41

Redundant Array of Inexpensive Disk (RAID)

우리가 디스크를 사용할 때 때때로 더 빨랐으면 하는 바램이 있다. I/O 작업은 느리기 때문에 전체 시스템의 병목이 되기도 한다. 우리가 디스크를 사용할 때 용량도 더 많았으면 한다. 점점 더 많은 데이터가 온라인으로 접근되도록 저장되고, 따라서 우리의 디스크는 점점 더 가득 차게 된다. 우리가 디스크를 사용할 때 때때로 디스크가 더 안정적으로 동작하기 바란다. 디스크가 고장날 경우, 백업이 되어 있지 않다면 소중한 데이터를 잃어버리고 만다.

핵심 질문: 대용량이면서 빠르고 신뢰할 수 있는 디스크를 어떻게 만들까
어떻게 하면 대용량이면서 고속의 신뢰할 수 있는 저장 시스템을 만들까? 핵심 기술들은 무엇인가? 서로 다른 접근법들의 절충안은 무엇인가?

이번 장에서는 **Redundant Array of Inexpensive Disk** 또는 **RAID**라고 더 잘 알려진 기술을 소개한다 [PGK88]. 이 기술은 여러 개의 디스크를 조화롭게 사용하여 고속이면서 대용량의 신뢰할 수 있는 디스크 시스템을 만든다. 1980년대 말쯤 UC 버클리(David Patterson과 Randy Katz, 그 당시에는 학생이었던 Garth Gibson이 주도한)에서 처음 소개되었다. 그 즈음에 여러 다른 연구진들은 더 좋은 저장 시스템을 만들기 위해 여러 개의 디스크를 활용하는 기본 개념들을 앞다투어 내놓았다 [BG88; Kim86; Kur88; PB86; SG86].

외면적으로 RAID는 하나의 디스크처럼 보인다. 읽거나 쓸 수 있는 블럭의 그룹으로 보인다. 안을 들여다보면 RAID는 여러 개의 디스크와 메모리(휘발성과 비휘발성을 모두 포함), 시스템을 관리하기 위한 하나 또는 그 이상의 프로세서로 이루어진 복잡한 기계이다. RAID의 하드웨어는 컴퓨터 시스템과 매우 유사하며 디스크의 그룹을 관리하기 위한 전용 시스템이다.

RAID는 단일 디스크에 비해 여러 장점들을 제공한다. 하나의 장점은 *성능*이다. 디스크 여러 개를 병렬적으로 사용하면 I/O 시간이 크게 개선된다. 또 다른 장점은 *용량*이다. 데이터의 양이 많아지면 더 많은 디스크 공간이 필요하다. 마지막으로 RAID는 *신뢰성*을 높일 수 있다. 데이터를 여러 디스크에 분산하여 저장하면(RAID 기술을 사용하지 않고) 디스크 중에 하나가 고장나도 데이터를 잃어버릴 수 있다. **데이터 중복**

팁: 투명성은 장비 설치를 가능하게 한다

시스템에 새로운 기능을 추가하려고 한다면 **투명하게(transparently)** 적용 가능한지, 즉 시스템의 나머지 부분을 수정하지 않고 추가될 수 있는지 항상 고려해야 한다. 기존의 소프트웨어를 완전히 다시 작성해야 한다면(또는 하드웨어 사양이 크게 변경되어야 한다면) 아이디어의 영향력이 줄어들게 된다. RAID는 이러한 완벽한 사례이며, 이 투명성이 RAID의 성공에 분명한 기여를 하였다. 관리자들은 SCSI 기반의 RAID 저장 장치를 SCSI 디스크 대신에 사용할 수 있었고 나머지 시스템(호스트 컴퓨터, 운영체제 등)은 이것을 활용하기 위해서 조금도 바꾸지 않아도 되었다. 이 장비 **설치(deployment)**에 관한 문제를 해결함으로써 RAID는 시작부터 더 성공적이었다.

기술(redundancy)을 사용함으로써 RAID는 디스크 한 개의 고장은 감내할 수 있으며, 고장이 전혀 없던 것처럼 동작할 수 있다.

놀랍게도, RAID는 자신을 활용하는 시스템에게 이러한 장점들을 **투명하게(transparently)** 제공한다. 호스트 시스템은 RAID를 그저 거대한 디스크로 인식한다. 물론 투명성이 갖는 아름다움은 한 줄의 소프트웨어 변경없이 디스크를 RAID로 바꿀 수 있다는 것이다. 운영체제와 클라이언트 응용 프로그램은 변경 없이 계속 동작할 수 있다. 이와 같이 투명성은 RAID의 **확산력(deployability)**을 크게 개선시켰다. 사용자와 관리자가 소프트웨어 호환성을 전혀 걱정하지 않고 RAID를 사용할 수 있게 하기 때문이다.

이제 RAID의 몇 가지 중요한 측면을 살펴보자. 먼저 인터페이스와 결함 모델을 설명한다. 그 후에 용량과 신뢰성, 그리고 성능이라는 세 개의 중요한 축을 중심으로 RAID를 평가하는 방법을 다루도록 하겠다. 그리고 RAID의 설계와 구현과 관련된 중요한 다른 쟁점을 다루겠다.

41.1 인터페이스와 RAID의 내부

상위에 존재하는 파일 시스템에게 RAID는 크고 (희망 사항) 빠르고, (희망 사항) 신뢰할 수 있는 디스크로 보인다. 단일 디스크가 그러한 것처럼 RAID도 선형적인 블럭들의 배열로 보이며 파일 시스템(또는 다른 클라이언트)이 각 블럭을 읽거나 쓸 수 있다.

파일 시스템이 RAID에 *논리적* I/O를 요청하면 RAID는 내부에서 어떤 디스크(또는 디스크들)를 접근해야 요청을 완료할 수 있는지 계산한 후에 하나 또는 그 이상의 *물리적* I/O를 발생시킨다. 물리적 I/O에 대한 정확한 특성은 RAID 레벨에 따라 다르며 그 내용은 잠시 후에 다루게 될 것이다. 하지만 간단한 예로 각 블럭에 대해서 서로 다른 디스크에 2개의 복사본을 유지하는 RAID를 생각해 보자. 이와 같은 **미러링** 기반의 RAID 시스템에서 블럭에 쓰는 경우 하나의 논리적 I/O에 대해서 두 개의 물리적 I/O를 실행해야 한다.

RAID 시스템은 보통 별도의 하드웨어 박스 형태로 되어 있으며 호스트와 SCSI나 SATA와 같은 표준 인터페이스로 연결된다. 내부적으로 RAID는 꽤 복잡한데, RAID의 작업을 지시하는 펌웨어를 실행하는 마이크로 컨트롤러 그리고 블럭을 읽고 쓸 때

버퍼로 사용할 DRAM과 같은 휘발성 메모리가 있으며, 어떤 경우에는 쓰기를 안전하게 버퍼링하기 위한 비휘발성 메모리뿐 아니라 패리티 계산(몇몇 RAID 레벨에서는 패리티가 유용하다는 것을 앞으로 보게 될 것이다)을 위한 전용 논리 회로를 갖고 있기도 하다. 상위 레벨의 관점에서 RAID는 특수한 컴퓨터 시스템이다. 프로세서와 메모리 그리고 디스크를 갖고 있다. 단, 차이가 있다면 응용 프로그램을 실행하는 대신 RAID를 동작시키기 위해 설계된 전용 소프트웨어를 실행한다는 것이다.

41.2 결함 모델

RAID를 이해하고 다른 접근법들과 비교하기 위해서는 결함 모델(fault model)이 있어야 한다. RAID는 특정 종류의 결함을 파악하고 이를 복구하도록 설계되어 있다. 그렇기 때문에 설계시 어떤 종류의 결함에 대비를 해야 하는지를 파악하는 것이 매우 중요하다.

첫 번째 모델은 간단한 것으로 **고장 시 멈춤(fail-stop)** 결함 모델 [Sch84]이라고 불린다. 이 모델에서 디스크는 "정상 작동"이거나 "멈춤" 둘중 하나의 상태로 있다고 가정한다[1]. 동작 중인 디스크에는 모든 블럭을 읽거나 쓸 수가 있다. 반대로 멈충 상태의 디스크는 완전히 사용 불가능하다고 간주한다.

고장 시 멈춤 모델의 치명적인 단점은 결함이 발견되는 경우에 대한 가정이다. 이 모델에서는 디스크가 고장나면 쉽게 파악할 수 있다고 가정한다. 예를 들어, RAID 컨트롤러 하드웨어(또는 소프트웨어)는 디스크 고장 시 즉시 알 수 있다고 가정하고자 한다. 실제로 인생은 그리 간단치 않다.

일단 지금은 디스크 섹터내용이 망가지는 것과 같은 "조용한" 고장들에 대해서는 고려하지 않는다. 디스크에서 한 블럭만 접근할 수 없는 고장에 대해서도 고려하지 않는다(이는 잠재된 섹터 에러라고 불리기도 한다). 이런 복잡한(그리고 불행하게도 좀 더 현실적인) 디스크 결함들에 대해서는 차후에 다루도록 한다.

41.3 RAID의 평가 방법

이제 곧 보게 되겠지만 RAID를 구성하는 몇 가지 방법이 있다. 각 구성 방법은 서로 다른 특성을 갖고 있기 때문에 여러 방법 간의 장점과 단점을 이해하기 위해서는 평가 수단이 필요하다.

구체적으로 세 개의 축을 중심으로 각 RAID 설계를 평가할 것이다. 첫 번째 축은 **용량**이다. B개의 블럭을 가지는 N개의 디스크가 주어졌을 때 RAID의 클라이언트가 사용할 수 있는 유효 용량은 얼마나 되는가? 중복 저장이 없는 경우라면 대답은 당연히 $N \cdot B$이다. 반대로 각 블럭에 대해 두 개의 복사본을 갖는 경우라면(미러링이라고 불림) 유효한 공간은 $(N \cdot B)/2$가 될 것이다. 다른 기법(예, 패리티 기반 기법)은 그 사이의 값을 가지게 된다.

1) 역자 주: 이 두 가지 상태말고 다른 상태는 뭐가 있을까? 비정상 작동이 있다. 작동 중이지만 잘못된 데이터를 읽거나 쓰는 경우다. 최악이다.

두 번째 평가의 축은 **신뢰성**이다. 평가 대상의 설계 방법은 몇 개의 디스크 결함을 감내할 수 있는가? 우리의 결함 모델에 따라 하나의 디스크만 고장 날 수 있다고 가정한다. 이후의 장(즉, 데이터 무결성에 관한)에서는 좀 더 복잡한 결함 모드를 처리하는 방법을 알아볼 것이다.

마지막으로 세 번째 축은 **성능**이다. 성능은 디스크 배열이 처리해야 할 워크로드에 따라 크게 달라지기 때문에 평가하기가 만만치 않다. 그러므로 성능을 평가하기 이전에 고려해야 할 일반적인 워크로드를 제시할 것이다.

이제 세 개의 중요한 RAID 설계를 살펴볼 것이다. RAID 레벨 0(스트라이핑), RAID 레벨 1(미러링) 그리고 RAID 레벨 4/5(패리티에 기반을 둔 중복 저장)를 다루도록 하겠다. 각 설계에 붙는 "레벨"의 어원은 Berkeley 대학의 Patterson, Gibson과 Katz의 선구적인 연구이다 [PGK88].

41.4 RAID 레벨 0: 스트라이핑

첫 번째 RAID 레벨은 사실 중복 저장을 하지 않기 때문에 RAID 레벨이 결코 아니다. 하지만, RAID 레벨 0 또는 **스트라이핑**(**striping**)이라고 더 잘 알려진 이 방식은 성능과 용량에 대한 훌륭한 상한 기준을 나타내기 때문에 이해하고 넘어갈 만하다.

가장 간단한 스트라이핑은 다음의 그림과 같이 블럭들을 여러 디스크에 걸쳐서 줄을 긋는 것처럼 저장한다(여기서는 네 개의 디스크 배열을 사용한다고 가정한다).

그림 41.1을 보면 기본 개념을 이해할 수 있다. 디스크 배열의 블럭들을 라운드 로빈 방식으로 디스크를 가로질러 펼치는 것이다. 이 접근법은 배열의 연속적인 청크(예를 들어, 크기가 큰 순차 읽기에서처럼)에 대해 요청을 받았을 때 병렬성을 가장 잘 활용할 수 있도록 설계되었다. 같은 행에 있는 블럭들을 **스트라이프**(**stripe**)라고 부른다. 따라서 블럭 0, 1, 2, 그리고 3은 같은 스트라이프에 존재한다.

디스크 0	**디스크 1**	**디스크 2**	**디스크 3**
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

〈그림 41.1〉 **RAID 0: 단순한 스트라이핑**

이 예에서는 단순하게 하나의 블럭(4 KB 단위의 크기)만이 다음 디스크로 넘어가지 전에 현재 디스크에 배치된다고 가정한다. 하지만 꼭 이렇게 배치할 필요는 없다. 예를 들면 그림 41.2처럼 블럭을 배치할 수도 있다.

이 예제에서 각 디스크 마다 두 개의 4 KB 블럭을 배치한 후에 다음 디스크로 넘어갔다. 그러므로 이 RAID 배열의 **청크 크기**는 8 KB이고, 스트라이프는 4개의 청크 또는

디스크 0	디스크 1	디스크 2	디스크 3	
0	2	4	6	청크 크기:
1	3	5	7	2 블럭
8	10	12	14	
9	11	13	15	

〈그림 41.2〉 **더 큰 청크 크기 단위로 스트라이핑하기**

여담: RAID 매핑 문제

RAID의 용량과 신뢰성 그리고 성능 특성을 공부하기 전에 먼저 여담으로 **매핑 문제**를 다루고자 한다. 모든 RAID 배열에서 일어나는 문제로 간단하게는 다음과 같다. 논리 블럭에 대한 읽기와 쓰기를 할 때에 RAID는 어떻게 해당 물리 디스크와 오프셋을 알아서 정확하게 그 위치에 접근할 수 있을까?

간단한 RAID 레벨의 경우에는 물리 위치와 논리 블럭을 정확하게 연결하기 위해 대단히 정교한 기법이 필요하지는 않다. 첫 번째 스트라이핑 예제(청크 크기 = 1 블럭 = 4 KB)로 생각해 보자. 논리 블럭 주소가 A라고 할 때에 RAID는 두 개의 간단한 수식으로 원하는 디스크와 오프셋을 계산해 낼 수가 있다.

```
Disk   = A % number_of_disks
Offset = A / number_of_disks
```

이 계산은 정수 계산(예, 4/3 는 1 이지 $1.333\dot{3}$이 아니다)이라는 것을 유의해야 한다. 간단한 예제에서 이 수식이 어떻게 계산되는지 살펴보자. 첫 번째 RAID 경우에서 14번 블럭에 대한 요청을 받았다고 하자. 네 개의 디스크가 있다고 할 때에 우리가 찾는 디스크는 $(14 \% 4 = 2)$로 디스크 2번이 된다. 정확한 블럭은 $14 / 4 = 3$ 으로 계산되어 3번 블럭이 된다. 그러므로 블럭 14번은 세 번째 디스크(0부터 시작하여 디스크 2번)의 4번째 블럭(0부터 시작하여 블럭 3번)으로 계산되며 실제 그곳에 있는 것을 확인할 수 있다.

청크 크기가 바뀌었을 때 이 수식들이 어떻게 바뀌어야 할지 한 번 생각해 보라. 해 보라! 그렇게 어렵지 않다.

32 KB 크기의 데이터를 저장한다.

청크 크기

청크 크기는 RAID의 성능에 큰 영향을 준다. 작은 청크 크기가 의미하는 것은 많은 파일들이 여러 디스크에 걸쳐서 스트라이프된다는 말이며 그 결과로 하나의 파일을 읽고 쓰는 데 병렬성이 증가하게 된다. 하지만 블럭의 위치를 여러 디스크에서 찾아야 하므로

위치 찾기 시간이 늘어난다. 왜냐하면 요청 처리 시간은 여러 디스크에 걸친 요청들 중에 가장 오래걸린 찾기 시간에 의해 결정되기 때문이다.

반면 큰 청크 크기의 경우에는 파일 내의 병렬성은 줄어들고 따라서 높은 처리 성능을 얻으려면 여러 요청을 병행하게 실행해야 한다. 하지만 큰 청크 크기의 경우 위치 찾기 시간은 줄어든다. 예를 들어, 파일이 작아서 단일 청크에 저장이 된다면, 이 파일을 읽는데 요구되는 위치 찾기 시간은 한 디스크에서 그 위치를 찾는 시간과 동일하게 된다.

"최적"의 청크 크기를 정하는 것은 디스크 시스템이 처리할 워크로드에 대한 심도 있는 이해를 필요로한다. 그리 단순한 일이 아니다 [CL95]. 이후의 논의에서는 청크 크기는 한 블럭의 크기(4 KB)로 가정한다. 대부분의 RAID에서는 큰 청크(예, 64 KB)를 사용하지만 이제부터 다룰 될 주제들에서는 청크의 크기가 문제되지 않는다. 논의를 간단하게 하기 위해 청크의 크기를 한 블럭으로 가정한다.

RAID-0 분석으로 돌아가서

이제 스트라이핑 방식의 용량, 신뢰성 그리고 성능을 평가해 보자. 용량 측면에서는 완벽하다. N 개의 디스크에서 스트라이핑은 $N \cdot B$ 개의 디스크 용량만큼 유효 용량을 갖는다. 신뢰성 측면에서는 스트라이핑은 역시 완벽하지만 나쁜 의미로 완벽하다. 어느 디스크라도 고장나면 전체 데이터가 손실된다. 마지막으로 성능은 훌륭하다. 병렬로 사용자의 I/O 요청을 처리할 수 있기 때문에 모든 디스크가 활용된다.

RAID의 성능 평가하기

RAID의 성능을 분석하기 위해 두 가지 다른 성능 척도를 고려해야 한다. 하나는 *단일 요청의 지연 시간*이다. 하나의 I/O를 처리하는 데 걸리는 RAID의 지연 시간에 대한 이해는 한 번의 논리적 I/O 동작을 처리할 때의 병렬성 정도를 파악하는 데 도움이 된다. 두 번째 척도는 RAID의 *정상 상태*(*steady state*)에서의 *처리성능*(*throughput*)이다. 병행 요청의 전체적인 대역폭을 말한다. 고성능을 요하는 환경에서 RAID가 주로 사용되기 때문에 안정(stable) 상태에서의 대역폭은 매우 중요하며 우리 분석의 주요 초점이다.

처리성능을 좀 더 자세하게 이해하기 위해서 관심을 가질만한 워크로드를 먼저 생각해 보자. 이 논의를 위해 **순차**와 **랜덤** 워크로드의 두 가지 유형이 있다고 있다고 가정한다. 순차 워크로드에서는 연속된 큰 청크의 형태로 RAID에 요청된다. 예를 들면, 1 MB의 데이터를 접근하는 요청(또는 연속적인 요청들)이 블럭 B에서 시작하여 B+1 MB에서 끝난다면 순차적이라고 본다. 여러 환경에서 순차 워크로드는 (큰 파일에서 키워드를 찾는 경우처럼) 흔히 발생하기 때문에 중요하게 생각된다.

랜덤 워크로드는 크기가 작은 요청이며 또한 디스크의 여러 불특정 위치를 접근한다고 가정한다. 예를 들면, 논리 주소 10에서 첫 번째 4 KB를 접근하고 다음으로 논리 주소 550,000 그리고 20,100 등의 위치로 이동하여 접근하는 경우다. 데이터베이스 관리 시스템의 트랜젝션과 같은 일부 중요한 워크로드는 이와 같은 접근 패턴을 보이기 때문에 역시 중요한 워크로드로 여겨진다.

물론 실제 워크로드는 그렇게 단순하지 않으며 순차와 랜덤의 성질이 섞여 있기 때문에 특성도 어중간하게 나타난다. 간단하게 하기 위해서 이 두 가능성만 고려하도록 한다.

순차와 랜덤 워크로드는 디스크로부터 큰 성능 차이를 만들어 낸다. 순차 접근의 경우 디스크가 가장 효율적으로 동작한다. 탐색과 회전 지연이 짧아 대부분의 시간을 데이터 전송에 사용한다. 랜덤 접근의 경우는 그 반대가 된다. 대부분의 시간을 탐색과 회전을 기다리는 데 사용되고 상대적으로 적은 시간이 데이터 전송에 사용된다. 분석에서 그 차이를 나타내기 위해서 디스크가 순차 워크로드의 경우 S MB/s, 랜덤 워크로드에서는 R MB/s의 속도로 전송한다고 가정하자. 일반적으로는 S가 R보다 매우 크다(즉, $S \gg R$).

차이를 이해했다는 것을 확인하기 위해서 간단한 연습문제를 풀어 보자. 구체적으로 다음의 디스크 특성에 따라 S와 R을 계산해 보자. 평균적으로 10 MB의 크기를 순차적으로 전송하고 평균 10 KB의 크기를 임의의 방식으로 전송한다고 가정하자. 또한 다음과 같이 디스크의 특성을 가정하자.

평균 탐색 시간	7 msec
평균 회전 시간	3 msec
디스크 전송 속도	50 MB/s

S를 계산하기 위해서 10 MB를 전송하는 데 어떻게 시간이 사용되는지를 먼저 보자. 우선, 탐색하는 데 7 msec를 사용하고, 회전하는 데 3 msec를 사용한다. 마침내 전송이 시작되며 10 MB @ 50 MB/s는 5분의 1초 또는 200 msec가 전송에 소요된다. 그러므로 매 10 MB마다 전송을 완료하는 데 210 msec가 걸린다. S는 다음과 같이 계산된다.

$$S = \frac{\text{데이터 총 크기}}{\text{접근 시간}} = \frac{10 \text{ MB}}{210 \text{ msec}} = 47.62 \text{ MB/s}$$

보는 바와 같이 데이터를 전송하는 데 많은 시간을 쓰기 때문에 S의 값은 디스크의 최대 대역폭에 근접한다(탐색과 지연 시간 비용은 전체 시간이 길어짐에 따라 영향이 줄어든다).

R도 비슷하게 계산할 수 있다. 탐색과 회전 지연은 동일하다. 전송하는 데 걸리는 시간을 계산하면 10 KB @ 50 MB/s 또는 0.195 ms가 걸린다.

$$R = \frac{\text{데이터 총 크기}}{\text{접근 시간}} = \frac{10 \text{ KB}}{10.195 \text{ msec}} = 0.981 \text{ MB/s}$$

보는 바와 같이 R은 1 MB 이하이며 S/R은 거의 50이다.

RAID-0 분석으로 또 다시 돌아가서

이제 단순 스트라이핑의 성능을 평가해 보자. 앞서 언급했듯이 전반적으로 괜찮다. 예를 들어, 지연 시간 측면에서 한 블럭에 대한 요청의 지연 시간은 하나의 디스크에 대한

요청의 지연 시간과 거의 동일하다. 결국 RAID-0은 받은 요청을 디스크 중 하나에게 전달하는 것뿐이기 때문이다.

정상 상태에서의 대역폭 측면에서 시스템의 최대 대역폭을 기대할 수 있다. 그러므로 처리성능은 N(디스크의 수) 곱하기 S(디스크 하나의 순차 접근 대역폭)와 같다. 많은 랜덤 I/O의 경우도 모든 디스크를 다 사용할 수 있기 때문에 $N \cdot R$ MB/s를 얻을 수 있다. 곧 살펴보겠지만, 이 수치들은 가장 간단하게 계산될 수 있으며, 다른 RAID 레벨들과 비교할 때 상한치로 사용될 것이다.

41.5 RAID 레벨 1: 미러링

스트라이핑 이후의 첫 번째 RAID 레벨은 RAID 레벨 1 또는 미러링으로 알려져 있다. 미러링을 사용하는 시스템에서는 각 블럭에 대해서 하나 이상의 사본을 둔다. 각 사본은 당연히 서로 다른 디스크에 저장되어야 한다. 그렇게 함으로써 디스크 고장에 대처할 수 있게 된다.

일반적인 미러링 시스템에서 RAID는 각 논리 블럭에 대해 두 개의 물리적 사본을 둔다고 가정한다. 여기에 그 예제가 있다.

디스크 0	디스크 1	디스크 2	디스크 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

〈그림 41.3〉 **단순한 RAID 1: 미러링**

이 예제에서는 디스크 0과 디스크 1이 동일한 값을 갖고 디스크 2와 디스크 3이 동일한 값을 갖고 있다. 데이터는 미러링된 쌍들에 대해서 스트라이핑을 적용하였다. 사실 디스크에 사본을 배치하는 방법이 무수히 많다는 것은 이미 알고 있을 것이다. 위에서 보인 방식은 일반적인 방법으로 **RAID-10** 또는 **RAID 1+0** 이라고 불린다. 그 이유는 미러링(RAID-1)을 적용한 것을 스트라이핑(RAID-0)하기 때문이다. 또 다른 구성은 **RAID-01** 또는 **RAID 0+1**로서 배열을 RAID-0으로 스트라이핑한 후에 이 배열을 RAID-1로 미러링하는 방식이다. 여기에서는 앞에서 보인 미러링 방식만을 가정한다.

미러링된 배열에서 블럭을 읽을 때 RAID는 원본을 읽을 건지 사본을 읽을건지 선택할 수 있다. 예를 들면, 논리 블럭 5번을 RAID에서 읽도록 요청하면 디스크 2에서 또는 디스크 3에서 읽을 수 있다. 반면에 블럭을 쓰는 경우에는 그렇게 선택할 수 없다. RAID는 신뢰성을 유지하기 위해서 두 벌의 데이터를 모두 갱신해야 한다. 단, 이러한

쓰기 요청은 병렬적으로 처리될 수 있다는 사실을 유념해야 한다. 예를 들어 논리 블럭 5번에 쓰는 요청은 디스크 2와 3에서 동시에 처리될 수 있다.

RAID-1 분석

RAID-1을 평가해 보자. 용량 측면에서는 RAID-1은 비용이 많이 든다. 미러링 레벨이 2라면 최대 사용 가능한 용량의 반만 사용할 수 있다. 그러므로 B개의 블럭을 가진 N개의 디스크 배열의 경우 유효 용량은 $(N \cdot B)/2$가 된다.

신뢰성 측면에서는 RAID-1은 괜찮은 편이다. 디스크 중 어느 것에서 고장이 발생하여도 감내할 수 있다. 실제로 운이 따라준다면 RAID-1은 더 많은 고장을 감내할 수 있다는 것을 알 수 있을 것이다. 앞의 그림에서 디스크 0과 디스크 2에서 모두 고장이 발생했다고 생각해 보자. 그런 경우라도 데이터를 잃어버리지 않는다! 일반적으로 미러링 시스템(미러링 레벨 2)의 경우 한 개의 디스크 고장은 확실히 감내할 수 있고 어떤 디스크가 고장나느냐에 따라 $N/2$개의 결함까지 감내할 수 있다. 일반적으로 현실에서는 이렇게 운에 맞기고 싶진 않을 것이다. 그러므로 대부분의 사람들은 하나의 고장만을 처리할 수 있다고 가정한다.

마지막으로 성능을 분석해 보자. 단일 읽기 요청의 지연 시간은 단일 디스크에서 읽는 경우의 지연 시간과 동일하다. RAID-1이 하는 일은 두 벌의 사본 중에 하나의 디스크로 요청을 전달하는 것이기 때문이다. 쓰기의 경우는 약간 다르다. 쓰기 요청이 완료하기 위해서는 두 개의 디스크로 전달된 쓰기가 모두 종료되어야 한다. 이 두 개의 쓰기는 병렬적으로 이루어지기 때문에 하나의 쓰기에 소요되는 시간과 거의 동일할 것이다. 하지만 RAID-1에 요청된 쓰기는 물리적으로 두개의 쓰기연산이 종료될 때까지 대기해야 하므로, 그 지연시간은 두 개의 요청 중 최악의 탐색과 회전 지연 시간에 의해 결정된다. 평균적으로 하나의 디스크에 쓰는 시간보다는 조금 더 길다.

순차 워크로드로 정상 상태의 처리성능에 대한 분석을 시작해 보자. 디스크에 순차적으로 쓰기를 할 때에 각 논리 쓰기는 항상 두 번의 물리 쓰기가 필요하다. 예를 들어 논리 블럭 0번(앞의 그림에서와 같이)에 쓴다고 해 보자. RAID는 내부에서 디스크 0번과 디스크 1번에 둘 다 기록해야 한다. 그러므로 미러링된 배열에 순차 쓰기의 경우 $(\frac{N}{2} \cdot S)$ 또는 최대 대역폭의 절반의 대역폭을 얻을 수 있다.

불행하게도 순차 읽기 상황에서도 동일한 성능을 얻는다. 순차 읽기의 경우에 두 벌의 사본 중 하나만 읽으면 되기 때문에 좀 더 잘 할 수 있을 것이라고 생각할 수도 있다. 그다지 이득을 얻을 수 없는 이유를 다음의 예를 통해서 살펴보자. 읽어야 할 블럭이 0, 1, 2, 3, 4, 5, 6, 그리고 7번이라고 해 보자. 0번은 디스크 0번에서 읽고, 1번 블럭은 디스크 2번에서, 그리고 2번 블럭은 디스크 1번에서, 3번 블럭은 디스크 3번에서 읽도록 명령을 내렸다고 하자. 계속해서 4, 5, 6, 그리고 7번을 읽는 데 각각 디스크 0, 2, 1, 3번에서 읽도록 명령을 내린다. 얕은 생각에 모든 디스크를 다 사용하기 때문에 배열의 전체 대역폭을 얻을 수 있지 않겠냐고 생각할 수 있다.

그렇지 않다는 것을 보이기 위해 하나의 디스크(0번이라고 하자)가 받는 요청을 살펴보자. 먼저 블럭 0번에 대한 요청을 받는다 그리고 두 블럭 떨어진 위치의 블럭 4

여담: RAID의 일관성 있는 갱신 문제

RAID-1의 분석 전에 여러 디스크로 이루어진 RAID 시스템에서 생길 수 있는 문제인 **일관성 갱신 문제(consistent-update problem)**를 살펴보자 [DAA05]. 이 문제는 쓰기 요청으로 인한 하나의 논리 작업을 위해 여러 개의 디스크를 갱신해야 하는 모든 RAID에서 일어날 수가 있다. 미러링된 디스크 배열을 가정하고 논의해 보자.

RAID에 쓰기가 요청되고 RAID는 디스크 0번과 1번에 기록한다고 가정하자. RAID는 디스크 0에 쓰기를 요청하고 나서 디스크 1번에 요청하기 직전에 전력 손실(또는 시스템 멈춤)이 발생했다. 이와 같은 불행한 경우에 디스크 0의 요청은 완료가 되었다고 가정하자(하지만 디스크 1번의 요청은 요청조차 없었기 때문에, 갱신하지 않았다).

때에 맞지 않는 전력 손실의 결과로 두 벌의 블럭의 내용이 **불일치** 상태가 된다. 디스크 0번의 사본은 새 버전이고 디스크 1번의 사본은 예전 것이다. 우리가 원하는 것은 두 디스크에 대한 상태의 갱신이 **원자적(atomically)**으로 수행되는 것이다. 즉, 두 디스크의 값이 모두 새로운 값으로 변경이 되거나 아무것도 변경이 되지 않기를 원한다.

이 문제를 해결하는 일반적인 방법은 **write-ahead 로그**의 일종을 사용하여 RAID가 수행하려는 동작(즉, 특정 값으로 두 디스크 갱신)을 먼저 기록한 후에 실제 동작을 수행하는 것이다. 이러한 접근법을 사용하면 시스템이 크래시되더라도 제대로 된 결과를 얻을 수 있도록 보장해 준다. RAID에 대기 중인 모든 트랜젝션을 재실행하는 복구 절차를 수행하여 어떤 두 벌의 사본(RAID-1의 경우)도 일치한다는 것을 보장할 수 없다.

마지막으로 첨언하면 모든 쓰기에 대해 디스크에 로깅하는 것은 엄두를 못 낼 만큼 비싸기 때문에 대부분의 RAID 하드웨어는 작은 크기의 비휘발성 RAM(예, 건전지에 연결된)에 로그를 남긴다. 그러므로 디스크에 로깅하는 비싼 방식을 사용하지 않고도 일관성 유지 갱신을 수행할 수 있다.

에 대한 요청을 받는다(2번 블럭을 건너뛰고) 사실 각 디스크는 하나씩 건너 뛴 블럭에 대한 요청을 받는다. 건너 뛴 블럭들을 블럭 위를 회전하는 동안은 사용자에게 유용한 대역폭을 제공하지 않는다. 그러므로 각 디스크는 자신이 낼 수 있는 최대 대역폭의 반만 쓸 수가 있다. 그래서 순차 읽기의 대역폭도 $(\frac{N}{2} \cdot S)$ MB/s가 된다.

램덤 읽기가 미러링된 RAID에서는 최고의 워크로드이다. 이 경우에는 모든 디스크에 읽기를 다 요청할 수 있기 때문에 얻을 수 있는 최대의 대역폭을 얻을 수 있다. 그러므로 랜덤 읽기의 경우 RAID-1은 $N \cdot R$ MB/s를 보인다.

마지막으로 랜덤 쓰기는 예상한 것과 같이 $\frac{N}{2} \cdot R$ MB/s를 갖는다. 각 논리 쓰기는 두 번의 물리 쓰기로 바뀌어야 하기 때문에 모든 디스크가 사용이 되더라도 사용자는 모든 대역폭의 반만 얻을 수 있다. 논리 블럭 x에 대해 두 개의 다른 물리 디스크로 나뉘어서 쓰기가 진행된다 하더라도 작은 크기의 쓰기 요청에 대한 대역폭은 스트라이핑의 경우의 절반만 얻을 수 있다. 이제 곧 전체 대역폭의 반도 사실 상당히 좋은 것이라는 것을 알게 될 것이다!

41.6 RAID 레벨 4: 패리티를 이용한 공간 절약

이제 패리티라고 알려진 중복성을 추가하는 다른 방법을 설명한다. 패리티 기반의 접근 방법은 저장 공간을 더 적게 사용하려고 하여 미러링 기반 시스템이 지불하는 엄청난 공간 낭비를 극복하려고 시도한다. 그 대신 성능이라는 비용을 지불해야 한다.

디스크 0	디스크 1	디스크 2	디스크 3	디스크 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

〈그림 41.4〉 **패리티가 있는 RAID-4**

여기의(그림 41.4) 디스크 5개로 이루어진 RAID-4 시스템을 예로 살펴보자. 각 데이터 스트라이프마다 해당 스트라이프에 대한 중복 정보를 담고 있는 **패리티** 블럭 하나를 추가한다. 블럭 P1은 블럭 4, 5, 6과 7번으로부터 계산된 중복 정보를 가지고 있다.

패리티를 계산하기 위해서 스트라이프에 속해 있는 블럭 중 하나의 블럭이 고장나더라도 견딜 수 있게 하는 수학 함수가 필요하다. 간단한 함수인 **XOR**이 우리가 원하는 목적에 잘 부합하는 것을 알 수 있다. 어떤 비트가 주어졌을 때 XOR은 1이 짝수 개일 때는 0을 반환하고 홀수 개의 1이 있으면 1을 반환한다. 예를 들면 다음과 같다.

C0	C1	C2	C3	P
0	0	1	1	XOR(0, 0, 1, 1) = 0
0	1	0	0	XOR(0, 1, 0, 0) = 1

첫째 줄(0, 0, 1, 1)에는 C2와 C3에 1이 있다. 그러므로 이 값들을 XOR하면 P로 0을 얻는다. 마찬가지로 두 번째 줄에는 1이 C1에만 있으므로 그 줄을 XOR을 하면 P는 1이 되어야 한다. 이것을 기억하는 간단한 방법은 어떤 줄이던 그 줄의 1의 수는 짝수(홀수가 아님)가 되어야 한다. 이것은 RAID의 패리티가 정확하게 동작하기 위해서 반드시 유지해야 하는 **불변량**이다.

위 예제에서 고장난 디스크로부터 데이터를 복구하기 위해 어떻게 사용할지 예상할 수 있을 것이다. C2 열이 깨졌다고 해 보자. 그 열에 있었던 값들을 알아보기 위해서 할 일은 단순하게 그 행의 모든 값(XOR의 결과로 얻은 패리티를 포함)을 읽은 후 올바른 값을 **다시 계산**하면 된다. 구체적으로 첫 번째 행의 C2 열의 값(1)이 깨졌다고 해 보자. 같은 행의 다른 값들(C0에서 0을, C1에서 0을, C3에서 1을 읽고 그리고 P 열에서 패리티로 0을 읽는다)을 읽어서 0, 0, 1 그리고 0의 값을 얻어낸다. 각 행의 값들을

XOR하면 해당 행에 1이 짝수 개가 되도록 한다는 것을 알기 때문에 사라진 데이터의 값은 1이 되어야 한다. 이게 바로 XOR기반의 패리티 기법에서 값을 복구하는 방법이다! 값을 복구하기 위해서 계산할 때 패리티 비트와 데이터 비트를 같이 XOR했다는 것을 유의해야 한다. 최초에 패리티를 계산할 때 모든 비트를 사용하여 XOR했던 것과 같은 계산이다.

이제 궁금증이 생겼을 것이다. 비트들을 모두 XOR해야 한다고 하는데, 각 디스크에 4 KB(또는 그 이상)의 블럭을 저장한다면 패리티를 계산하기 위해서 여러 개의 블럭들을 어떻게 XOR 해야 한다는 것인가? 이 또한 간단하게 할 수 있다. 데이터 블럭의 비트들에 대해서 비트 단위로 XOR을 하면 된다. 그리고 각 비트 연산으로 XOR한 각각의 결과를 패리티 블럭의 해당 비트의 위치에 저장하면 된다. 예를 들어 블럭이 4 비트 크기를 갖고 있다고 하면, (4 KB 블럭보다 한참 작기는 하지만) 다음과 같이 표현될 것이다.

블럭 0	**블럭 1**	**블럭 2**	**블럭 3**	**패리티**
00	10	11	10	11
10	01	00	01	10

그림에서 보는 것과 같이 패리티는 각 블럭의 각 비트들에 대해서 수행이 되었으며 결과는 패리티 블럭의 해당 위치에 나타나 있다.

RAID-4 분석

이제 RAID-4를 분석해 보자. 용량 측면에서 RAID-4를 살펴보면 패리티 정보의 저장을 위해 각 디스크 그룹에 속한 디스크 하나를 사용하기 때문에 $(N-1)\cdot B$의 저장 공간을 제공한다.

신뢰성 역시 쉽게 이해할 수 있다. RAID-4는 오직 하나만의 디스크 고장을 감내할 수 있다. 두 개 이상의 디스크가 고장나면 읽어 버린 데이터를 복원할 수 있는 방법이 없다.

마지막으로 성능이다. 먼저 정상 상태의 처리성능을 먼저 살펴보는 것으로 시작하자. 순차 읽기 성능의 경우 패리티 디스크를 제외한 모든 디스크를 활용할 수 있기 때문에 최대 유효한 대역폭은 $(N-1)\cdot S$ MB/s가 된다(쉬운 경우이다).

순차 쓰기 성능을 이해하려면 먼저 어떻게 동작하는지 이해해야 한다. 큰 청크의 데이터를 디스크에 쓰려고 할 때 RAID-4는 **스트라이프 전부 쓰기**(**full-stripe write**)라고 하는 간단한 최적화 방법을 수행할 수 있다. 예를 들어, 쓰기 요청의 일부로 블럭 0, 1, 2 그리고 3번이 RAID로 보내진 경우를 생각해 보자(그림 41.5).

이 경우에 RAID는 P0의 새로운 값을 계산하기 위해서 단순히 블럭 0, 1, 2 그리고 3번을 XOR하면 되고, 패리티 블럭을 포함한 모든 블럭을 다섯 개의 디스크에 병렬적으로 쓰면 된다(그림의 회색 부분). 그러므로 스트라이프 전부 쓰기가 RAID-4에서 쓰기 작업을 하는 가장 효율적인 방법이다.

디스크 0	디스크 1	디스크 2	디스크 3	디스크 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

〈그림 41.5〉 **RAID-4에서 스트라이프로 전부 쓰기**

스트라이프 전부 쓰기 방식을 이해했다면, RAID-4의 순차 쓰기 성능을 계산하는 것은 쉽다. 유효 대역폭은 마찬가지로 $(N-1)\cdot S$ MB/s가 된다. 패리티 디스크가 지속적으로 사용되고 있기는 하지만 클라이언트는 그 디스크로부터 성능의 이득을 얻을 수가 없다.

이제는 랜덤 읽기의 성능을 살펴보자. 앞의 그림에서 볼 수 있는 것과 같이 모든 데이터 디스크에서 랜덤 읽기로 한 블럭을 읽을 수는 있지만 패리티 디스크에서는 읽지 못한다. 그러므로 유효 성능은 $(N-1)\cdot R$ MB/s가 된다.

마지막까지 남겨둔 랜덤 쓰기의 경우에는 RAID-4에서 가장 흥미로운 상황을 만들어 낸다. 위의 예제에서 블럭 1번을 갱신하려는 경우를 생각해 보자. 아무 생각 없이 해당 블럭만 갱신할 수도 있겠지만 그렇게 하면 문제가 생긴다. 패리티 블럭 P0은 더 이상 해당 스트라이프의 정확한 패리티 값을 반영하지 못하게 된다. 이 예제에서 P0 역시 갱신되어야 한다. 어떻게 하면 정확하고 효율적으로 이 값을 갱신할 수 있을까?

두 가지 방법이 있다. 첫 번째는 **가산적 패리티**(**additive parity**)로 알려진 방법이며 다음과 같이 동작한다. 새로운 패리티 블럭의 값을 계산하기 위해 스트라이프 내의 다른 모든 데이터 블럭을 병렬적으로 읽고(예에서는 블럭 0, 2 그리고 3번) 새로운 블럭 1번과 함께 XOR을 한다. 쓰기를 완료하기 위하여 새 데이터와 새 패리티를 해당 디스크에 병렬적으로 쓴다.

이 기법의 문제는 디스크의 개수에 따라 계산의 양이 달라지기 때문에 더 큰 RAID의 경우에는 패리티 계산을 위해서는 많은 수의 읽기 연산이 필요하다. 그래서 **감산적 패리티**(**subtractive parity**) 방법이 나왔다.

다음과 같이 비트가 나열되어 있다고 해 보자(네 개의 데이터 비트와 하나의 패리티 비트).

C0	C1	C2	C3	P
0	0	1	1	XOR(0, 0, 1, 1) = 0

$C2$ 비트를 새로운 값인 $C2_{new}$로 갱신하고 싶은 경우를 생각해 보자. 감산적 방식은 세 단계로 동작한다. 먼저 $C2(C2_{old}=1)$와 패리티$(P_{old}=1)$의 옛날 값을 각각 읽어 들인다. 그리고 옛날 값과 새로운 값을 서로 비교하여 동일하다면(예, $C2_{new}=C2_{old}$)

패리티 값이 그대로 유지될 것이라는 것을 안다($P_{new} = P_{old}$). 그러나 만약 서로 다르다면, 이전의 패리티 비트를 현재 상태의 반대 값으로 뒤집어야 한다. 즉, $P_{old} = 1$이었다면 P_{new}는 0이 되고, $P_{old} = 0$이었다면 P_{new}는 1로 설정될 것이다. 이 복잡한 과정을 XOR로 깔끔하게 정리할 수 있다(이때 $\oplus$는 XOR 연산자이다).

$$P_{new} = (C_{old} \oplus C_{new}) \oplus P_{old} \tag{41.1}$$

비트가 아닌 블럭을 다루고 있으므로 블럭의 모든 비트들에 대해서 수행해야 한다(예, 각 블럭은 4096 bytes이며 byte당 8비트를 곱한 만큼 수행한다). 대부분의 경우 새로운 블럭은 예전 블럭과 다를 것이기 때문에 새로운 패리티 블럭 역시 다를 것이다.

이제 언제 가산적 패리티 계산을 사용하고 또 언제 감산적 방식을 사용하는지 알 수 있어야 한다. 가산적 패리티 방식이 감산적 방법보다 더 적은 수의 I/O 연산을 하려면 몇 개의 디스크가 필요한지 생각해 보라. 성능의 역전을 위한 디스크의 수는 얼마인가?

디스크 0	**디스크 1**	**디스크 2**	**디스크 3**	**디스크 4**
0	1	2	3	P0
*4	5	6	7	+P1
8	9	10	11	P2
12	*13	14	15	+P3

〈그림 41.6〉 **예제, 4와 13에 쓰기와 각각의 패리티 블럭들**

블럭 4번과 13번을 갱신하는 두 개의 작은 크기의 쓰기(그림에 *로 표시됨)가 RAID-4에 거의 동시에 요청되었다고 해 보자. 해당 데이터는 디스크 0과 1번에 있으며 읽기와 쓰기는 병렬적으로 일어날 수 있으니 좋은 상황이다. 문제는 패리티 디스크에 있다. 두 요청 모두 블럭 4와 13번의 패리티 블럭인 1번과 3번을 읽어야 한다(+로 표시됨). 문제가 무엇인지 확실해졌다. 패리티 디스크는 이러한 유형의 워크로드가 들어오면 병목으로 작용한다. 이와 같은 경우를 패리티 기반의 RAID의 **small-write 문제**라고 부른다. 데이터 디스크가 병렬적으로 접근될 수 있다 하더라도 패리티 디스크는 어떠한 병렬 연산도 실현될 수 없다. 시스템에서의 모든 쓰기 요청은 패리티 디스크로 인해서 순차적으로 처리된다. 패리티 디스크는 논리 I/O 당 읽기 1회와 쓰기 1회의 두 번의 I/O를 처리해야 하기 때문에, RAID-4에서 small random write의 성능을 패리티 디스크에서 발생하는 두 개 I/O의 처리 성능을 계산하여 얻을 수 있다. 결과적으로 $(R/2)$ MB/s를 얻게 된다. Small random write의 환경에서 RAID-4의 처리성능은 끔직하다. 디스크를 시스템에 추가한다고 하여 성능이 나아지지 않는다.

RAID-4의 I/O 지연 시간을 분석하는 것으로 논의를 마치도록 한다. 이제는 알고 있겠지만, 고장이 없다고 가정하면 읽기 요청 하나는 하나의 디스크로 전달되기 때문에 한 개의 읽기 요청의 지연 시간은 하나의 디스크 요청의 지연 시간과 같다. 쓰기 요청 한 개의 지연 시간은 읽기 2번과 쓰기 2번이 필요하다. 읽기와 쓰기는 병렬적으로 처리될

수 있기 때문에 전체 지연 시간은 한 개의 디스크 요청의 두 배 정도이다(두 개의 읽기가 끝나기를 기다려야 하기 때문에, 최악의 위치 잡기 시간이 걸리지만, 대신 갱신은 탐색 비용이 들지 않기 때문에 평균 위치 잡기 비용보다는 좋을 수 있기 때문에 정확히 두 배는 아니다).

41.7 RAID 레벨 5: 순환 패리티

Small write 문제를 해결하기 위해서(최소한 부분적으로라도), Patterson, Gibson과 Katz는 RAID-5를 소개하였다. RAID-5는 RAID-4와 거의 동일하게 동작하지만 패리티 블럭을 **순환**(**rotate**)시킨다는 점이 다르다(그림 41.7)

디스크 0	디스크 1	디스크 2	디스크 3	디스크 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

〈그림 41.7〉 **순환하는 패리티를 사용하는 RAID-5**

보는 바와 같이 각 스트라이프를 위한 패리티 블럭은 RAID-4의 패리티 디스크의 병목 현상을 없애기 위해 전체 디스크에 걸쳐 순환 배치된다.

RAID-5 분석

RAID-5 분석의 많은 부분은 RAID-4와 동일하다. 예를 들어, 두 레벨의 유효 용량과 결함 허용 정도는 동일하다. 따라서 순차 읽기와 쓰기 성능도 동일하다. 한 개의 요청의 지연 시간(읽기이든 쓰기든) 역시 RAID-4와 동일하다.

모든 디스크를 다 활용할 수 있기 때문에 랜덤 읽기 성능은 약간 더 좋다. 마지막으로 랜덤 쓰기 성능은 요청들을 병렬적으로 처리할 수 있기 때문에 RAID-4 대비 눈에 띄게 개선된다. 블럭 1번과 블럭 10번에 대한 쓰기가 요청되었다고 해 보자. 각 요청은 디스크 1번과 디스크 4번(블럭 1번 해당 패리티)에 대한 요청으로 바뀌고, 디스크 0번과 디스크 2번(블럭 10번과 해당 패리티)에 대한 요청으로 바뀐다. 그러므로 병렬적으로 처리할 수 있게 된다. 사실 랜덤 요청이 아주 많다면 모든 디스크들을 고르게 사용할 수 있다고 가정할 수 있다. 만약 그런 경우라면 small write에 대한 전체 대역폭은 $\frac{N}{4} \cdot R$ MB/s가 될 것이다. 사분의 일로 감소되는 것은 각 RAID-5는 여전히 총 4개의 I/O 연산을 유발하기 때문이다. 패리티 기반 RAID를 사용할 때 어쩔 수 없이 지불해야 하는 비용이다.

RAID-4의 성능이 좋은 몇 가지 예외를 제외한다면 RAID-5의 성능이 기본적으로 RAID-4의 성능과 동일하기 때문에 RAID-5는 시장에서 RAID-4를 완전히 대체하였다.

RAID-4가 쓰이는 경우는 large write만 발생하기 때문에 small write 문제가 전혀 존재하지 않는 시스템 뿐이다 [HLM94]. 이러한 경우, 구현이 더 간단하기 때문에 RAID-4가 가끔 사용된다.

41.8 RAID 비교: 정리

그림 41.8에서 RAID 레벨 사이의 간단한 비교를 정리하였다. 분석을 간단히 하기 위해서 세부 사항들은 생략하였다. 예를 들어, 미러링 기반의 시스템에서 쓰기 연산 시 평균 탐색 시간은 단일 디스크에 쓸 때의 시간보다 약간 더 길다. 두 개의 탐색 시간 중(각 디스크에 하나씩) 더 큰 값이 RAID-1의 탐색 시간이 되기 때문이다. 두 개의 디스크에서 랜덤 쓰기 성능은 일반적으로 하나의 디스크에 요청되는 랜덤 쓰기 성능보다 더 작다. 그리고 RAID-4/5의 패리티 디스크에 갱신할 때에는 먼저 예전의 패리티를 읽어야 하는데, 그것이 최대 탐색과 회전 지연을 일으킬 수 있지만 패리티에 두 번째 쓸 때에는 회전 지연만 있다는 사실도 있다.

하지만 그림 41.8의 비교는 기법간의 핵심 차이점을 잘 보여주고 있다. RAID 레벨들 간의 장단점을 파악하는 데 유용하다. 지연 시간 분석에서는 단일 디스크에 대한 요청 지연시간을 T로 표현하였다.

결론적으로 성능만을 원하고 신뢰성을 고려하지 않는다면 스트라이핑이 당연히 최고이다. 하지만 만약 임의 I/O의 성능과 신뢰성을 원한다면, 미러링이 최선이다. 하지만, 용량을 손해본다. 만약 용량과 신뢰성이 목적이라면, RAID-5가 승자이다. Small write

	RAID-0	**RAID-1**	**RAID-4**	**RAID-5**
용량	N	$N/2$	$N-1$	$N-1$
신뢰도	0	1 (반드시)	1	1
		$N/2$ (운이 좋다면)		
처리성능				
순차 읽기	$N \cdot S$	$(N/2) \cdot R$	$(N-1) \cdot S$	$(N-1) \cdot S$
순차 쓰기	$N \cdot S$	$(N/2) \cdot R$	$(N-1) \cdot S$	$(N-1) \cdot S$
랜덤 읽기	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
랜덤 쓰기	$N \cdot R$	$(N/2) \cdot R$	$(1/2) \cdot R$	$(N/4) \cdot R$
지연 시간				
읽기	T	T	T	T
쓰기	T	T	$2T$	$2T$

〈그림 41.8〉 **RAID의 용량, 신뢰도와 성능**

성능에 대한 비용만 지불하면 된다. 마지막으로 항상 순차 I/O만을 사용하고 용량을 극대화하기를 원한다면 RAID-5가 가장 적합하다.

41.9 RAID와 관련된 다른 흥미로운 주제들

RAID에서 고려해야 할 몇 가지 흥미로운 개념들이 있다. 나중에 이 책에 추가해야 할지도 모를 내용들이다.

레벨 2와 3, 복수의 디스크에서 발생하는 결함에 내성이 있는 레벨 6 등 여러 다른 RAID 설계 방법들이 있다 [Cor+04]. 디스크가 고장났을 때 RAID가 어떻게 대응하는지에 대한 논의도 있다. 때때로 **대체용 스페어**(**hot spare**)가 있어서 고장난 디스크를 즉시 대신하는 방법이 있다. 고장 발생 시 성능에 미치는 영향, 그리고 고장난 디스크를 복원할 때의 성능에는 어떤 영향을 미치는가? 그리고 **잠재된 섹터 오류**(**latent sector error**) 또는 **블럭 훼손**(**block corruption**)을 고려하는 결함 모델들도 있다 [Bai+08]. 여러 기법들이 이러한 결함들을 처리하고 있다(구체적인 내용을 위해서는 데이터 무결성에 대한 장을 참고하라). 마지막으로, 소프트웨어 레이어로 RAID를 구성할 수도 있다. 이러한 **소프트웨어 RAID** 시스템은 저렴하기는 하지만 일관성-유지 갱신 문제를 포함하여 다른 문제들을 가진다 [DAA05].

41.10 요약

RAID에 대해 논의하였다. RAID는 여러 개의 독립적인 디스크를 대용량의, 더 신뢰성 있는 하나의 개체로 변형시킨다. 더 중요한 것은 투명하게 이 일을 하기 때문에 상위 계층의 하드웨어와 소프트웨어는 변형 사실에 대해 인지하지 못한다.

다양한 RAID 레벨을 사용할 수 있는데, 최종 선택은 사용자가 중요하게 생각하는 것에 의해 결정된다. 예를 들어, 미러링 RAID는 간단하고 신뢰성이 있고 일반적으로 괜찮은 성능을 보이지만 용량면에서 불리하다. 대조적으로 RAID-5는 신뢰성이 높으며 용량면에서 우수하지만, small write 에서의 성능이 좋지 않다. 특정 워크로드에 적합한 RAID를 선택하고 변수 값들(청크 크기, 디스크의 수 등)을 제대로 결정하는 것은 도전적이고, 과학보다는 예술의 영역에 속한다.

참고 문헌

[Bai+08] **"An Analysis of Data Corruption in the Storage Stack"**
LakshmiN. Bairavasundaram, Garth R.Goodson, Bianca Schroeder, Andrea C.Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau
FAST '08, San Jose, CA, February 2008
얼마나 자주 디스크들이 데이터를 훼손시키는지를 연구한 우리의 논문이다. 자주는 아니지만 때때로 훼손시킨다! 그러므로 어떤 신뢰할 수 있는 저장 시스템을 고려해야 한다.

[BG88] **"Disk Shadowing"**
D. Bitton and J. Gray
VLDB 1988
미러링을 다룬 최초의 논문 중 하나로 이 논문에서는 "그림자 만들기(shadowing)"라고 하였다.

[CL95] **"Striping in a RAID level 5 disk array"**
Peter M. Chen and Edward K. Lee
SIGMETRICS 1995
RAID-5 디스크 배열의 중요한 몇 가지 변수들에 대해 잘 분석되어 있다.

[Cor+04] **"Row-Diagonal Parity for Double Disk Failure Correction"**
P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, and S. Sankar J. Leong
FAST '04, February 2004
두 개의 디스크를 패리티로 사용하는 RAID 시스템에 대한 첫 번째 논문은 아니지만 그 개념에 대한 아주 이해하기 쉽게 작성된 최근의 논문이다. 더 자세히 알고 싶다면 읽어보라.

[DAA05] **"Journal-guided Resynchronization for Software RAID"**
Timothy E. Denehy, A. Arpaci-Dusseau, and R. Arpaci-Dusseau
FAST 2005
우리가 제안한 일관성-유지 갱신 문제에 대한 해법이다. 제안한 해법은 소프트웨어 RAID의 경우에 대한 것으로 파일 시스템의 저널링이라는 장치와 소프트웨어 RAID를 통합시키도록 하였다.

[HLM94] **"File System Design for an NFS File Server Appliance"**
Dave Hitz, James Lau, and Michael Malcolm
USENIX Winter 1994, San Francisco, California, 1994
획기적인 저장장치인 NetApp 파일 서버의 근간을 이루는 어디든-쓰기 파일 배치(write-anywhere file layout) 또는 WAFL 파일 시스템을 소개하는 보기 드문 논문이다.

[Kim86] **"Synchronized Disk Interleaving"**
M.Y. Kim.
IEEE Transactions on Computers, Volume C-35: 11, November 1986
RAID에 대한 초기 연구이다.

[Kur88] **"Small Disk Arrays - The Emerging Approach to High Performance"**
F. Kurzweil.
Presentation at Spring COMPCON '88, March 1, 1988, San Francisco, California
RAID에 대한 또 다른 초기 참고 문헌이다.

[PB86] **"Providing Fault Tolerance in Parallel Secondary Storage Systems"**
A. Park and K. Balasubramaniam
Department of Computer Science, Princeton, CS-TR-O57-86, November 1986

또 다른 RAID에 대한 초기 연구이다.

[PGK88] **"Redundant Arrays of Inexpensive Disks"**
D. Patterson, G. Gibson, and R. Katz.
SIGMOD 1988
유명한 Patterson, Gibson과 Katz가 쓴 RAID에 대한 최고의 논문이다. 그 이후로 이 논문은 시간의 시련을 이겨냈으며 RAID라는 이름과 더불어 RAID의 시대로 안내하였다.

[S09] **"Barracuda ES.2 data sheet"**
URL: `http://www.seagate.com/docs/pdf/datasheet/disc/ds/cheetah/15k/5.pdf`
기술 명세서, 각오하고 읽어야 한다. 어떤 각오? 지루함.

[SG86] **"Disk Striping"**
K. Salem and H. Garcia-Molina.
IEEE International Conference on Data Engineering, 1986
마찬가지로 또 다른 RAID에 대한 초기 연구이다. RAID 논문이 SIGMOD에 투고된 이후에 갑자기 이런 류의 논문이 많이 나타났다.

[Sch84] **"Byzantine Generals in Action: Implementing Fail-Stop Processors"**
F.B. Schneider.
ACM Transactions on Computer Systems, 2(2):145154, May 1984
마침내 RAID가 아닌 다른 논문이다! 이 논문은 시스템이 어떻게 고장나는지, fail-stop(고장 시-정지) 방식으로 동작하도록 만드는 방법에 관한 논문이다.

숙제

이 절에서는 `raid.py`라는 툴을 소개한다. 간단한 RAID 시뮬레이터로서 RAID 시스템이 어떻게 동작하는지에 대한 당신의 지식을 든든하게 만들어 줄 것이다. README를 참고하도록 하자.

문제

1. 시뮬레이터를 사용하여 기본적인 RAID 매핑 검사들을 해 보자. 서로 다른 레벨들(0, 1, 4, 5)을 실행해 보고 요청들에 대한 매핑 정보를 파악할 수 있는지 살펴보자. RAID-5의 경우에는 좌측 대칭과 좌측 비대칭 배치 간의 차이를 알아보자. 다른 랜덤 시드를 사용하여 위와는 다른 문제를 만들어 보라.

2. 청크 크기를 -C 옵션으로 변경해 가면서 1번과 동일한 실험을 해 보라. 청크의 크기가 매핑을 어떻게 변경시키는가?

3. -r 플래그를 사용하여 각 문제의 특성을 뒤집어 위와 동일한 실험을 해 보라.

4. 뒤집기 플래그를 사용하지만 -S 플래그를 사용하여 각 요청의 크기를 증가시키자. 크기를 8 k, 12 k 그리고 16 k로 설정하고 RAID 레벨을 바꾸어 보자. 요청의 크기가 커지면 I/O 패턴이 어떻게 바뀌는가? 순차 워크로드에 대해서도 해 보자(-W sequential). RAID-4와 RAID-5의 I/O 성능이 훨씬 효율적일 때 요청의 크기는 얼마인가?

5. 4개의 디스크를 사용하여 RAID 레벨을 변경해 가면서 시뮬레이터의 시간 측정 모드를 사용하여(-t) RAID에서 랜덤 읽기를 100번 할 때의 성능을 추정해 보라.

6. 위와 동일하게 하고 대신 디스크의 수를 늘려보자. 사용된 디스크의 수를 증가시킬 때에 각 RAID 레벨의 성능은 어떻게 달라지는가?

7. 위와 동일하게 하되 읽기 요청 대신 모두 쓰기로 바꾸어서 해 보자(-w 100). 이번에는 RAID의 성능이 어떻게 달라지는가? 랜덤 쓰기를 100번 수행하는 워크로드가 완료되는 데 걸리는 대략적인 시간을 예측할 수 있는가?

8. 마지막으로 한 번 더 시간 측정 모드를 사용하여 순차 워크로드를 측정해 보자(-W sequential). RAID의 레벨에 따라 성능이 어떻게 변하는가 그리고 읽기와 쓰기를 비교해 보면 어떤가? 또 각 요청의 크기를 변경해보면 성능은 어떻게 바뀌는가? RAID-4와 RAID-5를 사용할 때에 쓰기 요청의 크기를 얼마로 정해야 할까?