반적으로 패리티 디클러스터링 배치를 위하여 블록 설계(block design) 방법론을 사용한다. 즉 v개의 구별되는 개체를 k개의 원소를 가진 b개의 튜플(tuple)로 만들되, 각 개체는 정확히 r개의 튜플에 나타나게 하고, 각 개체의 쌍은 정

확히 λ_p개의 튜플에 나타나게 하며, b는 최소치가 되어야 한다[7].

예를 들어 b=5, v=5, k=4, r=4 그리고 λ_p=3인 경우 블록 설계의 결과 <표 1>과 같은 튜플들이 만들어진다.

〈표 1〉 블록 설계의 결과

튜플 번호	튜플의 내용
0	0 1 2 3
1	0 1 2 4
2	0 1 3 4
3	0 2 3 4
4	1 2 3 4

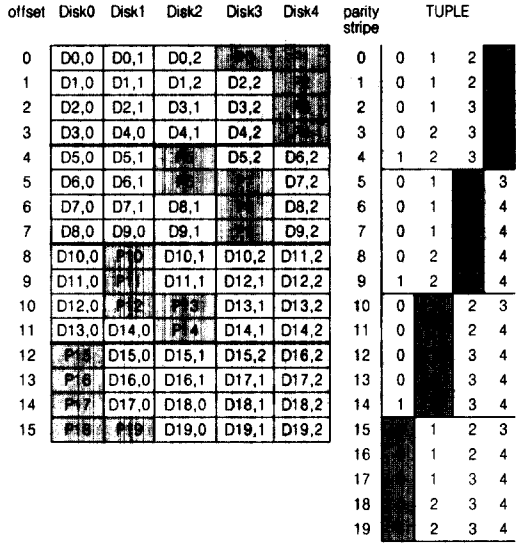
<표 1>에서 튜플을 스트라입, 튜플 내의 원소를 디스크로 대응시켜 만든 배치도가 (그림 1)이다.

(그림 1)의 배치를 만든, 총 $\binom{C}{G}$개의 튜플을 만들어 내는 방법을 완전 블록 설계(complete block design)라고 한다. 그러나 대규모의 배열에서 특히 작은 α값인 경우 튜플 표의 크기가 커서 구현하기 어려우므로 그 중 일부만 사용하는 BIBD(Balanced Incomplete Block Design) 방법을 이용하기도 한다[3, 5, 6, 7]. BIBD를 이용한 구조는 앞에서의 세 조건(A1, A2, A3) 및 A5 또는 A6을 만족하는 것으로 알려져 있다[5].

(그림 1)에서 보면 패리티는 디스크 5에 몰려 있어서 패리티 분산이 이루어져있지 않다. 이러한 경우 쓰기 동작의 성능이 저하되게 된다. 실제의 디클러스터링 방법에서는 블록 설계에서 만들어진 튜플을 반복시키되 패리티를 두는 순서를 달리하여 디스크 배치를 행한다. (그림 1)의 표를 블록 설계표(BDT : Block Design Table)이라 하고, 이를 k개만큼 반복시켜 전체 블록 설계표(FBDT : Full Block Design Table)를 구한다[3, 7].

(그림 2)에서 위의 과정을 설명하였다. (그림 2)의 오른쪽 부분에서 b개의 튜플 그룹으로 BDT를 만들고, 이를 k번 반복시켜 $b \cdot k$개의 튜플 엔트리를 가진 FBDT를 생성한다. 각 튜플 그룹에서 패리티의 위치를 (그림 2)에서와 같이 다르게 적용하면 왼쪽 부분의 배치도를 얻게 된다.

패리티 디클러스터링의 이러한 특성에서 볼 때, 패리티가 전체적으로 균등하다 하더라도 가까운 거리 내에서는 균등할 수 없다. 이러한 면에서 일반적인 RAID 5보다 정상 상태에서, 특히 쓰기 동작이 많은 경우 성능이 떨어 질 수 있는 요인이 있다.



(그림 2) 블록 설계 표에서 디스크 배치도 작성

3. 분산 스페어링을 이용한 디클러스터링

3.1 분산 스페어링의 필요성과 기존의 방법

일반적으로 독자적인 디스크를 사용하는 스페어링을 RAID 5에 적용하였을 때 발생할 수 있는 문제점은 다음과 같다. 우선 독자적인 디스크는 정상 동작일 때 쉬고 있으므로 자원의 활용도가 떨어지며, 병렬성이 낮아진다. 또한 재구축 작업 시 독자적인 스페어 디스크에 과도한 부하가 걸리므로 재구축 작업 시간 자체도 길어진다. 재구축 작업이 진행될 때는 사용자 요청을 처리하는 시간이 길어 질 수밖에 없으므로 입출력이 빈번한 응용에서는 디클러스터링의 장점을 저해할 수 있다. 특히 낮은 α값을 갖는 디클러스터링 구조에서는 더 큰 문제가 될 수 있다[3, 8].

분산 스페어링을 패리티 디클러스터링에 적용하게 되면 위의 문제를 완화할 수 있다. 즉 스페어 유닛을 적절하게 디스크에 할당(allocation)하고, 한 디스크에 고장이 나면 고장이 난 부분의 자료와 패리티를 복원하여 스페어 유닛으로 지정(assignment)하여 재구성(reconfiguration)해야 한다. [3]에서 소개된 분산 스페어링의 조건은 다음과 같다.

(조건 B1) 스페어 유닛들은 각 디스크에 골고루 분산되어야 한다.

(조건 B2) 디스크 고장이 발생하여 스페어 유닛들에 고

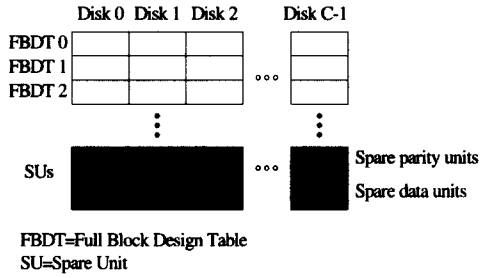
장이 난 디스크 부분이 복사되어 재구성이 된 상태에서도 향후 고장에 대해 감내성이 있어야 한다.

(조건 B3) 재구성이 된 상태에서도 패리티 유닛들이 골고루 분산되어 있어야 한다.

(조건 B4) 디스크 공간의 낭비가 없어야 한다.

(조건 B5) 재구성 작업 시 디스크 탐색(seek) 거리를 줄이기 위하여 스페어 부분이 고장이 난 유닛들에 물리적으로 가까워야 한다.

패리티 디클러스터링 구조에서 분산 스페어링을 도입한 연구는 [3]이외에는 거의 없는 실정이다. (그림 3)은 [3]에서 사용한 분산 스페어링 방법을 도시한 것이다.



(그림 3) 스페어 유닛의 할당[3]

(그림 3)에서 스페어 부분은 여러 개의 FBDT마다 하나씩 존재한다. 패리티 중심으로 볼 때 한 디스크에 고장이 발생하면 한 FBDT마다 r개의 패리티 유닛이 손상되고 이들을 v-1개의 다른 디스크에 분산하여 대피시켜야한다. 따라서 LCM(v-1, r)/r개의 FBDT를 모으고 이들 그룹마다 패리티를 위한 스페어 부분을 두는 것이 저장장치의 손실이 없다(조건 B4). 패리티가 아닌 데이터 부분도 마찬가지로 같은 개수의 FBDT마다 스페어 부분을 두면 된다.

위의 방법은 분산 스페어링이 만족해야하는 여러 조건들을 만족하지만 고장이 날 수 있는 부분과 스페어 부분 사이의 물리적인 거리가 설계에 따라서는 커질 수 있는 단점이 있다.

3.2 제안된 분산 스페어링 방법

분산 스페어링의 또 하나의 장점으로 생각할 수 있는 것은 분산하는 방법과 운영 방법에 따라 스페어링 부분을 이용하여 정상 상태에서도 성능을 개선할 수 있다는 점이다. 본 논문에서는 이러한 스페어 부분을 이용하여 2장에서 이야기된 부분적인 패리티의 불균형 문제를 완화시켜 정상 상태에서의 성능을 높이는 운영 방안도 고려한다. 즉 현재 사용 중인 디스크 내의 패리티에 접근해야하는 요청이 있을 때 이를 사용되고 있지 않는 디스크의 스페어 유닛으로 우선 접근하여 해결하고, 나중에 제대로 복원하는 방법이다.

이를 위하여 재구축 작업이 진행 중인 경우도 마찬가지만 데이터나 패리티가 존재하는 부분과 스페어 부분이 물리적으로 가까운 것이 필요하다.

본 논문에서 사용한 분산 스페어링은 기존의 배치도에서 스페어 유닛들을 대각선 방향의 좌대칭형(left symmetric)으로 끼워 넣고, 그 위치부터 오른쪽의 유닛들을 오른쪽으로 하나씩 쉬프트(shift)하는 방법을 사용한다. (그림 4)와 (그림 5)에서 그 과정을 보였다.

offset	Disk0	Disk1	Disk2	Disk3	Disk4	Disk5
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	
4	D5,0	D5,1	P5	D5,2	D6,2	
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	

(그림 4) 패리티 디클러스터링 배치

offset	Disk0	Disk1	Disk2	Disk3	Disk4	Disk5
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
4	D5,0		D5,1	P5	D5,2	D6,2
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

(그림 5) 분산 스페어를 추가한 배치

이 방법에서 우선 필요한 스페어 유닛의 양은 한 디스크의 용량 만큼이므로 디스크 공간 상의 손실은 없다. 패리티 디클러스터링의 기본 요건인 고장 감내성이 있는지를 살펴 본다. 위와 같이 스페어 유닛들을 삽입하고 그 유닛부터 오른쪽으로 이동하므로 만일 동일한 스트라입 내의 유닛들이 한 디스크로 모여지려면 오프셋이 2 이상 떨어져 있는 경우 뿐이다. 그런데 α값이 1보다 작은 패리티 디클러스터링 구조에서는 이러한 일

이 발생할 수 없다. 따라서 한 스트라입에 속한 어떤 유닛들도 동일한 디스크에 존재할 수 없다.

스페어 부분의 중요한 목적은 한 디스크가 고장이 났을 때 그 디스크에 있는 모든 데이터와 패리티 유닛들을 기록할 수 있어야한다는 것이다. 이러한 재구축 작업이 완료되었을 때도 마찬가지로 패리티 디클러스터링의 특성인 고장 감내성이 있어야하므로 한 스트라입의 여러 유닛들은 다른 디스크에 존재해야한다. 우선 스페어 유닛들의 수가 한 디스크의 용량과 일치하고, 두 개의 BDT 사이에는 동일한 스트라입의 유닛들이 존재할 수 없기 때문에 고장 감내성은 보장할 수 있다. 즉 현 BDT에서 보관해야할 유닛들을 다음 BDT의 스페어 유닛들에 보관하면 고장 감내성은 보장된다. 본 논문에서 사용한 방법은 (그림 6)에서 소개하였다. 이 경우는 (그림 5)의 디스크 2에 고장이 발생하였다고 가정하고 그 재구축 과정을(그림 6)에서 보인 것이다.

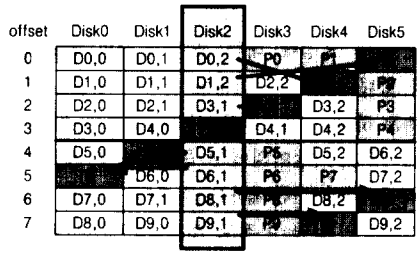
(그림 6)에서 대부분의 경우 고장이 난 부분을 오프셋이 동일한 인접한 디스크로 옮김으로써 재구축작업을 진행한다. 그러나 그렇게 할 수 없는 경우도 있다. 예를 들어 D1,2의 경우 Disk4로 이동하게되면 P1과 동일한 디스크에 위치하게되어 (조건 A1)을 만족하지 못 하게 된다. 이 경우 D0,2와 D1,21의 보관 위치를 바꾸면 되는데 이는 <표 1>의 블록 설계표와 스페어 블록의 위치를 관찰하여 해할 수 있다. 또 이를 위하여 매핑표를 위한 부가적인 기억 용량은 필요하지 않다.

이러한 분산 스페어링 방법이 앞에서 언급한 여러 조건을 만족하는지를 따져 본다.

(조건 B1) 스페어 유닛들은 각 디스크에 골고루 분산되어 있다. 이는 대각선 방향으로 스페어 유닛을 삽입하는 것을 반복하기 때문에 자명하다. 이때 스페어 유닛을 삽입하는 것은 (v+1)의 주기를 갖는다.

(조건 B2) 만일 어느 한 디스크가 고장이 나는 경우 앞에서 설명한대로 적절한 위치의 스페어 유닛들을 이용하여 고장 감내성을 보장할 수 있다.

(조건 B3) (그림 5)의 부분적인 배치도에서는 패리티의 분



offset	Disk0	Disk1	Disk2	Disk3	Disk4	Disk5
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
4	D5,0		D5,1	P5	D5,2	D6,2
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

(그림 6) 디스크 2가 고장 난 경우 재구축과정

offset	Disk0	Disk1	Disk2	Disk3	Disk4	Disk5
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
4	D5,0		D5,1	P5	D5,2	D6,2
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
9	D11,0	P11		D11,1	D12,1	D12,2
10	D12,0		P12	P13	D13,1	D13,2
11		D13,0	D14,0	P14	D14,1	D14,2
12	P15	D15,0	D15,1	D15,2	D16,2	
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
16	D20,0		D20,1	D20,2	P20	P21
17		D21,0	D21,1	D21,2	D22,2	P22
18	D22,0	D22,1	D23,1	D23,2	P23	
19	D23,0	D24,0	D24,1	D24,2		P24
20	D25,0	D25,1	P25		D25,2	D26,2
21	D26,0	D26,1		P26	P27	D27,2
22	D27,0		D27,1	D28,1	P28	D28,2
23		D28,0	D29,0	D29,1	P29	D29,2

offset	Disk0	Disk1	Disk2	Disk3	Disk4	Disk5
24	D30,0	P30	D30,1	D30,2	D31,2	
25	D31,0	P31	D31,1	D32,1		D32,2
26	D32,0	P32	P33		D33,1	D33,2
27	D33,0	D34,0		P34	D34,1	D34,2
28	P35		D35,0	D35,1	D35,2	D36,2
29		P36	D36,0	D36,1	D37,1	D37,2
30	P37	D37,0	D38,0	D38,1	D38,2	
31	P38	P39	D39,0	D39,1		D39,2
32	D40,0	D40,1	D40,2		P40	P41
33	D41,0	D41,1		D41,2	D42,2	P42
34	D42,0		D42,1	D43,1	D43,2	P43
35		D43,0	D44,0	D44,1	D44,2	P44
36	D45,0	D45,1	P45	D45,2	D46,2	
37	D46,0	D46,1	P46	P47		D47,2
38	D47,0	D47,1	D48,1		P48	D48,2
39	D48,0	D49,0		D49,1	P49	D49,2
40	D50,0		P50	D50,1	D50,2	D50,2
41		D51,0	P51	D51,1	D52,1	D52,2
42	D52,0	P52	P53	D53,1	D53,2	
43	D53,0	D54,0	P54	D54,1		D54,2
44	P55	D55,0	D55,1		D55,2	D56,2
45	P56	D56,0		D56,1	D57,1	D57,2
46	P57		D57,0	D58,0	D58,1	D58,2
47		P58	P59	D59,0	D59,1	D59,2

(그림 7) 패리티와 스페어 유닛들이 균등하게 분포된 배치도

포가 균일하지 않게 보이지만, FBDT와 스페어 유닛 삽입 주기가 일치하도록 표의 크기를 연장하면 패리티의 분포는 균일해진다. 그것은 균일하게 패리티가 분포되어 있는 상태에서 대각선 방향으로 균일하게 자리 이동을 해 나가기 때문이다. FBDT의 크기는 kr이고, 스페어 유닛 삽입 주기는 $(v+1)$이므로 패리티가 균일해지는 주기는 LCM(kr, $(v+1)$)이 된다.

(그림 7)은 b=5, v=5, r=4, k=4이고 <표 1>의 설계표를 이용한 경우의 배치도이다. 이때 패리티와 스페어 유닛이 균등해지는 주기는 LCM(4*4, 6)=48이다.

(조건 B4) 스페어 유닛들의 총 크기는 한 디스크 용량과 같으므로 저장 용량 손실은 없다.

(조건 B5) 스페어 유닛들은 데이터 및 패리티 부분과 물리적으로 가까운 것은 3.1절의 기존 방법과 비교했을 때 자명하다.

3.3 정상 상태에서의 성능 개선

본 논문에서 제시한 좌대칭형 분산 스페어링 방법은 데이터 및 패리티를 v개의 디스크에서 $(v+1)$개의 디스크로 분산시키므로 작은 크기의 읽기 및 쓰기 동작의 병렬성을 증가시킨다.

읽기 동작의 경우 동시에 읽을 수 있는 디스크의 수가 $(v+1)/v$ 만큼 늘어나게 되며, 그만큼 동일한 디스크에의 읽기 충돌의 확률을 줄여줄 것이다.

쓰기 동작의 성능은 패리티의 분산도에 비례할 것이므로 각 디스크에서의 패리티의 전체 용량에 대한 비율이 그 척도가 될 수 있을 것이다. 일반 패리티 디클러스터링 구조에서는 FBDT마다 디스크 당 r개의 패리티 유닛이 존재하므로,

$$r/kr = 1/k \quad (1)$$

이 된다. 그런데 본 논문에서 제안된 분산 스페어링 방법에서는 패리티가 균일해지는 주기마다 존재하는 패리티 유닛 수를 계산해 보면,

$$rv \times \text{LCM}(kr, (v+1))/kr$$

이다. 이때 디스크의 전체 유닛 수는

$$\text{LCM}(kr, (v+1)) \times (v+1)$$

이므로 패리티 밀도는

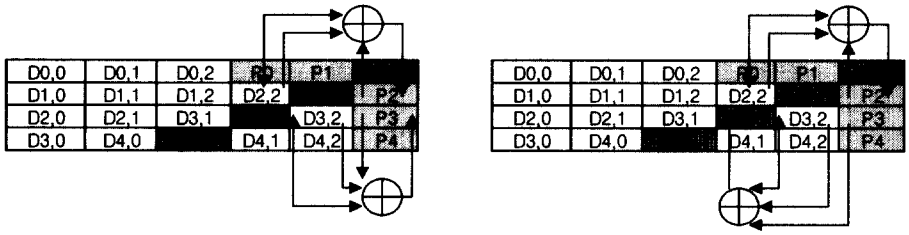
$$1/k \times (v/(v+1)) \quad (2)$$

이 된다.

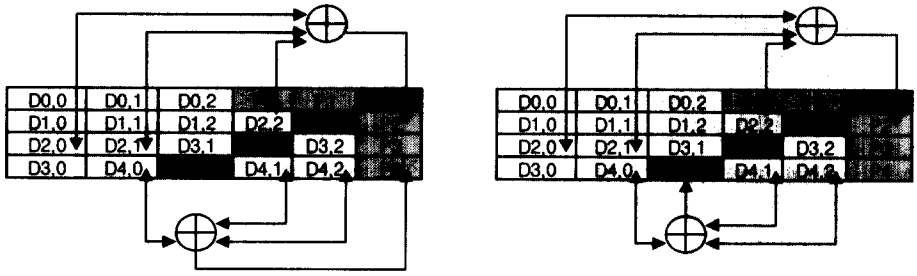
앞의 경우에서와 같이 b=5, v=5, r=4, k=4인 경우 식 (1)과 식 (2)를 계산해보면 식 (2)의 값이 16% 정도 작은 값이 된다. 이 값이 실제 성능에 얼마나 영향을 주는지는 전체 요청에 대한 쓰기 동작의 비율과 밀접한 관계가 있다.

이러한 분산 스페어링을 사용함으로써 기대할 수 있는 성능 상의 이점이외에도 물리적으로 자료와 패리티에 가까운 위치에 존재하는 스페어 유닛을 활용함으로써 기대할 수 있는 이점도 있다. RAID는 읽기 성능은 우수하나 쓰기인 경우 패리티 갱신 작업이 수반되어야 하므로 성능이 떨어진다. 따라서 패리티의 균일한 분산이 필요하다. 패리티 디클러스터링의 구조 상 가까운 영역의 경우 패리티가 분산되어 있지 않으므로 문제가 발생할 수 있다. 이러한 점에서 본 논문에서 제시한 좌대칭형의 스페어 유닛들을 정상 상태에서 활용한다면 더 성능을 개선할 수 있다.

(그림 8)은 작은 개수의 유닛에 대한 쓰기 작업의 읽기-수정-쓰기(Read-Modify-Write) 동작에서 좌측 그림에서 P2와 P3이 같은 디스크에 있어 충돌이 발생한 경우이다. 이때 그림의 우측과 같이 일단 스페어



(그림 8) 읽기-수정-쓰기 동작에서의 최적화



(그림 9) 재구축-쓰기 동작에서의 최적화

유닛에 P3을 기록하는 수정된 동작을 보인 것이다. 이 경우 한 번의 쓰기 동작이 지연 없이 진행된다.

(그림 9)는 큰 개수의 유닛들에 대한 쓰기 작업의 재구축-쓰기(reconstruct write) 동작을 보인 것인데 역시 좌측의 그림에서 P2와 P4는 같은 디스크에 있는데, P4를 스페어 유닛에 기록함으로써 한 번의 쓰기를 지연없이 진행한다.

스페어 유닛에 기록된 패리티는 디스크 유휴 시간에 제 위치로 옮겨야하는데, 이는 디스크 유휴 시간을 이용함을 가정한다. 본 논문에서 구현한 시뮬레이터에서도 디스크 유휴 시간이 50%정도이며, 실제 환경에서도 큰 무리가 없는 가정이라고 판단된다.

스페어 유닛에 기록된 패리티에 대한 정보를 기억하는 표의 기억 장치 오버헤드가 있다. 만일 전체 스페어 유닛을 다 기록한다면 디스크 용량을 1GByte, 유닛 크기를 8KByte로 가정하면 엔트리의 개수는 128K 개가 된다. 한 엔트리에 들어가는 정보는 기록되었다는 사실과, 동일한 오프셋의 어느 패리티인가 하는 정도이므로 1 바이트면 충분하다. 또한 모든 스페어 유닛들이 동시에 사용되지는 않으므로 이 표는 전체 스페어 유닛을 다 기억할 필요는 없다. 크기를 고정시키고 캐쉬(cache)처럼 운영할 수도 있다. 따라서 디스크 용량이 커진다 하더라도 큰 오버헤드는 아니라고 판단된다.

좌대칭형으로 분산된 스페어 유닛을 활용하는 방법을 사용하였을 때의 성능 향상 정도는 작업 부하와 밀접하게 관련되어 있다. 디스크 요청 중 쓰기의 비율이 높을수록, 디스크 요청의 크기가 작을수록, 지역성이 높을수록 커질 것이다.

4. 성능 분석

본 논문에서는 3장에서 이야기한 스페어 유닛을 분산시켜 정상 상태에서 성능이 개선됨을 보이기 위하여 시뮬레이션 방법을 사용하였다. 사용한 시뮬레이터는 DiskSim[12, 13]을 본 논문의 디스크 구성에 맞도록 수정한 것이다. DiskSim은 실행 시간 면에서 효율적이고, 다양한 입출력 장치 환경에 사용할 수 있는, 저장 장치 서브시스템(subsystem)의 성능을 시험할 수 있는 시뮬레이터이다. 시뮬레이터의 구성은 장치 구동기(device driver), 버스, 제어기(controller), 그리고 디스크의 네 부분으로 되고, 부가적으로 스케쥴러(scheduler)와 캐쉬가 들어가게 되어 있다. 본 논문에서는 디스크 자체의 성능만을 고려하기 위하여 캐쉬는 고려하지 않았다.

작업 부하는 DiskSim의 내부 인공 트레이스 합성 기능을 이용하였으며, 이는 UNIX 다수 사용자 환경에서 대형 소프트웨어를 여러 명의 사용자가 수정 및 디버그 작업을 하는 동작을 모방한 것이다. 이 작업 부하는 [13]에서 분석적 모형(analytic model)로 계산한 값과 시뮬레이션 결과가 거의 동일한 것으로 밝힌 바 있다. <표 2>는 본 논문에서 사용한 작업 부하의 특성이다.

본 논문에서 사용한 시뮬레이션 변수는 <표 3>과 같다. 요청 크기가 평균적으로 5KB 정도이므로 스트라입 유닛의 크기는 이보다 조금 큰 8KB로 하였다. 모델링한 디스크는 HPC 3323A이며 주요 특성은 <표 3>에 있다.

〈표 2〉 작업 부하의 특성

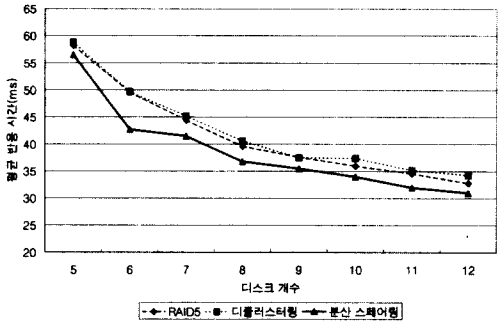
요청 간격	평균 24.24 ms, 표준 편차 27.46 ms, 최대 간격 444.8 ms
요청 크기	평균 4.9 KB, 표준 편차 3.8 KB, 최대 38 KB
읽기/쓰기 비율	0.80 : 0.20
요청 회수	매 시뮬레이션마다 200,000회

(그림 10)은 RAID5, 패리티 디클러스터링, 그리고 본

〈표 3〉 시뮬레이션 변수

디스크 배열	스트라입 유닛의 크기 : 8 KB 구성 방법 : 디스크 수를 다르게 한 RAID5, 패리티 디클러스터링 및 분산 스페어링 디스크 스케쥴링 : LBN 기반의 C-LOOK[14] 디스크의 수 : SCSI 형의 버스로 최대 15개까지 연결
디스크 (HPC 3323A)	최대 용량 : 1.03 GB 실린더 2982, 표면 7, 섹터 크기 512 B, 8 존(zone) 탐색시간 : $0.42 \cdot \sqrt{\text{거리}-1} + 0.01 \cdot (\text{거리}-1) + 1.178$ (ms) 평균 탐색시간 : 10 ms 헤드 회전 속도 : 5400 rpm

논문에서 제시한 좌대칭형 분산 스페어링을 사용한 경우의 평균 반응 시간(response time)에 대한 시뮬레이션 결과이다. 반응 시간은 장치 구동기에서 출발한 요청이 다시 장치 구동기에서 완료될 때까지 시간을 측정한 것으로, 이 속에는 제어기 큐(queue)에 있는 시간, 버스 전송 시간, 디스크 입출력 시간을 모두 더한 값이다.



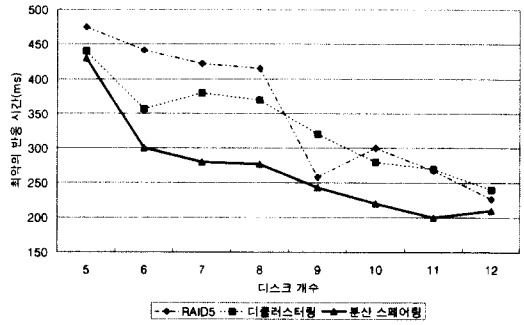
(그림 10) 디스크 수에 대한 평균 반응 시간

(그림 10)에서 분산 스페어링에서 사용하는 디스크의 수는 그림에서 표시된 수치보다 1개가 많은 것이다. 독자적인 스페어링을 사용하는 디클러스터링의 경우, 정상 상태에서는 스페어 유닛들이 사용되지 않음을 고려한 것이다.

분산 스페어링을 사용하는 경우 특히 디스크 수가 작은 경우 정상 상태에서 성능 향상이 두드러졌다. 반응 시간으로 따졌을 때 대체로 5~15%의 감소가 있음을 확인할 수 있다. 그것은 디스크 수가 하나 많아짐으로써 발생하는 병렬성의 증가와 충돌되는 쓰기 요청에서 스페어 유닛을 사용하는 방법에 기인한다. 시뮬레이션 결과, 충돌되는 요청이 존재할 확률은 16%로 관찰되었다.

RAID5와 디클러스터링의 성능 상의 차이는 발견되지 않았다. 그것은 요청의 크기가 비교적 작은 작업 부하 때문인 것으로 판단된다.

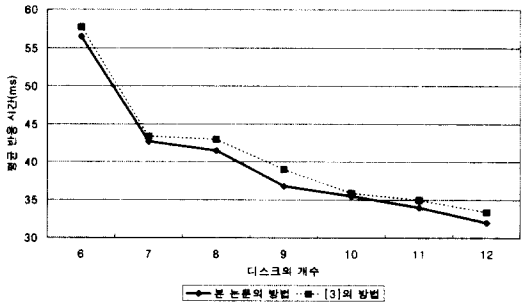
(그림 11)은 동일한 조건에서 최악의 반응 시간을 측정한 결과이다.



(그림 11) 디스크 수에 대한 최악의 반응 시간

최악의 반응 시간 면에서 RAID5와 디클러스터링의 경우 뚜렷한 차이는 발견되지 않았다. 그러나 분산 스페어링을 사용하는 경우는 분명히 줄어들고 있음을 확인할 수 있다. 특히 사용자 요청이 빈번하고, 반응 시간이 빨라야하는 응용 분야에서는 의미가 있는 결과라고 보여진다.

(그림 10)과 (그림 11)로부터 본 논문에서 구현한 좌대칭형 분산 스페어링을 이용한 패리티 디클러스터링 방법이 정상 상태에서 디클러스터링 방법에 비하여 성능을 개선할 수 있음을 보았다. 다음의 (그림 12)는 동일한 분산 스페어링이면서 스페어 유닛을 두는 방법이 다른 두 방법을 비교해 보았다. 하나는 [3]에서 제시한



(그림 12) 디스크 개수에 따른 두 가지 분산 스페어링 방법의 비교

방법이며, 두 번째는 본 논문에서 제시한 방법이다. [3]에서는 정상 상태에서는 스페어 유닛들을 활용하지 않으나 본 논문에서 제시한 방법과 비교를 위하여 충돌이 있는 경우 스페어 유닛들을 활용한다고 가정하였다.

두 방법에서 큰 차이는 아니지만 5~10% 정도는 일관되게 본 논문의 방법에서 성능이 낮다는 것을 확인할 수 있다. 이 차이는 스페어 유닛들의 위치가 패리티 유닛들에 얼마나 가까운가에 의하여 나오는 값이다.

5. 결 론

본 논문은 디스크의 가용성을 높이는 것이 목적인 패리티 디클러스터링 구조에 스페어 유닛들을 분산 시켜 삽입하는 새로운 방법을 제시하였다. 이로써 고장이 발생하여 재구축 작업이 진행 중일 때 사용자 요청을 신속하게 처리해 주고 재구축 작업을 빠르게 할 뿐 아니라, 고장이 없는 정상 상태에서도 성능이 개선될 수 있도록 하였다. DiskSim을 수정한 시뮬레이터를 이용한 시뮬레이션 결과 본 논문에서 제시한 좌대칭형의 분산 스페어링 방법이 단순 디클러스터링에 비하여 5~15% 정도 평균 반응 시간을 줄일 수 있음을 확인하였다. 또한 같은 분산 스페어링이라도 [3]에서 제시한 방법보다는 패리티 유닛에 가깝도록 배치함으로써 성능 개선에 이용될 수 있음을 확인하였다.

향후의 작업으로 본 논문에서 제시한 분산 스페어링 방법을 수학적으로 체계화하여 분석적 모형을 만들고 다른 여러 가지 작업 부하를 사용한 시뮬레이션 결과와 비교해 보는 방법을 연구 중에 있다.

참 고 문 헌

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

[4] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for Redundant Arrays of Inexpensive Disks," Proceeding of the ACM SIGMOD International Conference on Management of Data, pp.109-116, 1988.

[5] G. A. Alvarez, W. A. Burkhard, L. J. Stockmeyer, and F. Cristian, "Declustered Disk Array Architecture with Optimal and Neal-optimal Parallelism," The 25th Proceedings of International Symposium on Computer Architecture, pp.109-120, 1998.

[6] E. J. Schwabe and I. M. Sutherland, "Improved Parity Declustering Layouts for Disk Arrays," Proceeding of Symposium on Parallel Algorithms and Architectures, pp.76-84, 1994.

[7] M. Holland and G. A. Gibson, "Parity Declustering for Continuous Operation in Redundant Disk Arrays," Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, pp.23-35, 1992.

[8] M. Holland, G. Gibson, and D. Siewiorek, "Fast, On-Line Failure Recovery in Redundant Disk Arrays," Proceedings of the 23rd International Symposium on Fault Tolerant Computing, pp.422-431, 1993.

[9] J. Menon and D. Mattson, "Comparison of Sparing Alternatives for Disk Arrays," The 19th International Symposium on Computer Architecture, pp.318-329, 1992.

[10] R. R. Muntz and J. C. S. Lui, "Performance Analysis of Disk Arrays under Failure," Proceedings of the 16th Conference on VLDB, pp.162-173, 1990.

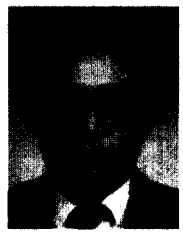
[11] A. Thomasian and J. Menon, "RAID5 Performance with Distributed Sparing," IEEE Trans. on Parallel and Distributed Systems, Vol.8, No.6, pp.640-657, June, 1997.

[12] G. R. Ganger, B. L. Worthington and Y. N. Patt, "The DiskSim Simulation Environment Version 1.0

Reference Manual," Technical Report CSE-TR-358-98, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1998.

[13] G. R. Ganger, "System-Oriented Evaluation of I/O Subsystem Performance," Technical Report CSE-TR-243-95, Department of EECS, University of Michigan, Ann Harbor, June, 1995.

[14] B. L. Worthington, G. R. Ganger and Y. N. Patt, "Scheduling Algorithms for Modern Disk Drives," ACM SIGMETRICS Performance Evaluation Review, Vol.22, No.1, pp.241-251, May 1994.



장 태 무

e-mail : jtm@cakra.dongguk.ac.kr

1977년 서울대학교 전자공학과 졸업(학사)

1979년 한국과학기술원 전산학과 졸업(이학석사)

1995년 서울대학교 컴퓨터공학과 졸업(공학박사)

1979년~1981년 한국전자기술연구소 연구원

1998년 University of Southeastern Lousiana 교환 교수

1981년~현재 동국대학교 컴퓨터·멀티미디어공학과 교수

관심분야 : 분산 및 병렬 처리, 컴퓨터구조, 입출력시스템 등