38

Redundant Arrays of Inexpensive Disks (RAIDs)

Bir disk kullandığımızda bazen daha hızlı olmasını isteriz. G/Ç işlemleri yavaştır ve bu nedenle tüm sistem için engel olabilir. Bir disk kullandığımızda, bazen daha büyük olmasını isteriz; giderek daha fazla veri çevrimiçi oluyor ve böylece disklerimiz daha da dolu hale geliyor. Bir diski kullandığımızda , bazen daha güvenilir olmasını isteriz; bir disk arızalandığında , eğer verilerimiz yedeklenmediyse, tüm bu değerli veriler kaybolur.

NELER OLUYOR: BÜYÜK , HIZLI , GÜVENİLİR BİR DİSK NASIL YAPILIR ?

Büyük, hızlı ve güvenilir depolama sistemini nasıl yapabiliriz? Anahtar teknikler ne ? Farklı yaklaşımlar arasındaki dengeler nelerdir?

Bu bölümde, daha çok **gereksiz dizi ucuz diskler (RAID)** [P+88] olarak bilinen **gereksiz dizi ucuz diskleri (Redundant Array of Inexpensive Disks)** tanıtıyoruz, Bir teknikte birden fazla disk kullanmak için daha hızlı, daha büyük ve daha güvenilir bir disk sistemi yapmaktır. Terim 1980 'lerin sonunda U.C. de bir grup araştırmacı tarafından tanıtıldı. Berke ley (Profesörler David Patterson ve Randy Katz liderliğinde ve ardından da öğrenci Garth GibsonBu sıralarda daha iyi bir depolama sistemi yapısı için birçok farklı araştırmacı aynı anda çoklu diskleri oluşturmak için kullanma temel firine vardı [BG88, K86,K88,PB86,SG86].

Harici olarak, RAID bir diske benzer: Bir grup blokları okunabilir ya da yazabilir. Dahili olarak, RAID; çoklu diskler, bellekten oluşan karmaşık bir canavardır (hem uçucu hem de olmayan), bir veya daha fazla işlemci sitemi yönetir. Bir donanım RAID' İ bir bilgisayara çok benzer bir grup diskleri yönetme görevi için uzmanlaşmış sistemdir.

RAID'ler, tek bir diskte göre çok sayıda avantaj sunar. İlk avantaj performanstır. Paralel olarak birden fazla disk kullanmak G/Ç sürelerini büyük ölçüde hızlandırabilir. Diğer bir avantaj ise kapasitedir. Büyük veri kümeleri büyük diskleri ister. Son olarak, RAID'ler güvenilirliği arttırabilir; verileri birden çok diske yaymak (RAID teknikleri olmadan), tek bir diskin kaybı verileri saldırılara karşı savunmasız hale getirir; bir tür **fazlalık (redundancy)** ile bir diskin kaybı RAID' ler tolere edebilir ve hiçbir sorun yokmuş gibi çalışmaya devam edin.

İPUCU: ŞEFFAFLIK DAĞITIMI SAĞLAR

Sistemin geri kalanında hiçbir değişiklik gerektirmeyecek şekilde bir sisteme yeni işlevlerin nasıl ekleneceği düşünülürken, her zaman bu tür işlevlerin **şeffaf olarak** (**transparently)** eklenip eklenmeyeceğini göz önünde bulundurun, mevcut yazılımın tamamen yeniden yazılmasını (veya radikal donanım değişikliklerini) gerektirmesi bir fikrin etki şansını azaltır. RAID mükemmel bir örnektir ve şeffaflığı kesinlikle başarısına katkıda bulunur; yöneticiler bir SCSI tabanlı bir RAID depolama dizisi kurabilir ve kullanmaya başlamak için sistemin geri kalanının (ana bilgisayar, işletim sistemi vb.) bit bit bile değiştirmesi gerekmez. Bu **dağıtım** (**deployment)** sorununu çözerek RAID ilk günden daha başarılıdır.

Şaşırtıcı bir şekilde, RAID' ler bu avantajları sistemlere **şeffaf olarak (transparently)** sağlar. RAID, ana sistem için sadece büyük bir disk gibi görünür. Şeffaflığın güzelliği, RAID'li ve tek bir yazılım satırını değiştirmeyen bir disk; işletim sistemi ve istemci uygulamaları herhangi bir değişiklik yapılmadan çalışmaya devam eder. Bu şekilde şeffaflık, RAID'in **dağıtılabilirliğini (deployability)** büyük ölçüde geliştirerek, kullanıcıların ve yöneticilerin yazılım uyumluluğu endişesi olmadan bir RAID 'i kullanmalarına olanak tanır.

Şimdi RAID'lerin bazı önemli yönlerini tartışacağız. Ara yüzle başlıyoruz, hata modeli ile daha sonra nasıl değerlendirebileceğini tartışırız. Üç önemli eksen boyunca RAID tasarımı: kapasite, güvenilirlik ve performans. Daha sonra, bizim için önemli olan bir RAID tasarımı ve uygulamasını başka konuda tartışırız.

38.1 Arayüz ve RAID Dahili Bileşenleri

Yukarıdaki bir dosya sistemine göre, RAID büyük,(umarım) hızlı ve (umarım) güvenilir bir disk. Tıpkı tek bir diskte olduğu gibi, kendisini her biri dosya tarafından okunabilen veya yazılabilen doğrusal bir blok dizisi sitemi (veya başka bir istemci) sunar.

Bir dosya sistemi RAID'e mantıksal bir G/Ç isteği göderdiğinde, RAID dahili olarak isteği tamamlamak için hangi diske (ya da disklere) erişeceğini hesaplamalı ve bunu yapmak için bir ya da daha fazla fiziksel G/Ç göndermelidir. Bu fiziksel G/Ç' lerin tam doğası RAID seviyesine bağlıdır. Aşağıda ayrıntılı olarak ele alınmaktadır. Ancak basit bir örnek olarak, bir RAID' i ele alalım. Her bloğun iki kopyasını tutan (her biri ayrı bir diskte): Böyle bir **yansıtılmış (mirrored)** RAID sistemine yazarken, RAID'in verdiği her bir mantıksal G/Ç için iki fiziksel G/Ç gerçekleştirmesi gerekecektir.

Bir RAID sistemi genellikle bir ana bilgisayara standart bir bağlantı (örneğin SCSI veya SATA) ile ayrı bir donanım kutusu olarak oluşturulur.

Bununla birlikte, RAID'ler dahili olarak oldukça karmaşıktır, RAID'in çalışmasını yönlendirmek, veri bloklarını okurken ve yazarken tampolanmak için DRAM gibi uçucu bellek ve bazı durumlarda, yazmaları güvenli bir şekilde tampolanmak için uçucu olmayan bellek ve hatta belki de eşlik hesaplamaları yapmak için özel mantık ve aygıt yazılımını çalıştıran bir mikrodenetleyiciden oluşur (aşağıda da göreceğimiz gibi bazı RAID seviyelerinde yararlı). Yüksek düzeyde, bir RAID çok özel bir bilgisayar sistemidir: bir işlemcisi, belleği ve diskleri vardır; ancak, uygulamaları çalıştırmak yerine RAID'I çalıştırmak için tasarlanmış özel yazılımları çalıştırır.

38.2 Hata Modeli

RAID'I anlamak ve farklı yaklaşımları karşılaştırmak için, bir hata modeli düşüncesine sahip olmalıyız. RAID'ler, belirli türde disk arızaları tespit etmek ve kurtarmak için tasarlanmıştır; böylece tam olarak hangi hataların bekleneceğini bilmek çalışan bir tasarıma ulaşmada kritik öneme sahiptir.

Varsayacağımız ilk hata modeli oldukça basittir, ve **arıza-durdurma (fail-stop)** hata modeli olarak adlandırılır [S84]. Bu modelde, bir disk tam olarak iki durumdan biri; çalışıyor veya başarısız. Çalışan bir diskle, tüm bloklar okunabilir veya yazılabilir. Buna karşılık, bir disk arızalandığında kalıcı olarak kaybolduğunu varsayalım.

Arıza durdurma modelinin kritik bir yönü, arıza tespitini varsayar. Özellikle, bir disk arızalandığında bunu kolayca tespit edebilir. Örneğin, bir RAID dizisinde bir disk arızalandığında RAID denetleyicisi donanımı (veya yazılımı) hemen gözlemleyebilir.

Dolayısıyla, şimdilik daha karmaşık disk bozulması gibi arızalar "sessiz" konular hakkında endişelenmemize gerek yoktur. Ayrıca çalışan bir diskte tek bir bloğun erişilemez hale gelmesi (bazen gizli sektör olarak adlandırılır) konusunda endişelenmemize gerek yoktur. Bunları daha karmaşık olarak disk arızalarını daha sonra ele alacağız (ve maalesef daha gerçekçi).

38.3 Bir RAID Nasıl Değerlendirilir

Yakında göreceğimiz gibi, dizide RAID oluşturmak için farklı yaklaşımlar vardır. Güçlü ve zayıf yönlerini anlamak için değerlendirmeye değer olan zayıflıklar, bu yaklaşımların her biri farklı özelliklere sahiptir.

Özellikle, her bir RAID tasarımını üç eksende değerlendireceğiz. Bu ilk eksen **kapasitedir (capacity);** her biri B bloklu N diskten oluşan bir set verildiğinde, ne kadarRAID'in istemcileri için yararlı kapasite mevcut mu? Yedekleme olmadan, cevap $N \cdot B$ 'dir; bunun aksine, iki kopya tutan bir sistemimiz varsa **(yansıtma (mirroring)** olarak adlandırılır),($N \cdot B)/2$ 'lik faydalı bir kapasite elde ederiz. Farklı şemalar (örneğin, eşlik tabanlı olanlar) bu ikisinin arasında kalma eğilimindedir.

İkinci değerlendirme ekseni ise **güvenilirliktir (reliablity).** Kaç disk hatası olabilir? Verilen tasarımı tolere edebilir mi? Hata modelimizle uyumlu olarak şunları varsayarız; tüm bir diskin arızalanabileceğini;

daha sonraki bölümlerde (yani, verinin bütünlüğü konusunda), daha karmaşık arıza modlarını nasıl ele alacağımızı düşüneceğiz.

Son olarak, üçüncü eksen **performanstır (performance)**. Performansı değerlendirmek biraz zordur, çünkü büyük ölçüde disk dizisine önceden göderilen iş yüküne bağlıdır. Bu nedenle, performansı değerlendirmeden once, ilk olarak dikkate alınması gereken bir dizi tipik iş yükü sunar.

Şimdi üç önemli RAID tasarımını ele alalım: RAID seviye 0 (şeritleme), RAID seviye 1 (yansıtma) ve RAID seviye 4/5 (eşlik tabanlı artıklık). Bu tasarımların her birinin bir "seviye" olarak adlandırılması Patterson, Gibson ve Katz'ın Berkeley'deki öncü çalışmalarından kaynaklanmaktadır [P+88].

38.4 RAID Seviye 0 : Şeritleme

İlk RAID seviyesi aslında bir RAID seviyesi değildir, çünkü fazlalık yoktur. Ancak RAID seviye 0 ya da daha iyi bilinen adıyla **şeritleme (striping),** performans ve kapasite üzerinde mükemmel bir üst sınır görevi görür ve bu nedenle anlaşılmaya değerdir.

En basit şeritleme **şerit (stripe)** blokları diskler arasında sistemi aşağıdaki gibidir (burada 4 diskli bir dizi olduğunu varsayalım):

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

Şekil 38.1: **RAID-0: Basit Şeritleme**

Şekil 38.1'den temel fikri edineblirsiniz: dizinin bloklarını yaymak diskler arasında yuvarlak robin tarzındadır. Bu yaklaşım için tasarlanmıştır, dizinin bitişik parçaları için istekler yapıldığında dizinin en fazla paralelliğini elde edin (örneğin büyük, sıralı bir okumada olduğu gibi). Aynı satırdaki bloklara bir **şerit (stripe)** diyoruz; böylece 0,1,2 ve 3 tanesi yukarıdaki gibi aynı şerittedir.

Örnekte, yalnızca 1' in basitleştirici varsayımını yaptık. Blok (her biri 4KB boyutunda), devam etmeden once, sonraki her diske yerleştirilir. Ancak, bu düzenlemenin geçerli olması gerekmez. Örneğin blokları diskler arasında Şekil 38.2' deki gibi düzenleyebiliriz:

Disk 0	Disk 1	Disk 2	Disk 3	
0	2	4	6	Yığın buyutu:
1	3	5	7	2 blok
8	10	12	14	
9	11	13	15	

Şekil 38.2: **Daha Büyük Yığın Boyutu ile Şeritleme**

Bu örnekte, bir sonraki diske hareket etmeden önce her diske iki adet 4KB blok yerleştiriyoruz. Dolayısıyla, bu RAID dizisinin **yığın boyutu (chunk size)** 8 KB'dır ve böylece bir şerit 4 parçadan veya 32 KB veriden oluşur.

KENAR: RAID EŞLEME PROBLEMİ

Kapasite, güvenilirlik ve performans özelliklerini incelemeden önce RAID'in bir parçası olarak, öncelikle **eşleme problemi (the mapping problem)** olarak adlandırdığımız bir konuyu ele alacağız. Bu sorun tüm RAID dizilerinde ortaya çıkar; basitçe söylemek gerekirse, bir mantıksal RAID tam olarak hangi fiziksel bloğun disk ve erişim için ofsetin okunacağını veya yazılacağını nasıl bilir?

Bu basit RAID seviyeleri için, mantıksal blokları fiziksel konumlarıyla doğru bir şekilde eşleştirmek için çok fazla karmaşıklığa ihtiyacımız yoktur. Yukarıdaki ilk şeritleme örneğini alın (yığın boyutu = 1 blok = 4KB). Bu durumda, mantıksal blok sadresi A verildiğinde, RAID istenen diski kolayca hesaplayabilir ve iki basit denklemle dengeleyebilir:

```
Disk   = A %diskler_için_sayı
Offset = A /diskler_için_sayı
```

Bunların hepsinin tamsayı işlemleri olduğuna dikkat edin (4/3 = 1.333… değil 1 dir). Basit bir örnek için bu denklemlerin nasıl çalıştığını görelim. Düşünün ki bunun üzerindeki ilk RAID, blok 14 için bir istek gelir. 4 disk olduğu göz önüne alındığında, bu ilgilendiğimiz diskin (14 % 4 = 2) olduğu anlamına gelir: disk2 . Tam blok (14 / 4 = 3) olarak hesaplanır: blok3. Böylece blok 14, üçüncü bloğun dördüncü bloğunda (blok3, 0'dan başlayarak) bulunmalıdır. Disk (disk2, 0'dan başlayarak)tam bulunduğu yerdir. Desteklemek için bu denklemlerin nasıl değiştirileceğini düşünebilirsiniz. Farklı parça boyutlarını dene! Çok zor değil.

Yığın boyutları

Yığın boyutu çoğunlukla dizinin performansını etkiler. Örneğin, küçük bir yığın boyutu birçok dosyanın birçok diske şeritleneceği anlamına gelir, bu nedenle tek bir dosyaya okuma ve yazma işlemlerinin paralelliğini artırır; ancak birden fazla diskteki bloklara erişim için konumlandırma süresi artar, çünkü tüm istek için konumlandırma süresi maksimum tüm sürücülerdeki isteklerin konumlandırma sürelerinin toplamıdır.

Diğer yandan büyük bir yığın boyutu, bu tür dosya içi paralelliği azaltır ve bu nedenle yüksek performans elde etmek için birden çok eşzamanlı isteğe dayanır. Ancak, büyük parça boyutları konumlandırma süresini azaltır; eğer için örneğin, tek bir dosya bir yığının içine sığar ve bu nedenle tek bir dosyaya yerleştirilir. Diske erişirken oluşan konumlandırma süresi, yalnızca tek bir diskin konumlandırma süresi olacaktır.

Bu nedenle, disk sistemine sunulan iş yükü hakkında çok fazla bilgi gerektirdiğinden "en iyi" yığın boyutunu belirlemek zordur [CL95]. Bu tartışmanın geri kalanında, dizinin kullandığını varsayacağız tek bir bloğun (4KB) bir yığın boyutudur. Çoğu dizi daha büyük yığın boyutları kullanır (örneğin, 64KB), ancak aşağıda tartıştığımız sorunlar nedeniyle, tam yığın boyutu önemli değildir; bu nedenle basitlik adına tek bir blok kullanıyoruz.

RAID – 0 Analizine Geri Dön

Şimdi şeritlemenin kapasitesini, güvenilirliğini ve performansını değerlendirelim. Kapasite açısından bakıldığında mükemmeldir: her biri N boyutunda diskler verildiğinde B blok, şeritleme $N \cdot B$ blok kapasite sağlar. Güvenilirlik açısından, şeritleme de mükemmeldir, ancak kötü bir şekilde: herhangi bir disk arızası veri kaybına yol açacaktır. Son olarak, performans mükemmel: tüm diskler kullanıcı G/Ç isteklerine hizmet etmek için genellikle paralel olarak kullanılır.

RAID Performansının Değerlendirilmesi

RAID performansını analiz ederken iki farklı performans ölçütü göz önünde bulundurulabilir. Bunlardan ilki tek istek gecikmesidir. Bir RAID'e yapılan tek bir G/Ç isteğinin gecikme süresini anlamak, ne kadar gecikme olduğunu ortaya koyduğu için yararlıdır. Paralellik tek bir G/Ç işlemi sırasında var olabilir. İkincisi RAID' in kararlı-durum verimi, yani birçok RAID'in eşzamanlı istekleri toplam bant genişliğidir. RAID'ler genellikle ortamlarında yüksek performanslı cihazlarda kullanıldığından, kararlı-durum bant genişliği kritiktir ve bu nedenle analizlerimizin ana odak noktasıdır.

Verimi daha ayrıntılı olarak anlamak için bazı ilgilenilen iş yüklerini ortaya koymamız gerekir. Bu tartışma için şu varsayımlarda bulunacağız, iki tür iş yükü vardır: **sıralı (sequential)** ve **rastgele (random)**. Sıralı bir iş yükünde, diziye yapılan isteklerin büyük bitişik parçalar halinde geldiğini varsayarız; örneğin, diziye erişen bir istek (veya istek dizisi) 1MB veri, x bloğunda başlayıp (x+1 MB) bloğunda bitiyorsa, sıralı olarak Kabul edilir. Sıralı iş yükleri birçok ortamda yaygındır (bir anahtar kelime için büyük bir arama yapmayı düşünün) ve bu nedenle önemli sayılırlar.

Rastgele iş yükleri için her bir isteğin oldukça küçük olduğunu varsayarız, ve her isteğin disk üzerinde farklı bir rastgele konumda yapıldığını varsayalım. Örneğin, rastgele bir istek akışı ilk olarak mantıksal adreste 4KB'ye erişebilir, 10, sonra 550.000 mantıksal adresinde, sonra 20.100'de ve bu şekilde devam eder. Bir veritabanı yönetim sistemi (VTYS) üzerindeki işlemsel iş yükleri gibi bazı önemli iş yükleri bu tür bir erişim modeli sergiler ve bu nedenle önemli bir iş yükü olarak görülmüştür.

Elbette, gerçek iş yükleri o kadar basit değildir ve genellikle sıralı ve rastgele görünen bileşenlerin yanı sıra ikisi arasındaki davranışların bir karışımına sahiptir. Basitlik için sadece bu iki olasılığı göz önünde bulunduruyoruz.

Anlayacağınız üzere, sıralı ve rastgele iş yüklerinin geniş ölçüde performans özellikleri bir diskten farklıdır. Sıralı erişim ile bir disk en verimlimodunda çalışır, aramak için çok az zaman harcar ve rotasyon için bekler ve zamanının çoğunda veri aktarır. Rastgele ile erişimin tam tersi doğrudur: Çoğu zaman aramak ve beklemek için harcanır ve veri aktarımı için nispeten daha az zaman harcanır. Yakalamak için analizimizde bu farkı göz önünde bulundurarak, bir diskin sıralı bir iş yükü altında S MB/s hızında veri ve sıralı bir iş yükü altında R MB/s hızında veri rastgele iş yükü genel olarak S, R' den çok daha büyüktür (yani $S \gg R$).

Bu farkı anladığınızdan emin olmak için basit bir alıştırma yapalım. Özellikle, aşağıdaki disk özellikleri göz önüne alındığında S ve R'yi hesaplayalım. Ortalama olarak 10 MB boyutunda sıralı bir aktarım ve ortalama 10KB rastgele aktarımı varsayalım. Ayrıca, aşağıdaki disk özelliklerini düşünelim:

Ortalama arama süresi 7ms
Ortalama Dönme Gecikmesi 3 ms
Diskin aktarım hızı 50 MB/s

S'yi anlamak için öncelikle tipik bir 10 MB aktarımla zamanın nasıl harcandığını bulmamız gerekir. İlk önce 7 ms arayarak, sonra 3 ms dönerek geçiririz. Son olarak transfer başlar, 50 MB/s' sn'sinde 10MB, saniyenin 1/5 ine yol açar veya 200 ms, transferde harcanır. Böylece, 10MB istek için 210 ms harcıyoruz. S' yi hesaplamak için sadece bölmemiz gerekiyor:

$$S = \frac{Veri\ Miktarı}{Erişim\ süresi} = \frac{10\ MB}{210\ ms} = 47.62\ MB/s$$

Gördüğümüz gibi, veri aktarımı için harcanan büyük zaman nedeniyle Sdiskinin en yüksek bant genişliğine çok yakındır (arama ve dönme maliyetleri amorti edilmiştir).

R'yi de benzer şekilde hesaplayabiliriz. Arama ve rotasyon aynıdır, daha sonra aktarımda harcanan süreyi hesaplarız, buda 50 MB/sn'de 10KB veya 0.195Ms'dir.

$$R = \frac{Veri\ miktarı}{Erişim\ zamanı} = \frac{10\ KB}{10.195\ ms} = 0.981\ MB/s$$

Gördüğünüz gibi, R 1MB / s ' den az ve S/R neredeyse 50'dir

Tekrar RAID-0 Analizine geri dön

Şimdi şeritleme performansını değerlendirelim. Yukarıda söylediğimiz gibi genellikle iyidir. Gecikme perspektifinden bakıldığında , örneğin, bir tek blokluk istek, tek bir diskin isteğiyle hemen hemen aynı olmalıdır. Sonuçta RAID-0 bu isteği kendi disklerinden birine yönlendirecektir.

Kararlı durum sıralı verim açısından bakıldığında, sistemin tüm bant genişliğini elde etmeyi bekleriz. Böylece, verim eşittir N (disk sayısı) ile S (bir diskin sıralı bant genişliği) çarpımı tek disktir. Çok sayıda rastgele G/Ç için N.R MB/s elde edilir. Aşağıda göreceğimiz gibi bu değerler hem hesalanması en basit olanlardır hem de diğer RAID seviyeleri ile karşılaştırmasıdır.

38.5 RAID Düzey 1 : Yansıtma

Şeritlemenin ötesindeki ilk RAID seviyemiz, RAID seviye 1 olarak bilinir veya yansıtma. Yansıtılmış bir sistemle birden fazla kopya oluşturuyoruz, sistemdeki her bloğun; tabiki disket, her kopya ayrı yere yerleştirilmelidir. Bunu yaparak disk arızalarını tolere edebiliriz.

Tipik bir yansıtılmış sistemde, her mantıksal blok için RAID'in iki fiziksel kopyasını sakladığını varsayacağız. İşte örnek:

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

Şekil 38.3: **Basit RAID-1: Yansıtma**

Örnekte, disk 0 ve disk 1 aynı içeriğe sahiptir, ve disk 2 ve disk 3 de aynıdır; veriler bu yansıtılmış çiftleri boyunca şeritlenmiştir. Aslında, blok kopyalarını disklere yerleştirin ve bunu yapmanın birkaç farklı yolu olduğunu fark etmiş olabilirsiniz. Yukarıdaki düzenleme yaygın bir düzenlemedir bazen **RAID-10** (veya **RAID 1+0**, **yansıtılmış şerit**) olarak adlandırılır çünkü yansıtılmış çiftleri (RAID-1) ve ardından şeritleri (RAID-0) kullanır; diğer bir yaygın düzenleme ise iki büyük şeritleme (RAID-0) dizisi içeren RAID-01 (veya RAID 0+1, şeritlerin aynası) ve daha sonra bunların üzerine aynalar vardır (RAID-1)

Yansıtılmış bir diziden bir blok okurken, RAID'in bir seçeneği vardır: Her bir kopyayı da okuyabilir. For Örneğin, mantıksal blok 5'e is bir okuma verirse RAID, disk 2 veya disk 3'ten okur. Bununla birlikte, blok yazarken böyle bir seçenek yoktur. RAID güvenilirliği korumak için verilerin her iki kopyasını da güncellemelidir. Yine de, bu yazımların paralel olarak gerçekleşebileceğini unutmayın; örneğin, mantıksal blok 5'e yazma, aynı anda disk 2 ve disk 3'e ilerleyebilir.

RAID-1 Analizi

RAID-1'I değerlendirelim. Kapasite açısından bakıldığında RAID-1 masraflıdır; Yansıtma seviyesi 2 olduğunda, en yüksek faydalı kapasitemizin yalnızca yarısını elde ederiz. B blokluk N disk ile RAID-1 faydalı kapasitesi $(N \cdot B)/2$ dir.

Güvenirlik açısından RAID-1 iyi sonuç verir. Herhangi bir diskin arızalanmasını tolere edebilir. RAID'in biraz şansla bundan daha iyisini yapabileceğini fark edebilirsiniz. Yukarıdaki şekilde, disk 0 ve disk 2'nin her ikisininde arızalandığını düşünün. Böyle bir durumda veri kaybı olmaz! Daha genel olarak yansıtılmış bir sistem (yansıtma seviyesi 2 olan) kesin olarak 1 disk arızasını ve hangi disklerin arızalandığına bağlı olarak N/2'ye kadar arızayı tolere edebilir. Uygulamada, genellikle bu gibi şeyleri şansa bırakmak istemeyiz; bu nedenle çoğu kişi yansıtmanın tek bir arızayla başa çıkmak için iyi olduğunu düşünür.

Son olarak, performansı analiz ediyoruz. Gecikme açısından diske bakıldığında tek bi okuma isteğinin gecikme süresiyle aynı olduğunu görebiliriz. RAID-1'in tek yaptığı okumayı kopyalarından birine yönlendirmektir. Yazı biraz farklıdır. tamamlamadan önce iki fiziksel yazma işlemi gerektirr. Bu iki yazma işlemi paralel olarak gerçekleşir ve bu nedenle zaman kabaca tek bir yazma süresine eşittir; ancak mantıksal yazma her iki fiziksel yazmanın da tamamlanmasını beklemek zorunda olduğundan, iki istek en kötü durumdaki arama ve dönme gecikmesine maruz kalır ve bu nedenle (ortalama üzerinde) tek bir diske yazmadan biraz daha yüksek olacaktır.

KENAR: RAID TUTARLI GÜNCELLEME SORUNU

RAID-1'i analiz etmeden önce, ilk olarak ortaya çıkan bir sorunu tartışalım. **Tutarlı güncelleme sorunu (consistent-update problem)** olarak bilinen herhangi birçok diskli RAID sistemidir[DAA05]. Sorun tek bir mantıksal işlem sırasında birden çok diski güncellemesi gereken herhangi bir RAID'e yazma sırasında ortaya çıkar. Bu durumda, ikizlenmiş bir disk dizisini düşündüğümüzü varsayalım.

Yazma işlemininn RAID'e verildiğini ve ardından RAID'in şu kararı verdiğini düşünün; disk 0 ve disk 1 olmak üzere iki diske yazılmalıdır. RAID daha sonra şunları yayınlar ki; disk 0'a yazma işlemini gerçekleştirir, ancak RAID diske istek göndermeden hemen önce 1, bir güç kaybı (veya sistem çökmesi) meydana gelir. Bu talihsiz durumda, izin verin disk 0'a yapılan isteğin tamamlandığını varsayalım (ancak açıkça disk1, hiç yayınlanmadığı için vermedi).

Bu zamansız güç kaybının sonucu olarak bloğun iki kopyası artık **tutarsızdır (inconsistent)**; disk 0'daki kopya yeni sürümdür ve 1.diskteki eskidir. Olmasını istediğimiz şey, her ikisinin de durumu içindir. Diskler **atomik (atomically)** olarak değişecek, yani ya her ikisininde yeni versiyonu ya da hiçbiridir.

Bu sorunu çözmenin genel yolu, RAID'in ilk olarak ne yapmak üzere olduğunu kaydetmek için bazılarının **ileriye dönük yazma günlüğünü (write-ahead log)** kullanmaktır (yani iki diski güncellemek belirli bir veri parçası ile). Biz bu yaklaşımı benimseyerek bir çarpışma durumunda doğru şeyin olmasını sağlayabilir; bekleyen tüm işlemleri yeniden yürüten bir **kurtarma (recoveri)** ile yordamı çalıştırma RAID, senkronize değilken iki yaynalanmış kopya olmamasını sağlayabiliriz (RAID-1 durumunda).

Son bir not: her yazma işleminde diske günlük kaydı yapmak çok pahalı olduğundan, çoğu RAID donanımı az miktarda uçucu olmayan bu tür bir günlüğe kaydetme işlemini gerçekleştirdiği RAM'dir (örnek. Pil destekli). Böylece diske kaydetmenin yüksek maliyeti olmadan tutarlı güncelleme sağlanır.

Kararlı durum iş yoğunluğu verimini analiz etmek için sıralıyla başlayalım. Diske sıralı olarak yazarken, her mantıksal yazma iki fiziksel yazma ile sonuçlanır; örneğin, mantıksal blok yazdığımızda 0 (yukarıdaki şekilde), RAID dahili olarak bunu her iki diske yazacaktır (0-1). Böylece yansıtılmış bir diziye sıralı yazma sırasında elde edilen maksimum bant genişliğinin ($N/2 \cdot S$) veya yarısı tepe bant genişliğidir.

Ne yazık ki, sıralı okuma sırasında da aynı performansı elde ediyoruz. Sıralı bir okumanın daha iyisini yapabileceği düşünülebilir, çünkü verinin ikisini birden değil sadece bir kopyasını okuması gerekir. Ancak bunun neden yardımcı olmadığını göstermek için bir örnek kullanalım. Düşünün ki 0, 1, 2, 3, 4, 5, 6, ve 7 numaralı blokları okuması gerekir. Diyelim ki 0 disk 0'a , 1' in okunması disk 2'ye 2' nin okunması disk 1' e ve 3'ten disk 3'e yayınladığımızı düşünün. Sırasıyla 0, 2, 1 ve 3 numaralı disklere 4,5,6 ve 7 numaralı okumaları göndererek devam ediyoruz.

Tüm diskleri kullandığımız için dizinin tam bant genişliğine ulaştığımız saflıkla düşünülebilir.

Bununla birlikte, durumun (zorunlu olarak) böyle olmadığını görmek için, tek bir diskin aldığı istekleri düşünelim (diyelim ki disk0). İlk olarak, bir istek alır 0.blok; daha sonra 4.blok için bir istek alır(2.bloğu atlayarak). Aslında her disk her diğer blok için bir istek alır. Üzerinde dönerken atlanan blok istemciye yararlı bant genişliği sağlamıyor. Böylece her disk en yüksek bant genişliğinin yalnızca yarısını sağlayacaktır. Dolayısıyla sıralı okuma yalnızca (N/2 · *S*) MB/s ' lik bir bant genişliği elde edecektir.

Rstgele okumalar, yansıtılmış bir RAID için en iyi durumdur. Bu durumda biz okumaları tüm disklere dağıtabilir ve böylece mümkün olan tüm bant genişliğini elde edebiliriz. Böylece rastgele okumalar için RAID-1 *N* · *R* MB/s sağlar.

Son olarak rastgele yazmalar beklediğiniz gibi performans gösteriyor. N/2.R MB/sn. Her biri mantıksal yazmanın iki fiziksel yazmaya dönüşmesi gerekir ve böylece tüm diskler kullanımda olacaksa, istemci bunu yalnızca bant genişliğini mevcut disklerin yarısı olarak algılayacaktır. Mantıksal blok x'e yazma işlemi iki paralel bloğa dönüşse bile iki farklı fiziksel diske yazıldığında birçok küçük isteğin bant genişliği şeritleme ile gördüğümüzün yalnızca yarısına ulaşır. Yakında göreceğimiz gibi mevcut bant genişliğinin yarısını almak aslında çok iyi!

38.6 RAID Seviye 4: Eşlik ile Alan Tasarrufu

Şimdi bir disk dizisine artıklık eklemek için **eşlik (parity)** olarak bilinen farklı bir yöntem sunuyoruz. Eşlik tabanlı yaklaşımlar daha az kapasite kullanmaya ve böylece yansıtılmış sistemler tarafından ödenen büyük alan cezasının üstesinden gelmeye çalışır**.** Ancak bunu bir maliyetle yaparlar: performans.

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

Şekil 38.4: **Eşlikli RAID-4**

İşte örnek bir beş diskli RAID-4 sistemi (Şekil 38.4) her bir veri şeridi için yedek bilgileri depolayan tek bir **eşlik (parity)** bloğu ekledik. Örneğin, eşlik bloğu P1 4, 5, 6 ve 7. bloklardan hesapladığı gereksiz bilgilere sahiptir.

Eşliği hesaplamak için şeridimizdeki herhangi bir bloğun kaybına karşı koymamızı sağlayan matematiksel bir fonksiyon kullanmamız gerekir. Basit fonksiyon **özel veya (XOR)** nın 0 hata yaptığını ortaya çıkardı. Belirli küme için bitler, çift sayıda varsa tüm bu bitlerin XOR değeri 0, bitlerde 1 ve tek sayıda 1 varsa 1 dödürür. Örneğin: ilk satırda (0,0,1,1), iki adet 1 (C2, C3) vardır ve dolayısıyla bu değerlerin tümünün XOR'u 0 (P) olacaktır:

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

Benzer şekilde ikinci satırda yalnızca bir adet 1 (C1) vardır ve dolayısıyla XOR 1 (P) olmalıdır. Bunu basit bir şekilde hatırlayabilirsiniz: Eşlik biti de dahil olmak üzere herhangi bir satırdaki 1'lerin sayısıdır, çift (tek değil) bir sayı olmalıdır; bu RAID 'in **değişmezidir (invariant),** eşitliğin doğru olması için sürdürmesi gerekir.

Yukarıdaki örnekten, eşitliğin nasıl olduğunu da tahmin edebilirsiniz bilgiler bir arızadan kurtulmak için kullanılabilir. C2 etiketi sutununun kaybolduğunu düşünün. Sutunda hangi değerlerin olması gerektiğini bulmak için, sadece o satırdaki diğer tüm değerleri okumamız gerekir (XOR'lanmış eşlik biti de dahil) ve doğru cevabı **yeniden yapılandırın (reconstruct).** Özellikle varsayalım ki C2 sutunundaki ilk satırın değeri kaybolur(1'dir); diğer satırları okuyarak bu satırdaki değerler (C0'dan 0, C1'den 0, C3'ten 1 ve eşlikten 0 P sutununu) 0, 0, 1, ve 0 değerlerini elde ederiz. Çünkü biliyoruz ki XOR her satırda çift sayıda 1 tutarsa, eksik verilerin ne olduğunu biliriz ve şöyle olmalıdır; a 1. İşte XOR tabanlı eşlik şemasında yeniden yapılandırma bu şekilde çalışır! Yeniden yapılandırılan değeri nasıl hesapladığımıza da dikkat edin: İlk etapta eşliği hesapladığımız şekilde veri bitlerini ve eşilk bitlerini XOR'luyoruz.

Şimdi merak ediyor olabilirsiniz: tüm XOR'lamadan bahsediyoruz, bu bitler ve yin de yukarıdan RAID'in 4KB (veya daha büyük) bloklar; eşliği hesaplamak için XOR'u bir grup bloğa nasıl uygularız? Bunun da kolay olduğu ortaya çıktı. Basitçe bir veri bloklrının her biti boyunca bitsel XOR; her bitsel XOR'un sonucunu eşlik bloğundaki ilgili bit yuvasına yerleştirin. Örneğin, 4 bit büyüklüğünde bloklarımız olsaydı (evet, bu hala bir 4KB bloğu ancak resmi anladınız) şöyle görünebilirler:

Blok0	Blok1	Blok2	Blok3	Eşlik
00	10	11	10	11
10	01	00	01	10

Şekilden de görebileceğiniz gibi eşlik her bit için hesaplanır. Her bir blok ve sonuç eşlik bloğuna yerleştirilir.

RAID-4 Analizi

Şimdi RAID-4'ü analiz edelim. Kapasite açısından koruduğu her disk grubu için, eşlik bilgileri için disk RAID-4 1 kullanır. Böylece RAID grubu için yararlı kapasitemiz $(N-1)\cdot B'$ *dir*.

Güvenilirliği anlamak da oldukça kolaydır: RAID-4 1 arıza ve daha fazlası için değil diske tolere gösterir. Birden fazla disk kaybolursa kayıp verileri yeniden yapılandırmak için bu diskleri kullanmanın hiçbir yolu yoktur.

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

Şekil 38.5: **RAID-4'te Tam Şeritli Yazmalar**

Son olarak performans var. Bu kez sabit durum verimini analiz ederek başlayalım. Sıralı okuma performansı tüm diskleri kullanabilir eşlik diski hariç ve böylece etkin bir tepe bant genişliği sağlar $(N - 1) \cdot S$ MB/s (kolay bir durum).

Sıralı yazma işlemlerinin performansını anlamak için öncelikle bunların nasıl yapıldığını anlamamız gerekir. Diske büyük bir veri yığını yazarken RAID-4 **tam şeritli yazma (full-stripe write)** olarak bilinen bir optimizasyon gerçekleştirebilir. Örneğin 0,1,2 ve 3 bloklarının olduğu durumu düşünün. Yazma isteğinin bir parçası olarak RAID'e gönderilir (Şekil 38.5).

Bu durumda, RAID basitçe P0'ın yeni değerini hesaplayabilir (şu şekilde 0, 1, 2, ve 3 blokları arasında bir XOR gerçekleştirerek) ve ardından tüm blokları (eşlik bloğu dahil) yukarıdaki beş diske paralel olarak aktarır (şekilde gri rankle vurgulanmıştır). Bu nedenle tam şeritli yazma işlemleri RAID-4' ün diske yazmasının en verimli yoludur.

Tam şeritli yazmayı anladıktan sonra performansı hesaplamak RAID-4 üzerinde sıralı yazma işlemi kolaydır, etkin bant genişliği de $(N - 1) \cdot S$ MB/s'dir. Eşlik diski sürekli olarak kullanımda olsa bile işleminden sonra istemci bundan performans avantajı elde etmez.

Şimdi rastgele okumaların performansını analiz edelim. Ayrıca yapabileceğiniz gibi yukarıdaki şekle bakın, bir dizi 1 blok rastgele okuma yayılacak sistemin veri diskleri arasında, ancak eşlik diski arasında değildir. Böylece etkin performans: $(N - 1) \cdot R$ MB/s dir.

En sona sakladığımız rastgele yazmalar RAID-4 için en ilginç durumu oluşturmaktadır. Blok 1'in üzerine yazmak istediğimizi düşünün Yukarıdaki örnek. Devam edip üzerine yazabiliriz, ancak bu bizi bir sorunla baş başa bırakıyor: eşlik bloğu P0 artık şeridin doğru eşlik değerini doğru bir şekilde yansıtmayacaktır; bu örnekte P0'ın da güncellenmesi gerekir. Bunu hem doğru hem de verimli bir şekilde nasıl güncelleyebiliriz?

İki yöntem olduğu ortaya çıktı. İlki, **eklemeli eşlik (additive parity**) olarak bilinir, aşağıdakileri yapmamızı gerektiriyor. Yeni eşlik değerini hesaplamak için bloğunda, şeritteki diğer tüm veri bloklarını paralel olarak okuyun (içinde örneğin, 0, 2 ve 3 numaralı bloklar) ve bunları yeni blokla (1) XOR'layın. Bu sonuç yeni eşlik bloğunuzdur. Yazma işlemini tamamlamak için şunları yazabilirsiniz; yeni verileri ve yeni pariteyi yine paralel olarak ilgili disklere aktarır. Bu teknikle ilgili sorun şu ki, bu teknik diskler ve dolayısıyla daha büyük RAID'lerde eşlik hesaplamak için yüksek sayıda okuma gerektirir. Böylece, **eksiltici eşlik (subtractive parity)** yöntemi olur.Örneğin bu bit dizisini hayal edi (4 veri biti, bir eşlik):

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

C2 bitinin üzerine yeni bir değer yazmak istediğimizi düşünelim C2yeni olarak adlandıracağız. Çıkarma yöntemi üç adımda çalışır. Birincisi, C2'deki eski verileri (C2eski = 1) ve eski eşliği (Pold = 0) okuruz. Ardından, eski veriler ile yeni verileri karşılaştırırız; eğer aynı iseler (örneğin, C2yeni = C2eski), o zaman eşlik bitinin de aynı kalacağını biliriz ve aynıdır (yani, Pyeni = Peski). Bununla birlikte, eğer farklılarsa, o zaman eski eşlik bitini mevcut durumunun tersine, yani (Peski == 1) ise, Pyeni 0'a ayarlanacaktır; eğer (Peski == 0) ise, Pyeni 1'e ayarlanacaktır. Şöyle ifade edebiliriz tüm bu karmaşayı XOR (burada $\oplus$ XOR operatörüdür) ile düzgün bir şekilde çözebilirsiniz:

$$P_{yeni} = (C_{eski} \oplus C_{yeni}) \oplus P_{eski} \quad (38.1)$$

Bitlerle değil bloklarla uğraştığımız için, bu hesaplamayı bloktaki tüm bitler üzerinden yaparız (örneğin, her blokta 4096 bayt çarpı bayt başına 8 bit). Bu nedenle, çoğu durumda, yeni blok farklı olacaktır ve dolayısıyla yeni eşlik bloğu da eski bloktan daha büyük olacaktır.

Artık toplama eşliği hesaplamasını ne zaman kullanacağımızı ve çıkarma yöntemini ne zaman kullanacağımızı anlayabilmelisiniz. Düşünün sistemde kaç tane disk olması gerektiği konusunda yöntemi, çıkarma yöntemine göre daha az G/Ç gerçekleştirir; bunların kesişme noktası ne?

Bu performans analizi için, çıkarma yöntemini kullandığımızı varsayalım. Bu nedenle, her yazma işlemi için RAID'in 4 fiziksel G/Ç gerçekleştirmesi gerekir (iki okuma ve iki yazma). Şimdi çok sayıda yazı olduğunu hayal edin RAID'e gönderildi; RAID-4 paralel olarak kaç tane gerçekleştirebilir? Anlamak için RAID-4 düzenine bir kez daha bakalım (Şekil 38.6).

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

Şekil 38.6: **Örnek 4,13 ve İlgili Eşlik Bloklarına Yazma**

Şimdi, RAID-4'e aynı anda 4 ve 13 numaralı bloklara (şemada $*$ ile işaretlenmiş) 2 küçük yazma yapıldığını hayal edin. Bu disklerin verileri 0 ve 1 disklerindedir ve bu nedenle okuma ve yazma verilere paralel olarak gerçekleşebilir ki bu iyidir. Ortaya çıkan sorun eşlik diski ile; her iki istek de ilgili eşlik diskini okumak zorundadır 4 ve 13 için bloklar, eşlik blokları 1 ve 3 (+ ile işaretlenmiştir). Umarım konu artık açıklığa kavuşmuştur: eşlik diski bu tür bir iş yükü altında bir engeldir; bu nedenle bazen bunu eşlik tabanlı RAID'ler için **küçük yazma sorunu (small-write problem)** olarak adlandırırız.

Böylece, veri disklerine paralel olarak erişilebilse bile, eşlik diski herhangi bir paralelliğin gerçekleşmesini engeller; tüm eşlik diski nedeniyle sisteme yazılanlar serileştirilecektir. Çünkü eşlik diskinin mantıksal disk başına iki G/Ç (bir okuma, bir yazma) gerçekleştirmesi gerekir G/Ç, RAID-4'te küçük rastgele yazma işlemlerinin performansını hesaplayabiliriz. Eşlik diskinin bu iki G/Ç üzerindeki performansını hesaplayarak (R/2) MB/s'ye ulaşırız. Rastgele küçük yazmalar altında RAID-4 verimi korkunçtur; sisteme disk ekledikçe iyileşmez.

RAID-4'teki I/O gecikmesini analiz ederek sonuca varıyoruz. Artık bildiğiniz gibi, tek bir okuma (arıza olmadığı varsayılarak) sadece tek bir diske eşlenir ve Bu nedenle gecikme süresi tek bir disk isteğinin gecikme süresine eşittir. Tek bir yazmanın gecikme süresi iki okuma ve ardından iki yazma gerektirir; okumalar, yazmalar gibi paralel olarak gerçekleşebilir ve bu nedenle toplam gecikme tek bir diskin yaklaşık iki katıdır (bazı farklılıklarla birlikte, her iki okumanın tamamlanmasını beklememiz ve böylece en kötü durum konumlandırma süresini elde etmemiz gerekir, ancak güncellemeler arama maliyetine neden olmaz ve bu nedenle ortalamadan daha iyi bir konumlandırma maliyeti olabilir).

38.7 RAID Seviye 5 : Dönen Eşlik

Küçük yazma sorununu (en azından kısmen) çözmek için Patterson, Gibson ve Katz RAID-5'i tanıttı. RAID-4, eşlik bloğunun sürücüler arasında **döndürmesi (rotates)** dışında RAID-5 neredeyse aynı şekilde çalışır (Şekil 38.7).

Disk 0	Disk 1	Disk 2	Disk 3	Disk 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

Şekil 38.7: **Döndürülmüş Eşlik ile RAID-5**

Gördüğünüz gibi, her bir şerit için eşlik bloğu artık RAID-4 için eşlik diski engelini ortadan kaldırmak amacıyla diskler döndürülüyor.

RAID-5 Analizi

RAID-5 için yapılan analizlerin çoğu RAID-4 ile aynıdır. Örneğin iki seviyenin etkin kapasitesi ve arıza toleransı aynıdır. Bu yüzden sıralı okuma ve yazma performansıdır. Tek bir isteğin gecikme süresi (ister okuma ister yazma olsun) RAID-4 ile de aynıdır.

Rastgele okuma performansı biraz daha iyidir, çünkü artık tüm diskleri kullanabiliriz. Son olarak, rastgele yazma performansı, istekler arasında paralelliğe izin verdiği için RAID-4'e göre gözle görülür şekilde iyileşir. Blok 1'e bir yazma ve blok 10'a bir yazma düşünün; bu, disk 1 ve disk 4'e (blok 1 ve paritesi için) ve disk 0 ve disk 2'ye (blok 10 ve paritesi için) isteklere dönüşecektir.

	RAID-0	RAID-1	RAID-4	RAID-5
Kapasite	N.B	(N.B)/2	(N-1).B	(N-1).B
Güvenilirlik	0	1 için N/2	1	1
Verim				
Sıralı Okuma	N.S	(N/2).S[1]	(N-1).S	(N-1).S
Sıralı Yazma	N.S	(N/2).S[1]	(N-1).S	(N-1).S
Rastgele okuma	N.R	N.R	(N-1).R	N.R
Rastgele yazma	N.R	(N/2).R	$\frac{1}{2}$. R	$\frac{N}{4}$ R
Gecikme				
Okuma	T	T	T	T
Yazma	T	T	_2T	2T

Şekil 38.8: **RAID Kapasitesi, Güvenilirlik ve Performans**

Böylece paralel olarak ilerleyebilirler. Aslında, biz genellikle çok sayıda rastgele istek verildiğinde tüm diskleri yaklaşık olarak eşit şekilde meşgul tutabilecektir. Eğer durum buysa, küçük yazmalar için toplam bant genişliğimiz N/4- R MB/s olacaktır. Faktör dört kaybın nedeni, her RAID-5 yazımının hala toplam 4 kayıp oluşturmasıdır. G/Ç işlemleri, ki bu sadece eşlik tabanlı RAID kullanmanın maliyetidir.

Çünkü RAID-5, birkaç durum dışında temelde RAID-4 ile aynıdır daha iyi olduğu yerlerde, pazarda neredeyse tamamen RAID-4'ün yerini almıştır. Bunun olmadığı tek yer, bunun olacağını bilen sistemlerdir. Hiçbir zaman büyük bir yazma işleminden başka bir şey gerçekleştirmez, böylece küçük yazma sorununu tamamen ortadan kaldırır [HLM94]; bu gibi durumlarda, RAID-4 bazen yapımı biraz daha basit olduğu için kullanılır.

38.8 RAID Karşılaştırması : Özet

Şimdi RAID seviyelerinin basitleştirilmiş karşılaştırmasını Şekil 38.8'de özetliyoruz. Basitleştirmek için analizde bazı ayrıntıları atladığımızı unutmayın. Örneğin, yansıtılmış bir sistemde yazarken, ortalama arama süresi tek bir diske yazarken olduğundan biraz daha yüksektir, çünkü arama süresi en fazla iki aramadır (her diskte bir tane). Bu nedenle, iki diske rastgele yazma performansı genellikle tek bir diskin rastgele yazma performansından biraz daha az olacaktır. Ayrıca, eşlik diskini güncellerken RAID-4/5'te, eski paritenin ilk okunması büyük olasılıkla tam aramaya neden olacaktır ve rotasyona neden olur, ancak paritenin ikinci yazımı sadece rotasyona neden olur. Son olarak, yansıtılmış RAID'lere sıralı I/O, diğer yaklaşımlara kıyasla 2 kat performans cezası öder.

[1] 1/2 cezası, yansıtma için saf bir okuma/yazma modeli varsayar; her yansıtmanın farklı bölümlerine büyük G/Ç istekleri gönderen daha karmaşık bir yaklaşım potansiyel olarak tam bant genişliğine ulaşabilir. Nedenini çözüp çözemeyeceğinizi görmek için bunu düşünün.

Bununla birlikte, Şekil 38.8'deki karşılaştırma temel farklılıkları yakalar ve RAID düzeyleri arasındaki ödünleşimleri anlamak için yararlıdır. Gecikme analizi için, tek bir diske yapılan bir isteğin alacağı süreyi temsil etmek için T'yi kullanırız.

Sonuç olarak, kesinlikle performans istiyorsanız ve umursamıyorsanız güvenilirlik, şeritleme kesinlikle en iyisidir. Ancak, rastgele G/Ç performansı ve güvenilirliği istiyorsanız, ödediğiniz maliyet kapasite kaybından yansıtma en iyisidir; Kapasite ve güvenilirlik ana hedeflerinizse, RAID5 kazanır; ödediğiniz bedel küçük yazma performansıdır. Sonunda, Her zaman sıralı G/Ç yapıyorsanız ve kapasiteyi en üst düzeye çıkarmak istiyorsanız, RAID-5 de en mantıklısıdır.

38.9 Diğer İlginç RAID Sorunları

RAID hakkında düşünürken tartışılabilecek (ve belki de tartışılması gereken) bir dizi başka ilginç fikir vardır. İşte sonunda bazı şeylerin hakkında yazabiliriz.

Örneğin, orijinal taksonomiden Düzey 2 ve 3 ve birden çok diski tolere etmek için Düzey 6 dahil olmak üzere birçok başka RAID tasarımı vardır [C+04]. Bir disk arızalandığında RAID'in yaptığı da vardır; bazen arızalı diski doldurmak için bekleyen **etkin bir yedeğe (hot spare)** sahiptir. Arıza sırasındaki performansa ve arızalı diskin yeniden yapılandırılması sırasındaki performansa ne olur? Daha gerçekçi fay modelleri de vardır, **gizli sektör hatalarını (latent sector errors)** veya **blok bozulmalarını (block corruption)** hesaba katmak için [B+08] Bu tür hataların üstesinden gelmek için çok sayıda teknik vardır (veri bütünlüğü bölümü için bkz. detaylar). Son olarak, RAID'i bir yazılım katmanı olarak bile oluşturabilirsiniz: bu tür **yazılım RAID (software RAID)** sistemleri daha ucuzdur, ancak başka sorunları vardır; tutarlı güncelleme sorunu [DAA05].

38.10 Özet

RAID'den bahsetmiştik. RAID, bir dizi bağımsız diskleri büyük, daha kapasitif ve daha güvenilir tek bir varlık haline getirir; daha da önemlisi, bunu şeffaf bir şekilde yapar ve bu nedenle yukarıdaki donanım ve yazılım değişimden nispeten habersizdi.

Aralarından seçim yapabileceğiniz birçok olası RAID seviyesi vardır ve tam olarak kullanılacak RAID seviyesi büyük ölçüde son kullanıcı için neyin önemli olduğuna bağlıdır. Örneğin, yansıtılmış RAID basit, güvenilirdir ve genellikle iyi performans ancak yüksek kapasite maliyetiyle sağlar. Buna karşın RAID-5, kapasite açısından güvenilir ve daha iyidir, ancak iş yükünde küçük yazmalar olduğunda oldukça düşük performans gösterir. Bir RAID seçme ve ayarlama parametrelerini (yığın boyutu, disk sayısı, vb.) belirli bir disk için uygun şekilde iş yükü zordur ve bilimden çok bir sanat olmaya devam etmektedir.

References

[B+08] "An Analysis of Data Corruption in the Storage Stack" by Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. FAST '08, San Jose, CA, February 2008. *Our own work analyzing how often disks actually corrupt your data. Not often, but sometimes! And thus something a reliable storage system must consider.*

[BJ88] "Disk Shadowing" by D. Bitton and J. Gray. VLDB 1988. *One of the first papers to discuss mirroring, therein called "shadowing".*

[CL95] "Striping in a RAID level 5 disk array" by Peter M. Chen and Edward K. Lee. SIGMETRICS 1995. *A nice analysis of some of the important parameters in a RAID-5 disk array.*

[C+04] "Row-Diagonal Parity for Double Disk Failure Correction" by P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, S. Sankar. FAST '04, February 2004. *Though not the first paper on a RAID system with two disks for parity, it is a recent and highly-understandable version of said idea. Read it to learn more.*

[DAA05] "Journal-guided Resynchronization for Software RAID" by Timothy E. Denehy, A. Arpaci-Dusseau, R. Arpaci-Dusseau. FAST 2005. *Our own work on the consistent-update problem. Here we solve it for Software RAID by integrating the journaling machinery of the file system above with the software RAID beneath it.*

[HLM94] "File System Design for an NFS File Server Appliance" by Dave Hitz, James Lau, Michael Malcolm. USENIX Winter 1994, San Francisco, California, 1994. *The sparse paper introducing a landmark product in storage, the write-anywhere file layout or WAFL file system that underlies the NetApp file server.*

[K86] "Synchronized Disk Interleaving" by M.Y. Kim. IEEE Transactions on Computers, Volume C-35: 11, November 1986. *Some of the earliest work on RAID is found here.*

[K88] "Small Disk Arrays – The Emerging Approach to High Performance" by F. Kurzweil. Presentation at Spring COMPCON '88, March 1, 1988, San Francisco, California. *Another early RAID reference.*

[P+88] "Redundant Arrays of Inexpensive Disks" by D. Patterson, G. Gibson, R. Katz. SIGMOD 1988. *This is considered **the** RAID paper, written by famous authors Patterson, Gibson, and Katz. The paper has since won many test-of-time awards and ushered in the RAID era, including the name RAID itself!*

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

[SG86] "Disk Striping" by K. Salem, H. Garcia-Molina. IEEE International Conference on Data Engineering, 1986. *And yes, another early RAID work. There are a lot of these, which kind of came out of the woodwork when the RAID paper was published in SIGMOD.*

[S84] "Byzantine Generals in Action: Implementing Fail-Stop Processors" by F.B. Schneider. ACM Transactions on Computer Systems, 2(2):145154, May 1984. *Finally, a paper that is not about RAID! This paper is actually about how systems fail, and how to make something behave in a fail-stop manner.*

Ev Ödevi (Simülasyon)

Bu bölümde, basit bir RAID simülatörü olan raid.py tanıtılmaktadır. RAID sistemlerinin nasıl çalıştığına dair bilginizi pekiştirmek için kullanı. Ayrıntılar için README.

Sorular

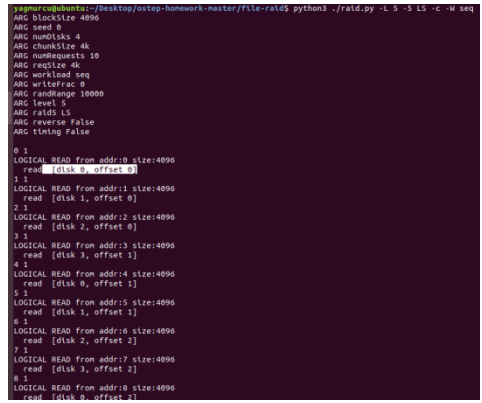
1. Bazı temel RAID eşleme testlerini gerçekleştirmek için simülatörü kullanın. Farklı seviyelerde (0, 1, 4, 5) çalıştırın ve bir dizi isteğin eşlemelerine bakalım. RAID-5 için şunları bulup bulamayacağınıza bakın : sol-simetrik ve sol-asimetrik düzenler arasındaki fark. Farklı problemler oluşturmak için bazı farklı rastgele, yukarıdakinden daha fazla kaynaklar kullanın.

python3 ./raid.py -s 2 -5 L 2 – c kodumu çalıştırıyorum ve kodun çıktısı:

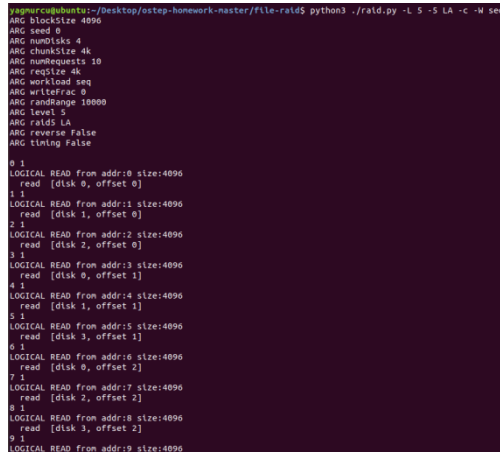
```
yagmurcu@ubuntu:~/Desktop/ostep-homework-master/file-raid$ python3 ./raid.py -s 2 -5 -L 2 -c
ARG blockSize 4096
ARG seed 2
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 10000
ARG level 0
ARG raid5 -L
ARG reverse False
ARG timing False

9560 1
LOGICAL READ from addr:9560 size:4096
  read  [disk 0, offset 2390]

565 1
LOGICAL READ from addr:565 size:4096
  read  [disk 1, offset 141]

8354 1
LOGICAL READ from addr:8354 size:4096
  read  [disk 2, offset 2088]

6697 1
LOGICAL READ from addr:6697 size:4096
  read  [disk 1, offset 1674]

6059 1
LOGICAL READ from addr:6059 size:4096
  read  [disk 3, offset 1514]

5812 1
LOGICAL READ from addr:5812 size:4096
  read  [disk 0, offset 1453]

4306 1
LOGICAL READ from addr:4306 size:4096
  read  [disk 2, offset 1076]
```

Yukarıda yaptığım kodlevel 0 RAID5 te çalıştırdığım blok boyutu 4096 'dır.
Diğer işlemlerde kodun çıktısında görüldüğü gibidir
disk = address % number_of_disks
offset = address / number_of_disks
formulden yola çıkarak

Sol simetrik bulabilmek için Level5'te RAID5 LS komutunu çalıştırdım.



Sola asimetrik bulabilmek için Level5'te RAID5 LA komutunu çalıştırdım.



Sola asimetrik düzen, Bu düzende, segmentler, şeritteki ilk eşliksiz sürücüden başlayarak sırayla numaralandırılır. Eşlik sürücüsü en soldaki sürücüden başlar ve şerit başına bir sürücü sağa hareket eder. Bu, "standart" RAID-5 düzenidir. Linux için varsayılan değildir.Sola simetrik düzen, Bu düzende, segmentler, eşlikten sonra şeritteki ilk sürücüden başlayarak sırayla numaralandırılır. Segmentler sarılır. Eşlik sürücüsü en soldaki sürücüden başlar ve şerit başına bir sürücü sağa hareket eder. Bu, Linux altındaki varsayılan RAID-5 segment düzenidir.Büyük okumalar için bu segment düzeni en hızlı olanıdır. Bunun nedeni, dizideki toplam disk sayısından daha büyük olmayan ardışık segment gruplarının dizideki tüm diskleri kullanmasıdır.
Sol asimetrik değerini bulabilmek için Level5'te RAID5 LA komutunu çalıştırdım.
Sonuç olarak:
Sol simetrik sol asimetrik değerleri bunlardır

0 1 2 P	0 1 2 P
4 5 P 3	3 4 P 5
8 P 6 7	6 P 7 8

2. İlk problemin aynısını yapın, ancak bu sefer yığın boyutunu C ile değiştirin. Yığın boyutu eşlemeleri nasıl değiştirir?

```
yagmurcu@ubuntu:~/Desktop/ostep-homework-master/file-raid$ python3 ./raid.py -s 2 -S 4k -L 2 -C 12288 -c
ARG blockSize 4096
ARG seed 2
ARG numDisks 4
ARG chunkSize 12288
ARG numRequests 10
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 10000
ARG level 0
ARG raid5 -L
ARG reverse False
ARG timing False

9560 1
LOGICAL READ from addr:9560 size:4096
  read  [disk 2, offset 2390]

565 1
LOGICAL READ from addr:565 size:4096
  read  [disk 0, offset 142]

8354 1
LOGICAL READ from addr:8354 size:4096
  read  [disk 0, offset 2090]

6697 1
LOGICAL READ from addr:6697 size:4096
  read  [disk 0, offset 1675]

6059 1
LOGICAL READ from addr:6059 size:4096
  read  [disk 3, offset 1514]

5812 1
LOGICAL READ from addr:5812 size:4096
  read  [disk 1, offset 1453]

4306 1
LOGICAL READ from addr:4306 size:4096
  read  [disk 3, offset 1075]
```

Bu sefer yığın boyutunu (stack size) C ile değiştirdim. Yığın boyutu çoğunlukla dizinin performansını etkiler. Diğer yandan, büyük bir yığın boyutu, bu tür dosya içi paralelliği azaltır ve bu nedenle yüksek performans elde etmek için birden çok eşzamanlı isteğe dayanır.

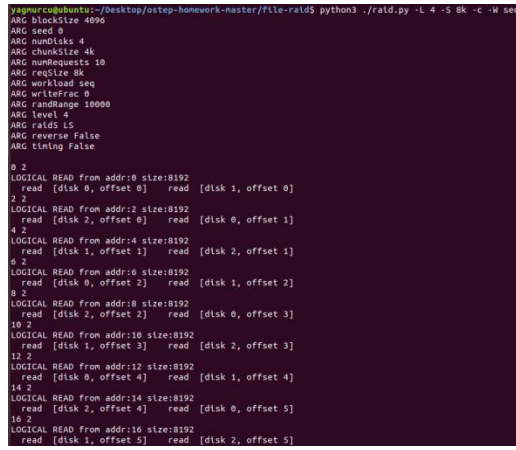
0 2 4 P
1 3 5 P
8 10 P 6
9 11 P 2

3. Yukarıdakiyle aynısını yapın, ancak her sorunun doğasını tersine çevirmek için -r bayrağını kullanın.

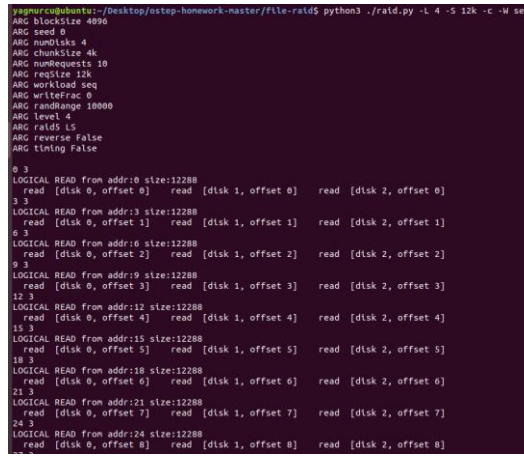
```
yagmurcu@ubuntu:~/Desktop/ostep-homework-master/file-raid$ python3 ./raid.py -L 5 -S LS -W seq -C 8K -n 12 -r
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 8K
ARG numRequests 12
ARG reqSize 4k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 5
ARG raid5 LS
ARG reverse True
ARG timing False

0 1
LOGICAL OPERATION is ?
  read  [disk 0, offset 0]
1 1
LOGICAL OPERATION is ?
  read  [disk 0, offset 1]
2 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 0]
3 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 1]
4 1
LOGICAL OPERATION is ?
  read  [disk 2, offset 0]
5 1
LOGICAL OPERATION is ?
  read  [disk 2, offset 1]
6 1
LOGICAL OPERATION is ?
  read  [disk 3, offset 2]
7 1
LOGICAL OPERATION is ?
  read  [disk 3, offset 3]
8 1
LOGICAL OPERATION is ?
  read  [disk 0, offset 2]
9 1
LOGICAL OPERATION is ?
```

4. Şimdi ters bayrağı kullanın, ancak her isteğin boyutunu -S bayrağıyla artırın. Boyutları değiştirirken 8k, 12k ve 16k olarak belirtmeyi deneyin. RAID düzeyi. Aşağıdaki durumlarda temeldeki G/Ç modeline ne olur? talebin boyutu artar mı? Sıralı iş yükü de (-W sıralı); hangi talep boyutları için RAID-4 ve RAID-5 çok daha fazla G/Ç verimlidir?

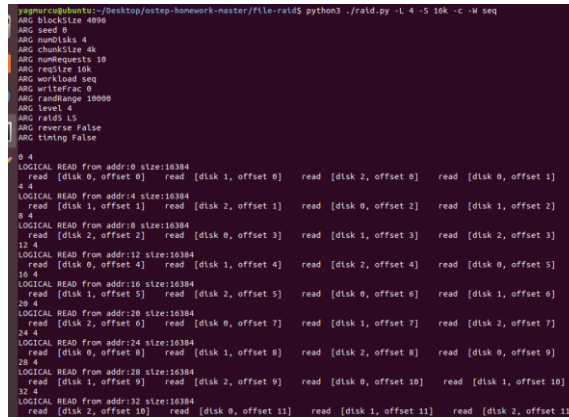
Burada Level-4'te, boyutunu 8k yaptım.



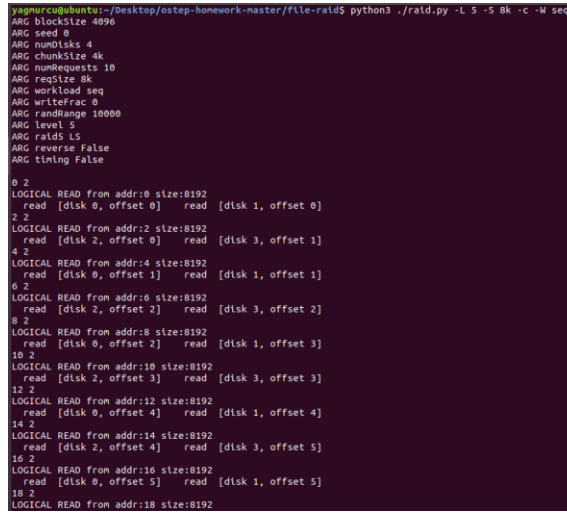
Burada Level-4'te, boyutunu 12k yaptım.



Burada Level-4'te, boyutunu 16k yaptım.

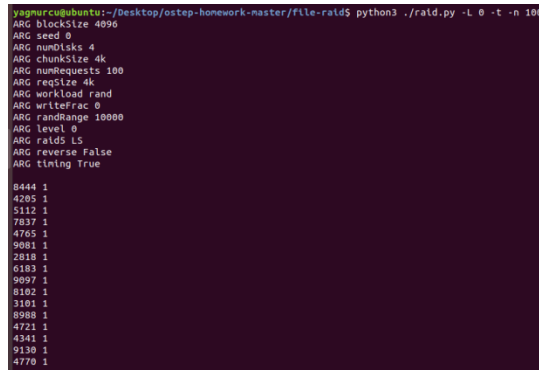


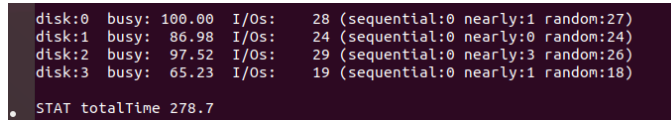
Boyuta göre istek boyutuda artar.
Burada ise istek boyutunun(reqsize) azaldığını görüyoruz. O yüzden RAID-4 daha verimlidir.

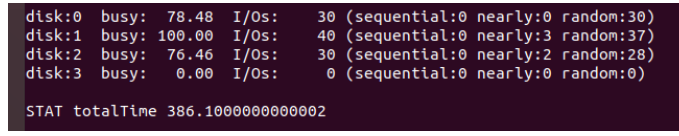


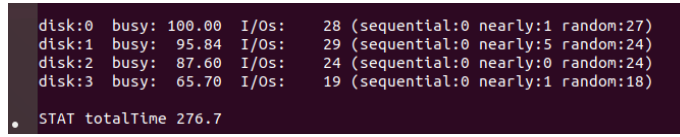
5. RAID'i değiştirirken RAID'e rastgele 100 okuma performansını tahmin etmek için, 4 disk kullanarak simülatörün zamanlama modunu (-t) seviyelerini kullanın.

python3 ./raid.py –L 0 –t –n 100 okuma modunda çalıştırdığımda kodun çıktısı aşağıdaki gibidir.









Farklı disklerde çalıştırdığımda (0,1,4,5) toplam zamanı disklere göre farkını bu şekilde görmüş olduk.

6. Yukarıdakinin aynısını yapın, ancak disk sayısını artırın. Disk sayısı arttıkça her RAID seviyesinin performansı da artar mı?

python3 ./raid.py –L 0 –n 100 –c – D 8 kodumu çalıştırdığımda çıktı:

```
disk:0  busy:  67.86  I/Os:    12 (sequential:0 nearly:3 random:9)
disk:1  busy:  63.58  I/Os:    12 (sequential:0 nearly:3 random:9)
disk:2  busy:  75.46  I/Os:    13 (sequential:0 nearly:3 random:10)
disk:3  busy:  33.35  I/Os:     6 (sequential:0 nearly:1 random:5)
disk:4  busy:  95.65  I/Os:    16 (sequential:0 nearly:2 random:14)
disk:5  busy: 100.00  I/Os:    17 (sequential:0 nearly:3 random:14)
disk:6  busy:  70.03  I/Os:    11 (sequential:0 nearly:1 random:10)
disk:7  busy:  77.44  I/Os:    13 (sequential:0 nearly:1 random:12)

STAT totalTime 156.49999999999994
```

Python3 ./raid.py –L 1 –n 100 –c – D 8 kodumu çalıştırdığımda çıktı:

```
disk:0  busy:  67.76  I/Os:    12 (sequential:0 nearly:1 random:11)
disk:1  busy:  92.07  I/Os:    16 (sequential:0 nearly:1 random:15)
disk:2  busy:  64.36  I/Os:    12 (sequential:0 nearly:2 random:10)
disk:3  busy: 100.00  I/Os:    17 (sequential:0 nearly:1 random:16)
disk:4  busy:  77.47  I/Os:    13 (sequential:0 nearly:1 random:12)
disk:5  busy:  66.21  I/Os:    11 (sequential:0 nearly:0 random:11)
disk:6  busy:  32.12  I/Os:     6 (sequential:0 nearly:1 random:5)
disk:7  busy:  72.23  I/Os:    13 (sequential:0 nearly:1 random:12)

STAT totalTime 167.79999999999995
```

python3 ./raid.py –L 4 –n 100 –c – D 8 kodumu çalıştırdığımda çıktı:

```
disk:0  busy:  94.00  I/Os:    17 (sequential:0 nearly:2 random:15)
disk:1  busy:  66.61  I/Os:    12 (sequential:0 nearly:2 random:10)
disk:2  busy: 100.00  I/Os:    18 (sequential:0 nearly:3 random:15)
disk:3  busy:  72.36  I/Os:    13 (sequential:0 nearly:2 random:11)
disk:4  busy:  76.73  I/Os:    13 (sequential:0 nearly:1 random:12)
disk:5  busy:  78.30  I/Os:    13 (sequential:0 nearly:1 random:12)
disk:6  busy:  83.70  I/Os:    14 (sequential:0 nearly:1 random:13)
disk:7  busy:   0.00  I/Os:     0 (sequential:0 nearly:0 random:0)

STAT totalTime 164.99999999999994
```

python3 ./raid.py –L 5 –n 100 –c – D 8 kodumu çalıştırdığımda çıktı:

```
disk:0  busy:  68.35  I/Os:    12 (sequential:0 nearly:3 random:9)
disk:1  busy:  63.49  I/Os:    12 (sequential:0 nearly:3 random:9)
disk:2  busy:  76.04  I/Os:    13 (sequential:0 nearly:3 random:10)
disk:3  busy:  33.04  I/Os:     6 (sequential:0 nearly:1 random:5)
disk:4  busy:  95.15  I/Os:    16 (sequential:0 nearly:2 random:14)
disk:5  busy: 100.00  I/Os:    17 (sequential:0 nearly:3 random:14)
disk:6  busy:  69.86  I/Os:    11 (sequential:0 nearly:1 random:10)
disk:7  busy:  76.42  I/Os:    13 (sequential:0 nearly:1 random:12)

STAT totalTime 158.59999999999997
```

RAID teknolojileri arasında en performanslı yapıdır. Performans disk sayısı ile doğru orantılıdır. Disk sayısı arttıkça performansda artar. 2 tane RAID 0 yapısının RAID 1 altında birleşmesi ile oluşur. Bu yapı için en az 4 disk gerekmektedir. Verileri tüm disklere dağıtarak okuma ve yazma işlemini gerçekleştirdiği için performansı çok yüksektir.

7. Yukarıdakiyle aynısını yapın, ancak okumalar yerine tüm yazmaları (-w 100) kullanın. Her RAID düzeyinin performansı şimdi nasıl ölçeklendiriliyor? 100 rastgele yazmanın iş yükünü tamamlamak için gereken sürenin kabaca bir tahminini yapabilir misiniz?

python3 ./raid.py –L 0 –t – n 100 –w 100 kodunu çalıştırdığımda çıktı: 100 * 10 /4

```
disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:1  busy:  93.91  I/Os:    29 (sequential:0 nearly:6 random:23)
disk:2  busy:  87.92  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:3  busy:  65.94  I/Os:    19 (sequential:0 nearly:1 random:18)

STAT totalTime 275.69999999999993
```

python3 ./raid.py –L 1 –t – n 100 –w 100 kodunu çalıştırdığımda çıktı: 100 * 10 / (4 / 2)

```
disk:0  busy: 100.00  I/Os:    52 (sequential:0 nearly:3 random:49)
disk:1  busy: 100.00  I/Os:    52 (sequential:0 nearly:3 random:49)
disk:2  busy:  92.90  I/Os:    48 (sequential:0 nearly:2 random:46)
disk:3  busy:  92.90  I/Os:    48 (sequential:0 nearly:2 random:46)

STAT totalTime 509.80000000000047
```

python3 ./raid.py –L 4 1 –t – n 100 –w 100 kodunu çalıştırdığımda çıktı

```
disk:0  busy:  30.84  I/Os:    60 (sequential:0 nearly:30 random:30)
disk:1  busy:  39.30  I/Os:    80 (sequential:0 nearly:43 random:37)
disk:2  busy:  30.05  I/Os:    60 (sequential:0 nearly:32 random:28)
disk:3  busy: 100.00  I/Os:   200 (sequential:0 nearly:107 random:93)

STAT totalTime 982.5000000000013
```

python3 ./raid.py –L 5 –t – n 100 –w 100 kodunu çalıştırdığımda çıktı:

```
disk:0  busy:  99.32  I/Os:   100 (sequential:0 nearly:53 random:47)
disk:1  busy:  96.02  I/Os:   100 (sequential:0 nearly:55 random:45)
disk:2  busy:  99.62  I/Os:   100 (sequential:0 nearly:52 random:48)
disk:3  busy: 100.00  I/Os:   100 (sequential:0 nearly:53 random:47)

STAT totalTime 497.40000000000043
```

python3 ./raid.py -L 0 -t -n 100 -c -D 8 -w 100 kodunu çalıştırdığımda çıktı: 156,5 = 1,76 100 * 10 / 8 tahinlerini yapabiliriz.

```
disk:0  busy:  67.86  I/Os:    12 (sequential:0 nearly:3 random:9)
disk:1  busy:  63.58  I/Os:    12 (sequential:0 nearly:3 random:9)
disk:2  busy:  75.46  I/Os:    13 (sequential:0 nearly:3 random:10)
disk:3  busy:  33.35  I/Os:     6 (sequential:0 nearly:1 random:5)
disk:4  busy:  95.65  I/Os:    16 (sequential:0 nearly:2 random:14)
disk:5  busy: 100.00  I/Os:    17 (sequential:0 nearly:3 random:14)
disk:6  busy:  70.03  I/Os:    11 (sequential:0 nearly:1 random:10)
disk:7  busy:  77.44  I/Os:    13 (sequential:0 nearly:1 random:12)

STAT totalTime 156.49999999999994
```

python3 ./raid.py -L 1 -t -n 100 -c -D 8 -w 100 kodunu çalıştırdığımda çıktı:
275,7 = 1,85 100 * 10 / (8 / 2)

```
disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:1  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:2  busy:  93.91  I/Os:    29 (sequential:0 nearly:6 random:23)
disk:3  busy:  93.91  I/Os:    29 (sequential:0 nearly:6 random:23)
disk:4  busy:  87.92  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:5  busy:  87.92  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:6  busy:  65.94  I/Os:    19 (sequential:0 nearly:1 random:18)
disk:7  busy:  65.94  I/Os:    19 (sequential:0 nearly:1 random:18)

STAT totalTime 275.69999999999993
```

python3 ./raid.py -L 4 -t -n 100 -c -D 8 -w 100 kodunu çalıştırdığımda çıktı:
937,8 = 1,05

```
disk:0  busy:  16.54  I/Os:    34 (sequential:0 nearly:19 random:15)
disk:1  busy:  11.72  I/Os:    24 (sequential:0 nearly:14 random:10)
disk:2  busy:  17.59  I/Os:    36 (sequential:0 nearly:21 random:15)
disk:3  busy:  12.73  I/Os:    26 (sequential:0 nearly:15 random:11)
disk:4  busy:  13.50  I/Os:    26 (sequential:0 nearly:14 random:12)
disk:5  busy:  13.78  I/Os:    26 (sequential:0 nearly:14 random:12)
disk:6  busy:  14.73  I/Os:    28 (sequential:0 nearly:15 random:13)
disk:7  busy: 100.00  I/Os:   200 (sequential:0 nearly:113 random:87)

STAT totalTime 937.8000000000014
```

python3 ./raid.py -L 5 -t -n 100 -c -D 8 -w 100 kodunu çalıştırdığımda çıktı:
290,9 = 1,71

```
disk:0  busy:  87.90  I/Os:    56 (sequential:0 nearly:33 random:23)
disk:1  busy:  58.95  I/Os:    40 (sequential:0 nearly:26 random:14)
disk:2  busy:  63.05  I/Os:    40 (sequential:0 nearly:23 random:17)
disk:3  busy:  72.91  I/Os:    42 (sequential:0 nearly:21 random:21)
disk:4  busy:  99.66  I/Os:    64 (sequential:0 nearly:37 random:27)
disk:5  busy:  85.60  I/Os:    54 (sequential:0 nearly:33 random:21)
disk:6  busy:  69.44  I/Os:    44 (sequential:0 nearly:26 random:18)
disk:7  busy: 100.00  I/Os:    60 (sequential:1 nearly:31 random:28)

STAT totalTime 290.9
```

8. Zamanlama modunu son bir kez çalıştırın, ancak bu sefer sıralı bir iş yüküyle (-W sıralı). RAID düzeyine göre ve okuma ve yazma işlemleri yaparken performans nasıl değişir? Her isteğin boyutunu değiştirirken ne dersiniz? RAID-4 veya RAID-5 kullanırken RAID'e hangi boyutta yazmalısınız?

python3 ./raid.py -L 0 -t -n 100 -c -w 100 -W seq // 275.7 / 12.5 = 22

```
97 1
98 1
99 1

disk:0  busy: 100.00  I/Os:    25 (sequential:24 nearly:0 random:1)
disk:1  busy: 100.00  I/Os:    25 (sequential:24 nearly:0 random:1)
disk:2  busy: 100.00  I/Os:    25 (sequential:24 nearly:0 random:1)
disk:3  busy: 100.00  I/Os:    25 (sequential:24 nearly:0 random:1)

STAT totalTime 12.499999999999991
```

python3 ./raid.py -L 1 -t -n 100 -c -w 100 -W seq // 509.8 / 15 = 34

```
disk:0  busy: 100.00  I/Os:    50 (sequential:49 nearly:0 random:1)
disk:1  busy: 100.00  I/Os:    50 (sequential:49 nearly:0 random:1)
disk:2  busy: 100.00  I/Os:    50 (sequential:49 nearly:0 random:1)
disk:3  busy: 100.00  I/Os:    50 (sequential:49 nearly:0 random:1)

STAT totalTime 14.999999999999982
```

python3 ./raid.py -L 4 -t -n 100 -c -w 100 -W seq // 982.5 / 13.4 = 73

```
99 1

disk:0  busy: 100.00  I/Os:    68 (sequential:33 nearly:34 random:1)
disk:1  busy:  99.25  I/Os:    66 (sequential:32 nearly:33 random:1)
disk:2  busy:  99.25  I/Os:    66 (sequential:32 nearly:33 random:1)
disk:3  busy: 100.00  I/Os:   200 (sequential:33 nearly:166 random:1)

STAT totalTime 13.399999999999988
```

python3 ./raid.py -L 5 -t -n 100 -c -w 100 -W seq // 497.4 / 13.4 = 37

```
98 1
99 1

disk:0  busy:  99.25  I/Os:    98 (sequential:32 nearly:65 random:1)
disk:1  busy:  99.25  I/Os:    98 (sequential:32 nearly:65 random:1)
disk:2  busy: 100.00  I/Os:   100 (sequential:33 nearly:66 random:1)
disk:3  busy: 100.00  I/Os:   104 (sequential:33 nearly:70 random:1)

STAT totalTime 13.399999999999988
```

Sıralı okuma performansı tüm diskleri kullanabilir eşlik diski hariç ve böylece etkin bir tepe bant genişliği sağlar $(N - 1) \cdot S$ MB/s (kolay bir durum). Sıralı bir okumanın daha iyisini yapabileceği düşünülebilir, çünkü verinin ikisini birden değil sadece bir kopyasını okuması gerekir.

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

Tek bir yazmanın gecikme süresi iki okuma ve ardından iki yazma gerektirir; okumalar, yazmalar gibi paralel olarak gerçekleşebilir ve bu nedenle toplam gecikme tek bir diskin yaklaşık iki katıdır (bazı farklılıklarla birlikte, her iki okumanın tamamlanmasını beklememiz ve böylece en kötü durum konumlandırma süresini elde etmemiz gerekir, ancak güncellemeler arama maliyetine neden olmaz ve bu nedenle ortalamadan daha iyi bir konumlandırma maliyeti olabilir).