38

Ucuz Disklerin Yedekli Dizileri (RAIDs)

Bir disk kullandığımızda, bazen daha hızlı olmasını isteriz; G/Ç işlemleri yavaştır ve bu nedenle tüm sistem için darboğaz olabilir. Bir disk kullandığımızda, bazen daha büyük olmasını isteriz; giderek daha fazla veri çevrimiçi hale getiriliyor ve böylece disklerimiz daha da doluyor. Bir disk kullandığımızda, bazen daha güvenilir olmasını isteriz; Bir disk arızalandığında, verilerimiz yedeklenmezse, tüm bu değerli veriler kaybolur.

PÜF NOKTA: BÜYÜK, HIZLI, GÜVENİLİR BİR DİSK NASIL YAPILIR

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

Bu bölümde, daha hızlı, daha büyük ve daha güvenilir bir disk sistemi oluşturmak için birden fazla diski birlikte kullanma tekniği olan RAID [P + 88] olarak bilinen **Ucuz Disklerin Yedek Dizisi** (**Redundant Array of Inexpensive Disks**)'ni tanıtıyoruz. Terim, 1980'lerin sonlarında U.C. Berkeley'deki bir grup araştırmacı tarafından tanıtıldı (Profesör David Patterson ve Randy Katz ve daha sonra öğrenci Garth Gibson liderliğinde); bu sıralarda birçok farklı araştırmacı aynı anda daha iyi bir depolama sistemi oluşturmak için birden fazla disk kullanma temel fikrine ulaştı [BG88, K86, K88, PB86, SG86].

Harici olarak, RAID bir diske benzer: birinin okuyabileceği veya yazabileceği bir grup blok. Dahili olarak, RAID birden fazla disk, bellek (hem geçici hem de uçucu olmayan) ve sistemi yönetmek için bir veya daha fazla işlemciden oluşan karmaşık bir canavardır. Bir donanım RAID'i bir bilgisayar sistemine çok benzer, bir grup diski yönetme görevi için uzmanlaşmıştır.

RAID'ler, tekli disklere göre çok sayıda avantaj sunar. Bu avantajlardan biri de performanstır. Birden fazla diski paralel olarak kullanmak, G/Ç sürelerini büyük ölçüde hızlandırabilir. Diğer bir faydası ise kapasitedir. Büyük veri kümeleri büyük diskler gerektirir. Son olarak, RAID'ler güvenilirliği artırabilir; verileri birden fazla diske yaymak (RAID teknikleri olmadan), verileri tek bir diskin kaybına karşı savunmasız hale getirir; bir tür **yedeklilik** (**redundancy**) ile, RAID'ler bir diskin kaybını tolere edebilir ve hiçbir şey yanlış değilmiş gibi çalışmaya devam edebilir.

IPUCU: ŞEFFAFLIK DAĞITIMI MÜMKÜN KILAR

Bir sisteme nasıl yeni işlevsellik ekleneceğini düşünürken, bu işlevselliğin **şeffaf**(**transparently**) bir şekilde, sistemin geri kalanında herhangi bir değişiklik gerektirmeyecek şekilde eklenip eklenemeyeceği her zaman göz önünde bulundurulmalıdır. Mevcut yazılımın (veya radikal donanım değişikliklerinin) tamamen yeniden yazılmasını gerektirmek, bir fikrin etki olasılığını azaltır. RAID mükemmel bir örnektir ve kesinlikle şeffaflığı başarısına katkıda bulunmuştur; yöneticiler SCSI diski yerine SCSI tabanlı RAID depolama dizisi takabiliyordu ve sistemin geri kalanının (ana bilgisayar, işletim sistemi vb.) kullanmaya başlamak için bir bit değiştirmesi gerekmiyordu. Bu **dağıtım**(**deployment**) sorununu çözerek, RAID ilk günden itibaren daha başarılı hale getirildi.

Şaşırtıcı bir şekilde, RAID'ler bu avantajları onları kullanan sistemlere **şeffaf**(**transparently)** bir şekilde sağlar, yani bir RAID ana bilgisayar sistemine büyük bir disk gibi görünür. Şeffaflığın güzelliği, elbette, bir diskin RAID ile değiştirilmesini ve tek bir yazılım satırını değiştirmemesini sağlamasıdır; çalışan sistem ve istemci uygulamaları değiştirilmeden çalışmaya devam eder. Bu şekilde, saydamlık, RAID'in **dağıtılabilirliğini**(**deployability)** büyük ölçüde artırarak, kullanıcıların ve yöneticilerin yazılım uyumluluğu endişesi olmadan bir RAID kullanmalarını sağlar.

Şimdi RAID'lerin bazı önemli yönlerini tartışıyoruz. Arayüz, hata modeli ile başlıyoruz ve ardından RAID tasarımının üç önemli eksen boyunca nasıl değerlendirilebileceğini tartışıyoruz: kapasite, güvenilirlik ve performans. Daha sonra RAID tasarımı ve uygulaması için önemli olan bir dizi başka konuyu tartışıyoruz.

38.1 Arayüz ve RAID İç Yapıları

Yukarıdaki dosya sistemi için RAID, büyük, hızlı ve güvenilir bir disk gibi görünür. Tıpkı tek bir diskte olduğu gibi, kendisini her biri dosya sistemi (veya başka bir istemci) tarafından okunabilen veya yazılabilen doğrusal bir blok dizisi olarak sunar.

Bir dosya sistemi RAID'e mantıksal bir G/Ç isteği gönderdiğinde, RAID dahili olarak isteği tamamlamak için hangi diske (veya disklere) erişileceğini hesaplamalı ve bunu yapmak için bir veya daha fazla fiziksel G/Ç vermelidir. Bu fiziksel G/Ç'lerin tam doğası, aşağıda ayrıntılı olarak tartışacağımız gibi RAID seviyesine bağlıdır. Bununla birlikte, basit bir örnek olarak, her bloğun iki kopyasını (her biri ayrı bir diskte) tutan bir RAID düşünün; Böyle bir **yansıtılmış**(**mirrored**) RAID sistemine yazarken, RAID'in verildiği her mantıksal G/Ç için iki fiziksel G/Ç gerçekleştirmesi gerekir.

RAID sistemi genellikle bir ana bilgisayara standart bir bağlantı (SCSI, SATA vb.) ile ayrı bir donanım kutusu olarak oluşturulur. Ancak dahili olarak, RAID'ler oldukça karmaşıktır, RAID'in çalışmasını yönlendirmek için üretici yazılımını çalıştıran bir mikrodenetleyiciden oluşan, veri bloklarını okundukça ve

yazıldıkça arabelleğe almak için DRAM gibi geçici bellek (bazı durumlarda arabelleğe almak için geçici olmayan bellek) güvenli bir şekilde ve hatta belki de eşlik hesaplamaları yapmak için özel mantık yazar (aşağıda da göreceğimiz gibi, bazı RAID düzeylerinde kullanışlıdır). Yüksek düzeyde, RAID çok özel bir bilgisayar sistemidir: bir işlemciye, belleğe ve disklere sahiptir; ancak, uygulamaları çalıştırmak yerine, RAID'i çalıştırmak için tasarlanmış özel bir yazılım çalıştırır.

38.2 Hata Modeli

RAID'i anlamak ve farklı yaklaşımları karşılaştırmak için aklımızda bir hata modeli olmalıdır. RAID'ler, belirli disk hatası türlerini algılamak ve bunlardan kurtulmak için tasarlanmıştır; Bu nedenle, tam olarak hangi hataların bekleneceğini bilmek, çalışan bir tasarıma ulaşmada kritik öneme sahiptir.

Varsayacağımız ilk arıza modeli oldukça basittir ve **arıza-durdurma**(**fail-stop**) arıza modeli [S84] olarak adlandırılmıştır. Bu modelde, bir disk tam olarak iki durumdan birinde olabilir: çalışıyor veya başarısız oluyor. Çalışan bir diskle, tüm bloklar okunabilir veya yazılabilir. Buna karşılık, bir disk arızalandığında, kalıcı olarak kaybolduğunu varsayıyoruz.

Arıza durdurma modelinin kritik bir yönü, arıza algılama konusunda varsaydığı şeydir. Özellikle, bir disk arızalandığında, bunun kolayca algılandığını varsayıyoruz. Örneğin, bir RAID dizisinde, RAID denetleyici donanımının (veya yazılımının) bir diskin arızalandığını hemen gözlemleyebileceğini varsayıyoruz.

Bu nedenle, şimdilik, disk bozulması gibi daha karmaşık "sessiz" arızalar hakkında endişelenmemize gerek yok. Ayrıca, başka türlü çalışan bir diskte (bazen gizli sektör hatası olarak adlandırılır) tek bir bloğun erişilemez hale gelmesi konusunda endişelenmemize gerek yoktur. Bu daha karmaşık (ve ne yazık ki daha gerçekçi) disk hatalarını daha sonra ele alacağız.

38.3 RAID Nasıl Değerlendirilir

Yakında göreceğimiz gibi, bir RAID oluşturmak için bir dizi farklı yaklaşım var. Bu yaklaşımların her biri, güçlü ve zayıf yönlerini anlamak için değerlendirmeye değer farklı özelliklere sahiptir.

Özellikle, her RAID tasarımını üç eksen boyunca değerlendireceğiz. İlk eksen **kapasitedir**(**capacity**); Her biri B bloklu bir dizi N disk verildiğinde, RAID istemcileri için ne kadar yararlı kapasite mevcuttur? Yedeklilik olmadan, cevap $N \cdot B$'dir; Buna karşılık, her bloğun iki kopyasını tutan (**yansıtma**(**mirroring**) adı verilen) bir sistemimiz varsa, $(N \cdot B)/2$'lik bir yararlı kapasite elde ederiz. Farklı şemalar (örneğin, parite tabanlı olanlar) ikisinin arasına girme eğilimindedir.

Değerlendirmenin ikinci ekseni **güvenilirliktir** (**reliability**). Verilen tasarım kaç disk hatasını tolere edebilir? Hata modelimize uygun olarak, yalnızca tüm bir diskin arızalanabileceğini varsayıyoruz; sonraki bölümlerde (yani veri bütünlüğü üzerine), daha karmaşık hata modlarının nasıl ele alınacağını düşüneceğiz.

Son olarak, üçüncü eksen **performanstır**(**performance**). Performansın değerlendirilmesi biraz zordur, çünkü büyük ölçüde disk dizisine sunulan iş yüküne bağlıdır. Bu nedenle, performansı değerlendirmeden önce, öncelikle göz önünde bulundurulması gereken bir dizi tipik iş yükü sunacağız.

Artık üç önemli RAID tasarımını göz önünde bulunduruyoruz: RAID Düzey 0 (şeritleme), RAID Düzey 1 (yansıtma) ve RAID Düzey 4/5 (eşlik tabanlı yedeklilik). Bu tasarımların her birinin "seviye" olarak adlandırılması, Patterson, Gibson ve Katz'ın Berkeley'deki öncü çalışmalarından kaynaklanmaktadır [P + 88].

38.4 RAID Düzey 0: Şeritleme

İlk RAID seviyesi aslında bir RAID seviyesi değildir, çünkü artıklık yoktur. Bununla birlikte, RAID seviye 0 veya daha iyi bilindiği şekliyle **şeritleme**(**striping**), performans ve kapasite açısından mükemmel bir üst sınır görevi görür ve bu nedenle anlaşılmaya değerdir.

Şeritlemenin en basit biçimi, blokları sistemin diskleri boyunca şu şekilde **şeritleyecektir**(**stripe**) (burada 4 diskli bir dizi varsayın):

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
34	5	6	7
8	9	10	11
12	13	14	15

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

Şekil 38.1'den temel fikri elde edersiniz: dizinin bloklarını diskler boyunca dairesel bir şekilde dağıtın. Bu yaklaşım, dizinin bitişik parçaları için istek yapıldığında (örneğin, büyük, sıralı bir okumada olduğu gibi) diziden en fazla paralelliği çıkarmak için tasarlanmıştır. Aynı sıradaki bloklara şerit diyoruz; bu nedenle, 0, 1, 2 ve 3 blokları yukarıdaki aynı şerittedir.

Örnekte, bir sonrakine geçmeden önce her diske yalnızca 1 bloğun (her biri 4KB boyutunda) yerleştirildiği basitleştirici varsayımını yaptık. Ancak bu düzenleme böyle olmak zorunda değildir. Örneğin, Şekil 38.2'deki gibi blokları diskler arasında düzenleyebiliriz:

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

Şekil 38.2: **Daha Büyük Bir Yığın Boyutuyla Şeritleme**

Bu örnekte, bir sonraki diske geçmeden önce her diske iki adet 4KB blok yerleştirdik. Böylece, bu RAID dizisinin öbek boyutu 8 KB'dir ve bu nedenle bir şerit 4 parçadan veya 32 KB veriden oluşur.

BIR KENARA: RAID EŞLEME SORUNU

RAID'in kapasitesini, güvenilirliğini ve performans özelliklerini incelemeden önce **haritalama problemi** (**the mapping problem)** dediğimiz şeye bir kenara koyuyoruz. Bu sorun tüm RAID dizilerinde ortaya çıkar; Basitçe söylemek gerekirse, okumak veya yazmak için mantıksal bir blok göz önüne alındığında, RAID tam olarak hangi fiziksel diske ve ofsete erişeceğini nasıl bilir?
Bu basit RAID seviyeleri için, mantıksal blokları fiziksel konumlarına doğru bir şekilde eşlemek için fazla karmaşıklığa ihtiyacımız yok. Yukarıdaki ilk şeritleme örneğini ele alalım (öbek boyutu = 1 blok = 4KB). Bu durumda, mantıksal bir blok adresi A verildiğinde, RAID istenen diski kolayca hesaplayabilir ve iki basit denklemle ofset yapabilir:

```
Disk   = A % disk sayısı
Ofset = A / disk sayısı
```

Bunların hepsinin tamsayı işlemleri olduğuna dikkat edin (ör. 4 / 3 = 1, 1.33333 değil...). Basit bir örnek için bu denklemlerin nasıl çalıştığını görelim. Yukarıdaki ilk RAID'de blok 14 için bir istek geldiğini hayal edin. 4 disk olduğu göz önüne alındığında, bu, ilgilendiğimiz diskin (%14 4 = 2): disk 2 olduğu anlamına gelir. Tam blok şu şekilde hesaplanır: (14 / 4 = 3): blok 3. Bu nedenle, blok 14, tam olarak bulunduğu yer olan üçüncü diskin dördüncü bloğunda (blok 3, 0'dan başlayan) (disk 2, 0'dan başlayan) bulunmalıdır. Bu denklemlerin farklı parça boyutlarını destekleyecek şekilde nasıl değiştirileceğini düşünebilirsiniz. Deneyin! Çok zor değil.

Yığın Boyutları

Yığın boyutu çoğunlukla dizinin performansını etkiler. Örneğin, küçük bir yığın boyutu, birçok dosyanın birçok diskte şeritleneceği anlamına gelir, böylece tek bir dosyaya okuma ve yazma paralelliğini artırır; ancak, birden çok diskteki bloklara erişmek için konumlandırma süresi artar, çünkü tüm istek için konumlandırma süresi, isteklerin tüm sürücülerdeki konumlandırma sürelerinin maksimumuna göre belirlenir.

Öte yandan, büyük bir yığın boyutu, bu tür dosya içi paralelliği azaltır ve böylece yüksek verim elde etmek için birden çok eşzamanlı isteğe dayanır. Bununla birlikte, büyük yığın boyutları konumlandırma süresini azaltır; Örneğin, tek bir dosya bir yığının içine sığarsa ve bu nedenle tek bir diske yerleştirilirse, erişim sırasında ortaya çıkan konumlandırma süresi yalnızca tek bir diskin konumlandırma süresi olacaktır.

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

RAID-0 Analizine Geri Dön

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

RAID Performansını Değerlendirme

RAID performansını analiz ederken, iki farklı performans metriği göz önünde bulundurulabilir. İlki, tek istek gecikmesidir. Tek bir G/Ç isteğinin RAID'e gecikmesini anlamak, tek bir mantıksal G/Ç işlemi sırasında ne kadar paralelliğin var olabileceğini ortaya koyduğu için yararlıdır. İkincisi, RAID'in sabit durum verimidir, yani birçok eşzamanlı isteğin toplam bant genişliğidir. RAID'ler genellikle yüksek performanslı ortamlarda kullanıldığından, kararlı durum bant genişliği kritiktir ve bu nedenle analizlerimizin ana odağı olacaktır.

Aktarım hızını daha ayrıntılı olarak anlamak için, ilgilendiğimiz bazı iş yüklerini ortaya koymamız gerekir. Bu tartışma için iki tür iş yükü olduğunu varsayacağız: **sıralı**(**sequential**) ve **rastgele**(**random**). Sıralı bir iş yüküyle, diziye yapılan isteklerin büyük bitişik parçalar halinde geldiğini varsayıyoruz; örneğin, x bloğundan başlayıp blokta (x+1 MB) biten 1 MB veriye erişen bir istek (veya bir dizi istek) sıralı olarak kabul edilir. Sıralı iş yükleri birçok ortamda yaygındır (bir anahtar kelime için büyük bir dosyada arama yapmayı düşünün) ve bu nedenle önemli kabul edilir.

Rastgele iş yükleri için, her isteğin oldukça küçük olduğunu ve her isteğin diskte farklı bir rasgele konuma olduğunu varsayıyoruz. Örneğin, rastgele bir istek akışı önce mantıksal adres 10'da, sonra mantıksal adres 550.000'de, sonra 20.100'de vb. 4KB'ye erişebilir. Veritabanı yönetim sistemindeki (DBMS) işlemsel iş yükleri gibi bazı önemli iş yükleri bu tür bir erişim modeli sergiler ve bu nedenle önemli bir iş yükü olarak kabul edilir.

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

Anlayacağınız gibi, sıralı ve rastgele iş yükleri bir diskten çok farklı performans özelliklerine neden olur. Sıralı erişimle, bir disk en verimli modunda çalışır, rotasyonu aramak ve beklemek için çok az zaman harcar ve zamanının çoğunu veri aktarır. Rastgele erişimde, bunun tam tersi geçerlidir: çoğu zaman rotasyonu aramak ve beklemek için harcanır ve veri aktarımı için nispeten az zaman harcanır. Analizimizdeki bu farkı yakalamak için, bir diskin sıralı bir iş yükü altında S MB/s'de veri aktarabileceğini ve bir diskin altında R MB/sn veri aktarabildiğini varsayacağız. rastgele iş yükü. Genel olarak, S, R'den çok daha büyüktür (yani, S ≫ R).

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

Ortalama arama süresi 7 ms

Ortalama dönme gecikmesi 3 ms

Diskin aktarım hızı 50 MB/s

S'yi hesaplamak için öncelikle tipik bir 10 MB aktarımda zamanın nasıl harcandığını bulmamız gerekir. İlk önce 7 ms arayarak, sonra 3 ms dönerek geçiririz. Son olarak transfer başlar; 50 MB/sn'de 10 MB, aktarım için saniyenin 1/5'ine veya 200 ms'ye kadar harcama yapılmasına neden olur. Böylece, her 10 MB istek için, isteği tamamlamak için 210 ms harcıyoruz. S'yi hesaplamak için sadece bölmemiz gerekiyor:

$$S = \frac{Veri\ Miktarı}{Erişim\ Zamanı} = \frac{10\ MB}{210\ ms} = 47{,}62\ \text{MB/s}$$

Gördüğümüz gibi, veri aktarımı için harcanan büyük zaman nedeniyle, S diskin tepe bant genişliğine çok yakındır (arama ve dönme maliyetleri amortismana tabi tutulmuştur).

R'yi de benzer şekilde hesaplayabiliriz. Arama ve rotasyon aynıdır; daha sonra aktarımda harcanan süreyi hesaplarız, bu da 50 MB/sn @ 10 KB veya 0,195 ms'dir.

$$R = \frac{Veri\ Miktarı}{Erişim\ Zamanı} = \frac{10\ KB}{10.195\ ms} = 0{,}981\ \text{MB/s}$$

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

RAID-0 Analizine Geri Dön, Tekrar

Şimdi şeritleme performansını değerlendirelim. Yukarıda söylediğimiz gibi, genellikle iyidir. Örneğin, gecikme perspektifinden bakıldığında, tek bloklu bir isteğin gecikmesi, tek bir diskinkiyle hemen hemen aynı olmalıdır; sonuçta, RAID-0 bu isteği disklerinden birine yönlendirecektir.

Sabit durum sıralı aktarım hızı perspektifinden, sistemin tam bant genişliğini elde etmeyi bekleriz. Böylece, aktarım hızı N'ye (disk sayısı) eşittir ve S ile çarpılır (tek bir diskin sıralı bant genişliği). Çok sayıda rastgele G/Ç için, hepsini tekrar kullanabiliriz. diskler ve böylece N elde · R MB/s. Aşağıda göreceğimiz gibi, bu değerler hem hesaplanması en basit değerlerdir hem de bir üst sınır görevi görecektir. diğer RAID düzeyleriyle karşılaştırma.

38.5 RAID Düzey 1: Yansıtma

Şeritlemenin ötesindeki ilk RAID seviyemiz RAID seviye 1 veya yansıtma olarak bilinir. Aynalı bir sistemle, sistemdeki her bloğun birden fazla kopyasını oluştururuz; elbette her kopya ayrı bir diske yerleştirilmelidir. Bunu yaparak, disk arızalarını tolere edebiliriz.

Tipik bir yansıtılmış sistemde, her mantıksal blok için RAID'in iki fiziksel kopyasını sakladığını, varsayacağız. İşte bir 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 ayna çiftleri arasında şeritlenir. Aslında, blok kopyaları disklere yerleştirmenin birkaç farklı yolu olduğunu fark etmiş olabilirsiniz. Yukarıdaki düzenleme yaygın bir düzenlemedir ve bazen RAID-10 (veya RAID 1 + 0, **ayna şeridi** (**stripe of mirrors**)) olarak adlandırılır, çünkü bunların üzerinde yansıtılmış çiftler (RAID-1) ve ardından çizgiler (RAID-0) kullanır; diğer bir yaygın düzenleme, iki büyük şeritleme (RAID-0) dizisi ve ardından bunların üzerinde ikizlemeler (RAID-1) içeren RAID-01'dir (veya RAID 0+1, aynalı şeritler). Şimdilik, yukarıdaki düzeni varsayarak yansıtma hakkında konuşacağız.

Yansıtılmış bir diziden bir blok okurken, RAID'in bir seçeneği vardır: her iki kopyayı da okuyabilir. Örneğin, RAID'e mantıksal blok 5'e bir okuma verilirse, bunu disk 2 veya disk 3'ten okumak ücretsizdir. Bununla birlikte, bir 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 3'e ilerleyebilir.

RAID-1 Analizi

RAID-1'i değerlendirelim. Kapasite açısından bakıldığında, RAID-1 pahalıdır; yansıtma seviyesi = 2 ile, en yüksek faydalı kapasitemizin sadece yarısını elde ederiz. B bloklu N disk ile RAID-1'in faydalı kapasitesi $(N \cdot B)/2$'dir.

Güvenilirlik açısından bakıldığında, RAID-1 iyi iş çıkarıyor. Herhangi bir diskin arızalanmasını tolere edebilir. RAID-1'in biraz şansla bundan daha iyisini yapabileceğini de fark edebilirsiniz. Yukarıdaki şekilde, disk 0 ve disk 2'nin her ikisinin de başarısız olduğunu hayal edin. Böyle bir durumda veri kaybı olmaz! Daha genel olarak, ikizlenmiş bir sistem (iki yansıtma düzeyiyle) 1 disk arızasını kesin olarak ve hangi disklerin arızalandığına bağlı olarak N/2 arızaya kadar tolere edebilir. Pratikte, genellikle böyle şeyleri şansa bırakmayı sevmiyoruz; bu nedenle çoğu insan yansıtmanın tek bir başarısızlığı ele almak için iyi olduğunu düşünür.

Son olarak, performansı analiz ediyoruz. Tek bir okuma isteğinin gecikmesi açısından, tek bir diskteki gecikmeyle aynı olduğunu görebiliriz; RAID-1'in tek yaptığı, okumayı kopyalarından birine yönlendirmek. Yazma biraz farklıdır: bitmeden önce tamamlanması için iki fiziksel yazma gerekir. Bu iki yazma paralel olarak gerçekleşir ve bu nedenle süre kabaca tek bir yazmanın süresine eşit olacaktır; ancak, mantıksal yazma her iki fiziksel yazmanın da tamamlanmasını beklemesi gerektiğinden, iki isteğin en kötü durum araması ve dönüş gecikmesinden muzdariptir ve bu nedenle (ortalama olarak) tek bir diske yazmaya göre biraz daha yüksek olacaktır.

BİR YANA: RAİD TUTARLI GÜNCELLEŞTİRME SORUNU

RAID-1'i analiz etmeden önce, **tutarlı güncelleme sorunu** (**consistent-update problem**) [DAA05] olarak bilinen herhangi birçok diskli RAID sisteminde ortaya çıkan bir sorunu tartışalım. Sorun, tek bir mantıksal işlem sırasında birden çok diski güncelleştirmesi gereken herhangi bir RAID'e yazma sırasında oluşur. Bu durumda, yansıtılmış bir disk dizisi düşündüğümüzü varsayalım.

Yazmanın RAID'e verildiğini ve ardından RAID'in disk 0 ve disk 1 olmak üzere iki diske yazılması gerektiğine karar verdiğini düşünün. Ardından RAID, disk 0'a yazma işlemini gerçekleştirir, ancak RAID, disk 1'e istek göndermeden hemen önce, bir güç kaybı (veya sistem çökmesi) meydana gelir. Bu talihsiz durumda, 0 diskine yapılan talebin tamamlandığını varsayalım (ancak disk 1'e yapılan talebin, hiçbir zaman yapılmadığı için açıkça yapılmadı).

Bu zamansız güç kaybının sonucu, bloğun iki kopyasının artık **tutarsız** (**inconsistent**) olmasıdır; disk 0'daki kopya yeni sürümdür ve disk 1'deki kopya eskidir. Olmasını istediğimiz şey, her iki diskin durumunun atomik olarak (**atomically**) değişmesi, yani ya her ikisi de yeni sürüm olarak bitmeli ya da hiçbiri.

Bu sorunu çözmenin genel yolu, bunu yapmadan önce RAID'in ne yapmak üzere olduğunu (yani iki diski belirli bir veri parçasıyla güncellemek) kaydetmek için bir tür **önceden yazma günlüğü** (**write-ahead log**) kullanmaktır. Bu yaklaşımı benimseyerek, bir çarpışma durumunda doğru şeyin olmasını sağlayabiliriz; bekleyen tüm işlemleri RAID'de yeniden oynatan bir **kurtarma** (**recovery**) prosedürü çalıştırarak, iki ikizlenmiş kopyanın (RAID-1 durumunda) senkronize olmamasını sağlayabiliriz.

Son bir not: Her yazma işleminde diskte oturum açmak çok pahalı olduğundan, çoğu RAID donanımı bu tür bir günlük kaydı gerçekleştirdiği yerde az miktarda geçici olmayan RAM (örn. pil destekli) içerir. Böylece, diske kaydetmenin yüksek maliyeti olmadan tutarlı güncelleme sağlanır.

Kararlı durum verimini analiz etmek için sıralı iş yüküyle başlayalım. Sırayla diske yazarken, her mantıksal yazma iki fiziksel yazmayla sonuçlanmalıdır; örneğin, mantıksal blok 0'ı yazdığımızda (yukarıdaki şekilde), RAID bunu dahili olarak hem disk 0'a hem de disk 1'e yazar. Böylece, ikizlenmiş bir diziye sıralı yazma sırasında elde edilen maksimum bant genişliğinin ($N/2 \cdot S$) veya tepe bant genişliğinin yarısı olduğu sonucuna varabiliriz.

Ne yazık ki, sıralı bir okuma sırasında tam olarak aynı performansı elde ediyoruz. Sıralı bir okumanın daha iyi yapabileceği düşünülebilir, çünkü verinin yalnızca bir kopyasını okuması gerekir, ikisini birden değil. Ancak, bunun neden pek yardımcı olmadığını göstermek için bir örnek kullanalım. 0, 1, 2, 3, 4, 5, 6 ve 7 bloklarını okumamız gerektiğini düşünün. Diyelim ki 0'ın okumasını disk 0'a, 1'in okumasını disk 2'ye, 2'nin okumasını disk 1'e veriyoruz ve 3'ün okuması disk 3'e. Sırasıyla 4, 5, 6 ve 7'nin okumalarını disk 0, 2, 1 ve 3'e vererek devam ediyoruz. Tüm diskleri kullandığımız için dizinin tam bant genişliğine ulaştığımız safça düşünülebilir.

Bununla birlikte, durumun (zorunlu olarak) böyle olmadığını görmek için, tek bir diskin (disk 0 diyelim) aldığı istekleri göz önünde bulundurun. İlk olarak, blok 0 için bir istek alır; ardından 4. blok için bir talep alır (2. blok atlanır). Aslında, her disk diğer her blok için bir istek alır. Atlanan blok üzerinde dönerken istemciye faydalı bant genişliği sağlamıyor. Bu nedenle, her disk en yüksek bant genişliğinin yalnızca yarısını sunacaktır. Ve bu nedenle, sıralı okuma yalnızca (N/2 · S) MB/s'lik bir bant genişliği elde edecektir.

Rastgele okumalar, ikizlenmiş bir RAID için en iyi durumdur. Bu durumda, biz okumaları tüm disklere dağıtabilir ve böylece mümkün olan tam bant genişliğini elde edebilir. Böylece, rastgele okumalar için RAID-1, N · R MB/s sunar.

Son olarak, rastgele yazma işlemleri beklediğiniz gibi çalışır: N/2 ·R MB/sn. Her biri mantıksal yazma, iki fiziksel yazmaya dönüşmelidir ve bu nedenle tüm diskler kullanımda olacak, istemci bunu yalnızca kullanılabilir bant genişliğinin yarısı olarak algılayacaktır. Mantıksal blok x'e yazma, iki farklı fiziksel diske iki paralel yazmaya dönüşse de, birçok küçük isteğin bant genişliği şeritlemede gördüğümüzün yalnızca yarısına ulaşır. Yakında göreceğimiz gibi, mevcut bant genişliğinin yarısını almak aslında oldukça iyi!

38.6 RAID Seviye 4: Eşlik ile Yerden Tasarruf

Şimdi, **eşlik** (**parity**) olarak bilinen bir disk dizisine artıklık eklemenin farklı bir yöntemini sunuyoruz. Eşliğe dayalı yaklaşımlar, daha az kapasite kullanmaya çalışır ve böylece aynalı sistemlerin ödediği devasa alan cezasının üstesinden gelir. Ancak bunu bir maliyet karşılığında 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 veri şeridi için, o blok şeridi için fazlalık bilgileri depolayan tek bir eşlik bloğu ekledik. Örneğin, eşlik bloğu P1, blok 4, 5, 6 ve 7'den hesapladığı fazladan bilgiye sahiptir.

Eşliği hesaplamak için, şeridimizden herhangi bir bloğun kaybına dayanmamızı sağlayan matematiksel bir fonksiyon kullanmamız gerekiyor. Görünüşe göre XOR basit işlevi oldukça iyi yapıyor. Belirli bir bit kümesi için, tüm bu bitlerin XOR'u, bitlerde çift sayıda 1 varsa 0 ve tek sayıda 1 varsa 1 döndürür. Örneğin:

İlk satırda (0,0,1,1), iki 1 (C2, C3) vardır ve bu nedenle tüm bu değerlerin XOR'u 0 (P) olacaktır; benzer şekilde, ikinci satırda yalnızca bir 1 (C1) vardır ve bu nedenle XOR, 1 (P) olmalıdır. Bunu basit bir şekilde hatırlayabilirsiniz: herhangi bir sıradaki 1'lerin sayısı, eşlik biti dahil, çift (tek değil) bir sayı olmalıdır; bu, paritenin doğru olması için RAID'in sürdürmesi gereken **değişmezdir**(**invariant**).

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

Yukarıdaki örnekten, eşlik bilgilerinin bir arızadan kurtulmak için nasıl kullanılabileceğini de tahmin edebilirsiniz. C2 etiketli sütunun kaybolduğunu hayal edin. Sütunda hangi değerlerin olması gerektiğini bulmak için, o satırdaki diğer tüm değerleri okumamız (XOR'd eşlik biti dahil) ve doğru cevabı **yeniden oluşturmamız** (**reconstruct**) gerekir. Spesifik olarak, C2 sütunundaki ilk satırın değerinin kaybolduğunu varsayalım (bu bir 1'dir); o satırdaki diğer değerleri okuyarak (C0'dan 0, C1'den 0, C3'ten 1 ve P eşlik sütunundan 0) 0, 0, 1 ve 0 değerlerini alıyoruz. her satırdaki 1'lerin çift sayısı, eksik verinin ne olması gerektiğini biliyoruz: a 1. XOR tabanlı bir parite şemasında yeniden yapılandırma bu şekilde çalışır! Yeniden yapılandırılmış değeri nasıl hesapladığımıza da dikkat edin: ilk etapta pariteyi hesapladığımız şekilde, veri bitlerini ve eşlik bitlerini birlikte XOR'larız.

Şimdi merak ediyor olabilirsiniz: tüm bu bitleri XOR'lamaktan bahsediyoruz ve yine de yukarıdan biliyoruz ki RAID her diske 4 KB (veya daha büyük) bloklar yerleştiriyor; pariteyi hesaplamak için bir grup bloğa XOR'u nasıl uygularız? Bunun da kolay olduğu ortaya çıktı. Basitçe, veri bloklarının her bir biti boyunca bit düzeyinde bir XOR gerçekleştirin; her bitsel XOR'un sonucunu parite bloğundaki karşılık gelen bit yuvasına yerleştirin. Örneğin, 4 bit boyutunda bloklarımız olsaydı (evet, bu hala 4 KB'lık bir bloktan biraz daha küçüktür, ancak resmi anladınız), şöyle görünebilirler:

Block0	Block1	Block2	Block3	Parity
00	10	11	10	11
10	01	00	01	10

Şekilden de görebileceğiniz gibi, her bloğun her biti için parite hesaplanır ve sonuç parite bloğuna yerleştirilir.

RAID-4 Analizi

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

Güvenilirliğin anlaşılması da oldukça kolaydır: RAID-4, 1 disk arızasını tolere eder ve daha fazlasını değil. Birden fazla disk kaybolursa, kaybolan verileri yeniden oluşturmanın 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 Şerit Yazma**

Son olarak, performans var. Bu kez, kararlı durum verimini analiz ederek başlayalım. Ardışık okuma performansı, eşlik diski dışındaki tüm diskleri kullanabilir ve bu nedenle (N − 1) · S MB/sn'lik (kolay bir durum) etkili tepe bant genişliği sağlar.

Sıralı yazmaların performansını anlamak için önce nasıl yapıldığını anlamalıyız. Diske büyük miktarda veri yazarken RAID-4, tam şerit yazma olarak bilinen basit bir optimizasyon gerçekleştirebilir. Örneğin, 0, 1, 2 ve 3 bloklarının bir yazma talebinin parçası olarak RAID'e gönderildiği durumu hayal edin (Şekil 38.5).

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

Tam şerit yazmayı anladıktan sonra, RAID-4'teki sıralı yazmaların performansını hesaplamak kolaydır; etkin bant genişliği de (N − 1) · S MB/sn'dir. İşlem sırasında parite diski sürekli kullanımda olsa da istemci bundan performans avantajı elde etmez.

Şimdi rastgele okumaların performansını analiz edelim. Yukarıdaki şekilde de görebileceğiniz gibi, 1 blokluk rasgele okumalar sistemin veri disklerine dağıtılacak, ancak parite diskine dağıtılmayacaktır. Böylece etkin performans: (N− 1) · R MB/s.

En sona sakladığımız rastgele yazmalar, RAID-4 için en ilginç durumu sunuyor. Yukarıdaki örnekte blok 1'in üzerine yazmak istediğimizi düşünün. Devam edip üzerine yazabilirdik ama bu bizi bir sorunla baş başa bırakırdı: parite bloğu P0 artık şeridin doğru parite değerini doğru bir şekilde yansıtmayacaktı; bu örnekte, P0 da güncellenmelidir. Hem doğru hem de verimli bir şekilde nasıl güncelleyebiliriz?

Görünüşe göre iki yöntem var. **Ek parite** (**additive parity**) olarak bilinen ilki, aşağıdakileri yapmamızı gerektirir. Yeni eşlik bloğunun değerini hesaplamak için, şeritteki diğer tüm veri bloklarını paralel olarak okuyun (örnekte, blok 0, 2 ve 3) ve yeni blokla (1) XOR. Sonuç, yeni eşlik bloğunuzdur. Yazmayı tamamlamak için, yeni verileri ve yeni pariteyi yine paralel olarak ilgili disklere yazabilirsiniz.

Bu teknikle ilgili sorun, disk sayısıyla ölçeklenmesi ve bu nedenle daha büyük RAID'lerde hesaplama eşliği için yüksek sayıda okuma gerektirmesidir. Bu nedenle, **çıkarma paritesi** (**subtractive parity**) yöntemi.

Örneğin, şu bit dizesini düşünün (4 veri biti, bir eşlik):

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

C2 bitinin üzerine C2*new* adını vereceğimiz yeni bir değer yazmak istediğimizi düşünelim. Çıkarma yöntemi üç adımda çalışır. İlk önce C2'deki eski verileri (C2old = 1) ve eski pariteyi (Pold = 0) okuyoruz. Ardından eski verilerle yeni verileri karşılaştırıyoruz; eğer aynı iseler (örneğin, C2new = C2old), o zaman parite bitinin de aynı kalacağını biliyoruz (yani, Pnew = Pold). Ancak farklılarsa, o zaman eski parite bitini mevcut durumunun tersine çevirmemiz gerekir, yani eğer (Pold == 1), Pnew 0 olarak ayarlanacaksa; eğer (Pold == 0), Pnew 1 olarak set edilecek. Tüm bu karışıklığı XOR ile düzgün bir şekilde ifade edebiliriz (burada $\oplus$, XOR operatörüdür):

$$P_{new} = (C_{old} \oplus C_{new}) \oplus P_{old} \quad (38.1)$$

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

Artık ne zaman ek parite hesaplamasını ve ne zaman çıkarma yöntemini kullanacağımızı anlayabilmelisiniz. Toplama yönteminin çıkarma yönteminden daha az G/Ç gerçekleştirmesi için sistemde kaç disk olması gerektiğini düşünün; kesişme noktası nedir?

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 RAID'e gönderilen çok sayıda yazma olduğunu hayal edin; RAID-4 paralel olarak kaç tanesini gerçekleştirebilir? Anlamak için tekrar RAID-4 düzenine 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 Parite Bloklarına Yazar**

Şimdi, RAID-4'e yaklaşık aynı anda 4 ve 13 bloklarına gönderilen 2 küçük yazma olduğunu hayal edin (şemada (*) ile işaretlenmiştir). Bu disklerin verileri 0 ve 1 disklerindedir. Bu nedenle verilerin okunması ve yazılması paralel olarak gerçekleşebilir, bu da iyidir. Ortaya çıkan sorun, eşlik diskindedir; her iki istek de 4 ve 13 için ilgili eşlik bloklarını, eşlik blokları 1 ve 3'ü (+ ile işaretlenmiş) okumak zorundadır. Umarız sorun artık açıktır: eşlik diski bu tür iş yükü altında bir darboğazdır; bu nedenle bazen buna parite tabanlı RAID için **küçük yazma sorunu** (**small-write problem**) diyoruz. Böylece, veri disklerine paralel olarak erişilebilmesine rağmen, eşlik diski herhangi bir paralelliğin gerçekleşmesini engeller;

parite diski nedeniyle sisteme yapılan tüm yazma işlemleri serileştirilecektir. Eşlik diskinin mantıksal G/Ç başına iki G/Ç (bir okuma, bir yazma) gerçekleştirmesi gerektiğinden, eşlik diskinin bu iki G/Ç üzerindeki performansını hesaplayarak RAID-4'teki küçük rasgele yazmaların performansını hesaplayabiliriz. ve böylece (R/2) MB/s elde ederiz. Rastgele küçük yazmalar altında RAID-4 verimi korkunç; sisteme disk ekledikçe düzelmez.

RAID 4'teki G/Ç gecikmesini analiz ederek sonuca varıyoruz. Artık bildiğiniz gibi, tek bir okuma (hata olmadığı varsayılarak) yalnızca tek bir diske eşlenir ve bu nedenle gecikme süresi, tek bir disk isteğinin gecikme süresine eşdeğerdir. Tek bir yazmanın gecikmesi, 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 farklılıklar vardır, çünkü her iki okumanın da tamamlanmasını beklememiz ve böylece en kötü durum konumlandırma süresini elde etmemiz gerekir, ancak sonra güncellemeler arama maliyetine neden olmaz ve bu nedenle ortalamadan daha iyi bir konumlandırma maliyeti olabilir).

38.7 RAID Düzey 5: Dönen Eşlik

Küçük yazma sorununu çözmek için (en azından kısmen), Patterson, Gibson ve Katz RAID-5'i tanıttı. RAID-5, eşlik bloğunu sürücüler arasında döndürmesi (**rotates**) dışında RAID-4 ile 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: **RAID Düzey 5: Dönen Eşlik**

Gördüğünüz gibi, RAID-4 için eşlik diski darboğazını ortadan kaldırmak için artık her şerit için eşlik bloğu diskler arasında döndürülüyor.

RAID-5 Analizi

RAID-5 için yapılan analizlerin çoğu, RAID-4 ile aynıdır. Örneğin, iki seviyenin etkin kapasitesi ve arıza toleransı aynıdır. Sıralı okuma ve yazma performansı da öyle. Tek bir isteğin (okuma veya yazma) gecikme süresi de RAID-4 ile aynıdır.

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

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

Şekil 38.8: **RAID Kapasitesi, Güvenilirliği ve Performansı**

Böylece paralel ilerleyebilirler. Aslında, çok sayıda rasgele istek verildiğinde, tüm diskleri eşit şekilde meşgul tutabileceğimizi varsayabiliriz. Durum buysa, küçük yazmalar için toplam bant genişliğimiz N/4 · R MB/s olacaktır. Dört kayıp faktörü, her bir RAID-5 yazmanın hala toplam 4 G/Ç işlemi oluşturmasından kaynaklanır; bu, yalnızca eşlik tabanlı RAID kullanmanın maliyetidir.

RAID-5, daha iyi olduğu birkaç durum dışında temel olarak RAID-4 ile aynı olduğundan, pazarda neredeyse tamamen RAID-4'ün yerini almıştır. Olmadığı tek yer, büyük yazma dışında hiçbir şey yapmayacaklarını bilen sistemlerdir, böylece küçük yazma sorununu tamamen ortadan kaldırır [HLM94]; bu durumlarda, oluşturulması biraz daha basit olduğu için RAID-4 bazen kullanılır.

38.8 RAID Karşılaştırması: Özet

Şimdi, Şekil 38.8'deki RAID düzeylerinin basitleştirilmiş karşılaştırmasını özetliyoruz. Analizimizi basitleştirmek için bazı ayrıntıları atladığımızı unutmayın. Örneğin, ikizlenmiş bir sistemde yazarken ortalama arama süresi, yalnızca tek bir diske yazarken olduğundan biraz daha yüksektir, çünkü arama süresi en fazla iki aramadır (her diskte bir tane). Bu nedenle, iki diske rasgele yazma performansı genellikle tek bir diskin rasgele yazma performansından biraz daha az olacaktır. Ayrıca, eşlik diskini RAID-4/5'te güncellerken, eski eşliğin ilk okuması büyük olasılıkla tam bir arama ve dönüşe neden olur, ancak eşliğin ikinci yazılması yalnızca dönüşle sonuçlanır. Son olarak, ikizlenmiş RAID'lere yönelik sıralı G/Ç, diğer yaklaşımlara[1] 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 düzeylerindeki 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 güvenilirliği umursamıyorsanız, şeritleme kesinlikle en iyisidir. Ancak rastgele G/Ç performansı ve güvenilirliği istiyorsanız, yansıtma en iyisidir; ödediğiniz maliyet kayıp kapasitededir. Ana hedefleriniz kapasite ve güvenilirlik ise, RAID-5 kazanır; ödediğiniz maliyet küçük yazma performansındadır. Son olarak, 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 var. İşte sonunda hakkında yazabileceğimiz bazı şeyler.

Örneğin, orijinal taksonomiden Düzey 2, 3 ve çoklu disk hatalarını 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 yedeği (**hot spare**) vardır. Başarısız durumdaki performansa ve arızalı diskin yeniden oluşturulması sırasındaki performansa ne olur? **Gizli sektör hatalarını** (**latent sector errors**) hesaba katmak veya **bozulmayı engellemek** (**block corruption**) [B+08] için daha gerçekçi hata modelleri ve bu tür hataları işlemek için birçok teknik vardır (ayrıntılar için veri bütünlüğü bölümüne bakın). Son olarak, **RAID'i bir yazılım** (**software RAID**) katmanı olarak bile oluşturabilirsiniz: bu tür yazılım RAID sistemleri daha ucuzdur ancak tutarlı güncelleme sorunu [DAA05] dahil olmak üzere başka sorunları vardır.

38.10 Özet

RAID'i tartıştık. RAID, bir dizi bağımsız diski büyük, daha geniş ve daha güvenilir tek bir varlığa dönüştürür; daha da önemlisi, bunu şeffaf bir şekilde yapar ve bu nedenle yukarıdaki donanım ve yazılım, değişiklikten nispeten habersizdir.

Aralarından seçim yapabileceğiniz pek çok olası RAID düzeyi vardır ve kullanılacak tam RAID düzeyi büyük ölçüde son kullanıcı için neyin önemli olduğuna bağlıdır. Örneğin, ikizlenmiş RAID basit, güvenilirdir ve genellikle yüksek bir kapasite maliyeti karşılığında iyi performans sağlar. Buna karşılık RAID-5, güvenilirdir ve kapasite açısından daha iyidir, ancak iş yükünde küçük yazmalar olduğunda oldukça düşük performans gösterir. Belirli bir iş yükü için bir RAID seçmek ve parametrelerini (yığın boyutu, disk sayısı vb.) uygun şekilde ayarlamak 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. SIGMET- RICS 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 abovewith 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 intro- ducing 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. SIG- MOD 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 afail-stop manner.*

Ödev (Simülasyon)

Bu bölüm, RAID sistemlerinin nasıl çalıştığına ilişkin bilginizi pekiştirmek için kullanabileceğiniz basit bir RAID simülatörü olan raid.py'yi tanıtmaktadır. Ayrıntılar için "README" dosyasına bakın.

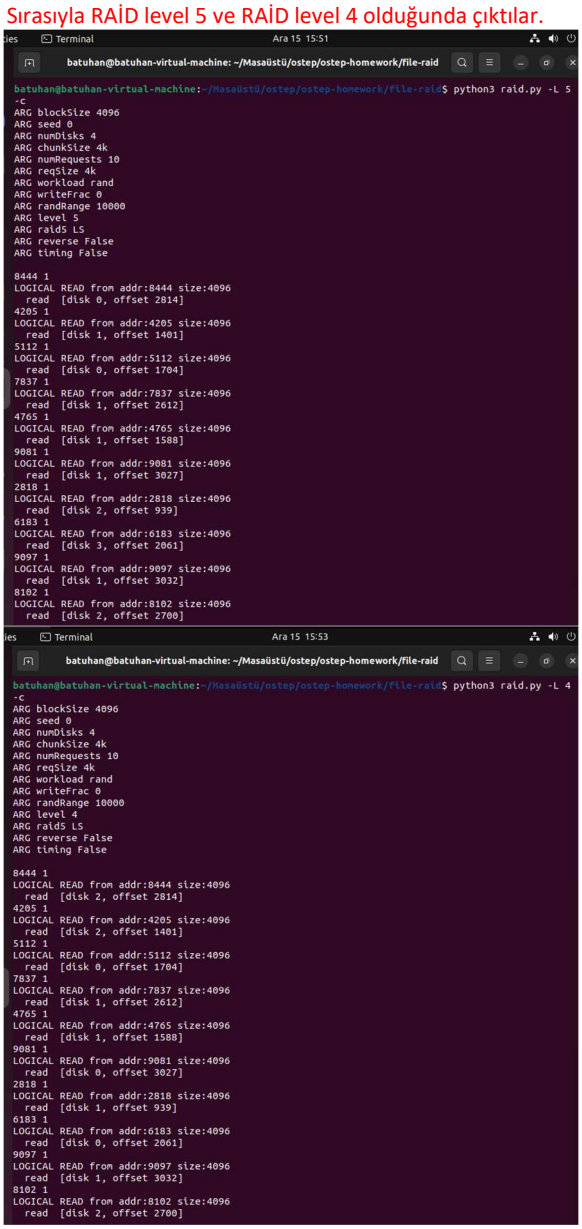
Sorular

1. Simülatörü kullanarak bazı temel RAID eşleme testleri gerçekleştirin. Farklı seviyelerde (0, 1, 4, 5) çalıştırın ve bir istek kümesinin eşleştirmelerini anlayıp anlamadığınızı görmek için deneyin. RAID-5 için sol simetrik ve sol asimetrik düzenler arasındaki farkı anlayıp anlayamadığınıza bakın. Yukarıdakinden farklı problemler oluşturmak için bazı farklı rastgele tohumlar kullanın.
2. İlk problemin aynısını yapın, ancak bu sefer yığın boyutunu `-C` ile değiştirin. Yığın boyutu eşlemeleri nasıl değiştirir?
3. Yukarıdakinin aynısını yapın, ancak her sorunun doğasını tersine çevirmek için `-r` bayrağını kullanın.
4. Şimdi ters bayrağını kullanın, ancak her isteğin boyutunu `-S` bayrağıyla artırın. RAID düzeyini değiştirirken 8k, 12k ve 16k boyutlarını belirtmeyi deneyin. İsteğin boyutu arttığında temeldeki G/Ç modeline ne olur? Bunu sıralı iş yükü ile de denediğinizden emin olun (`-W sıralı`); RAID-4 ve RAID-5 hangi istek boyutları için G/Ç açısından çok daha verimlidir?
5. 4 disk kullanarak RAID seviyelerini değiştirirken RAID'e 100 rasgele okumanın performansını tahmin etmek için simülatörün zamanlama modunu (`-t`) kullanın.
6. Yukarıdakilerin aynısını yapın, ancak disk sayısını artırın. Disk sayısı arttıkça her bir RAID düzeyinin performansı nasıl ölçeklenir?
7. Yukarıdakilerin aynısını yapın, ancak okumalar yerine tüm yazmaları (`-w 100`) kullanın. Her bir RAID seviyesinin performansı şimdi nasıl ölçekleniyor? 100 rasgele yazmanın iş yükünü tamamlamanın ne kadar süreceğini kabaca tahmin edebilir misiniz?
8. Zamanlama modunu son bir kez çalıştırın, ancak bu kez sıralı bir iş yüküyle (-W sıralı). Performans, RAID düzeyine göre ve okuma ve yazma yaparken nasıl değişir? Her isteğin boyutunu değiştirirken ne olur? RAID-4 veya RAID-5 kullanırken bir RAID'e hangi boyutta yazmalısınız?

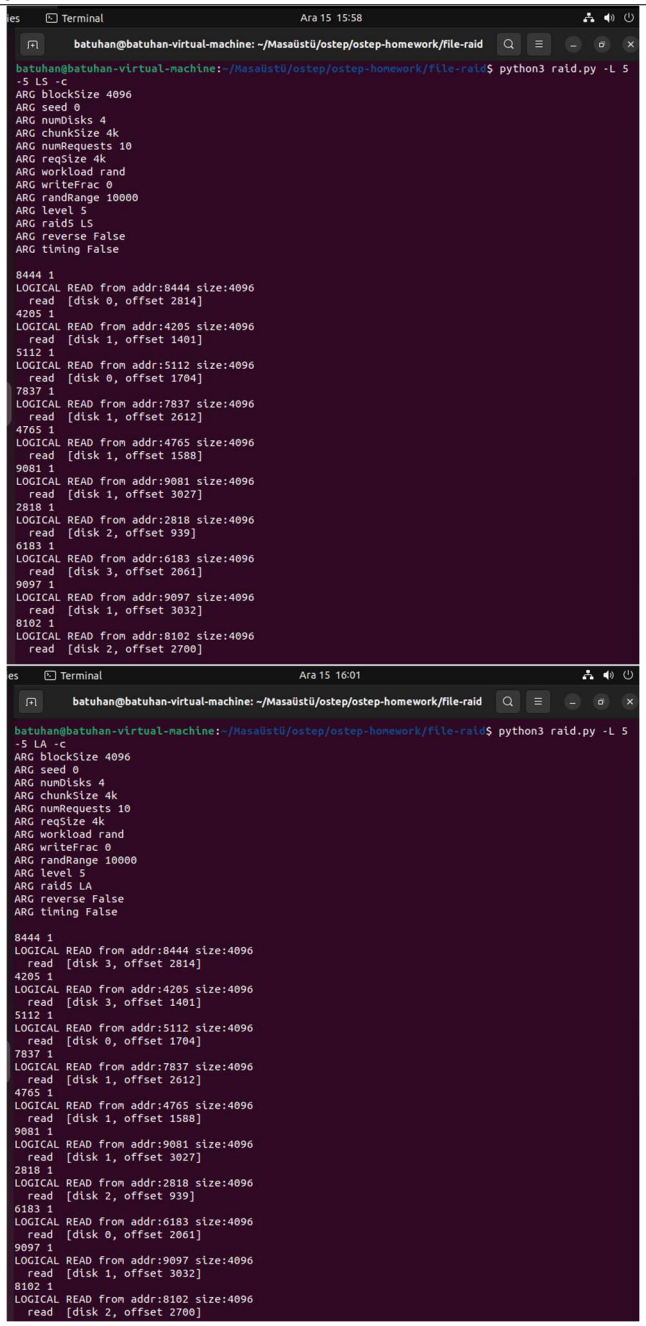
Çözümler sonraki sayfalarda

Cevaplar

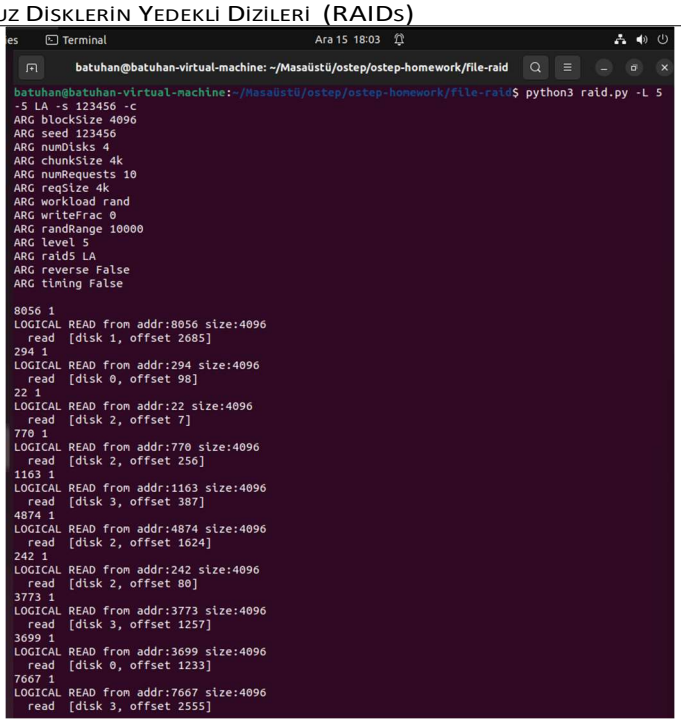
1. Sırasıyla RAİD level 5 ve RAİD level 4 olduğunda çıktılar.



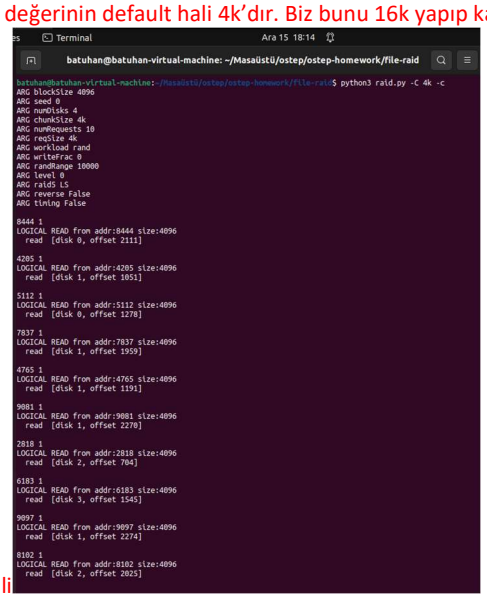
Sol simetrik ve sol asimetrik(fotograflar Sırasıyla eklendi) düzenler arasındaki fark.



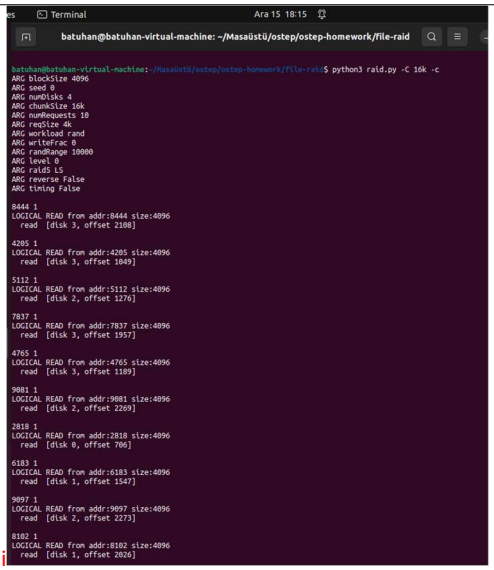
Hemen bir üstteki fotograftaki çıktıyı seedi değiştirip tekrar görelim



2. Chunksize değerinin default hali 4k'dır. Biz bunu 16k yapıp karşılaştıralım.



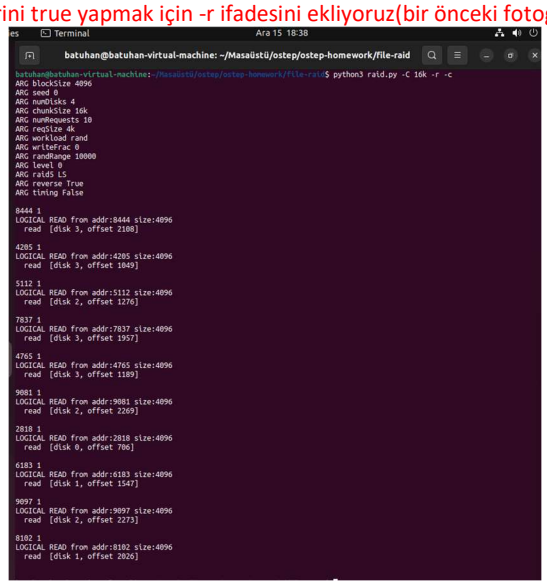
4k değerli hali



16k değer hali

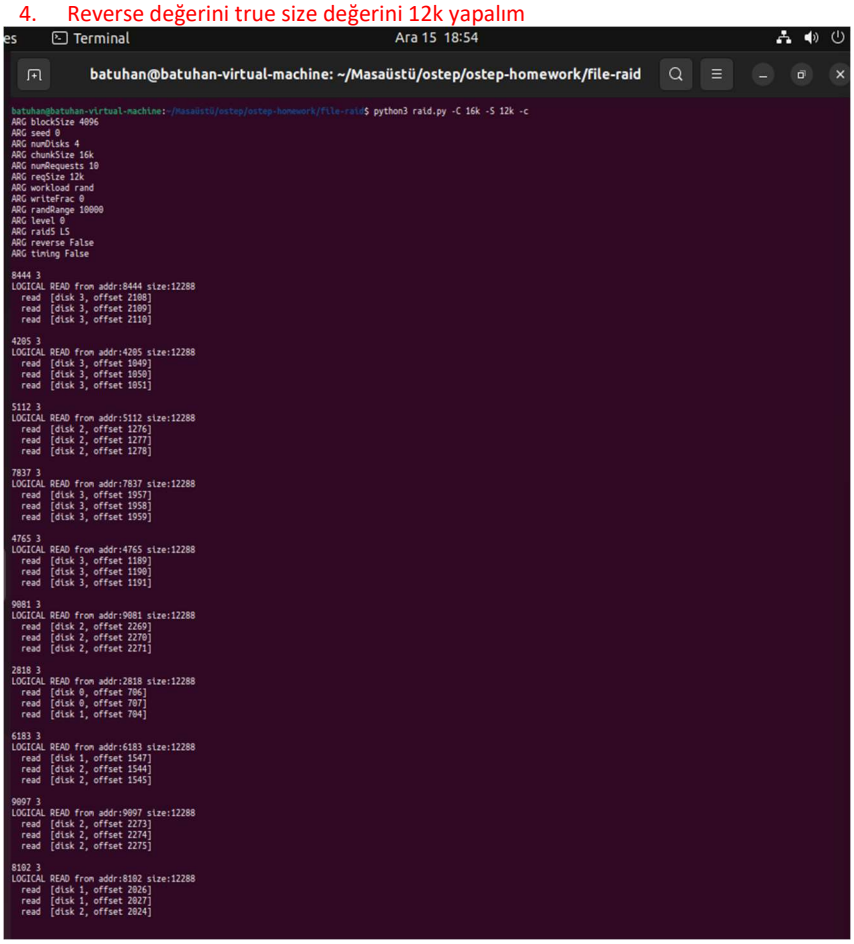
Daha küçük bir **chunksize** değeri, daha küçük veri bloğu boyutuna sahip olacak ve dolayısıyla daha fazla blok erişimi için daha fazla işlem gerektirecektir. Bu, sistem performansını düşürebilir ancak daha fazla veri koruması sağlayacaktır. Daha büyük bir **chunksize** değeri ise, daha büyük veri bloğu boyutuna sahip olacak ve dolayısıyla daha az işlem gerektirecek ancak daha az veri koruması sağlayacaktır. Bu, sistem performansını artırabilir ancak daha az veri koruması sağlayacaktır.

3. Revers değerini true yapmak için -r ifadesini ekliyoruz(bir önceki fotografın true hali)



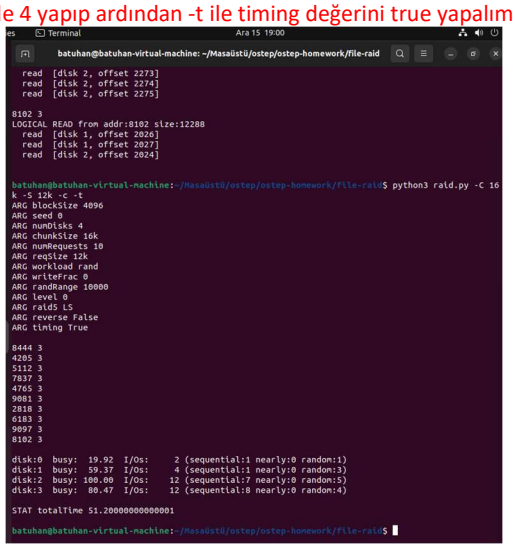
reverse değişkeninin **true** ve **false** olması arasındaki fark, veri bloğu okuma ve yazma işlemlerinin nasıl gerçekleştirileceğini belirtir. **reverse** değeri **true** olarak ayarlanırsa, veri bloğu okuma ve yazma işlemleri tersine çalışacaktır. Bu, veri bloğu erişim sırasının tersine çalışmasına neden olacak ve dolayısıyla sistem performansını etkileyebilir. Ancak, bu durum veri korumasını artırabilir. **reverse** değeri **false** olarak ayarlanırsa, veri bloğu okuma ve yazma işlemleri normale göre gerçekleştirilecektir. Bu, veri bloğu erişim sırasının normale göre çalışmasına neden olacak ve dolayısıyla sistem performansını etkilemeyecektir. Ancak, bu durum veri korumasını azaltabilir.

4. Reverse değerini true size değerini 12k yapalım



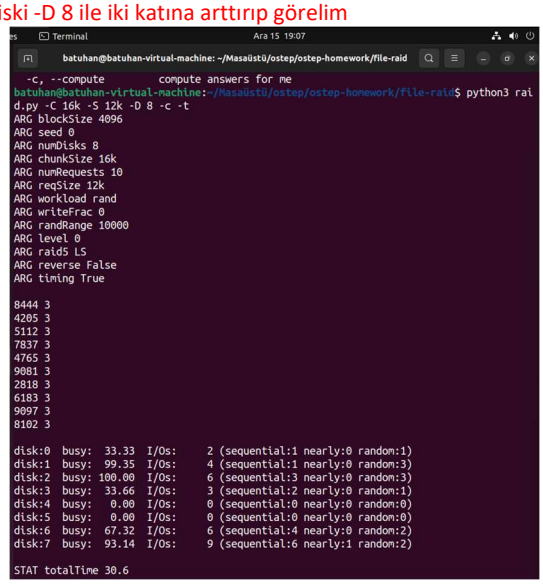
Bu durumda, **reverse** değişkeni **false** olarak ayarlanmış olsa da **true** olarak değiştirilerek veri bloğu okuma ve yazma işlemlerini tersine çalıştırılır. Ayrıca, **size** değişkeni **4k** olarak ayarlanmış olsa da **12k** olarak değiştirilerek veri bloğu boyutu değiştirilir.

5. Diski -D 4 ile 4 yapıp ardından -t ile timing değerini true yapalım



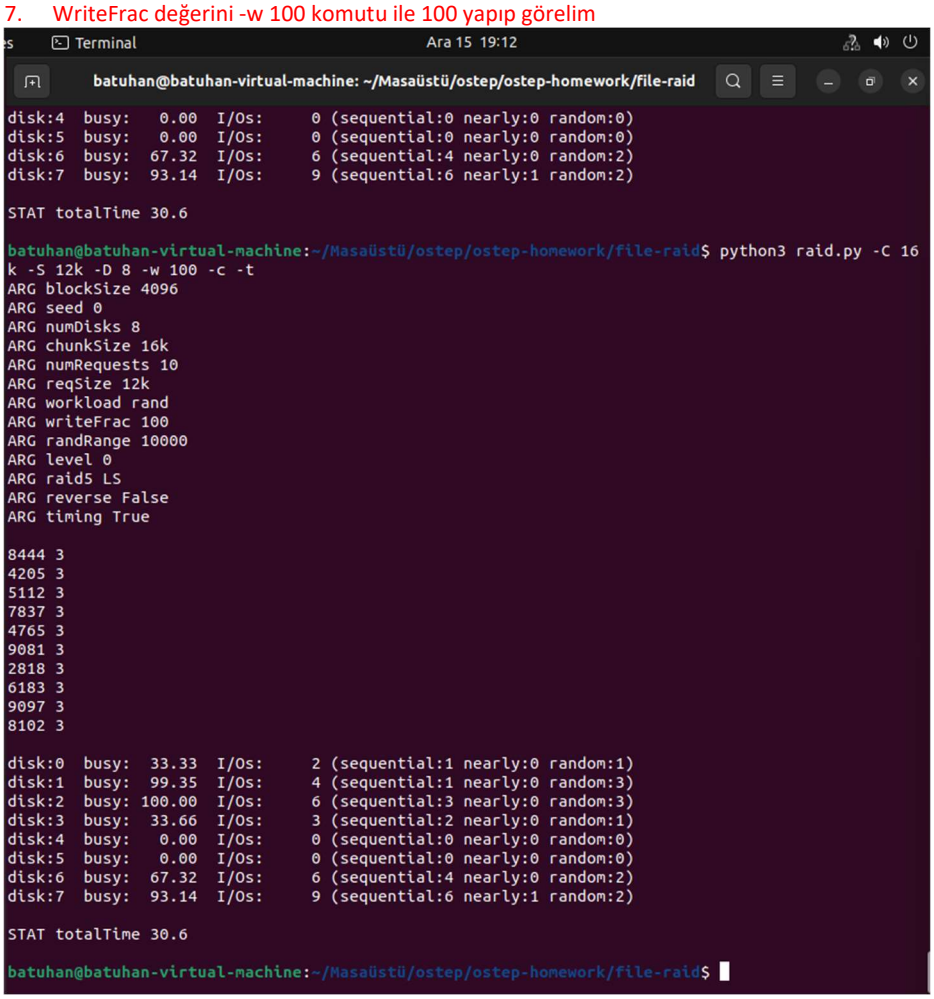
timing değişkeni, sistem performansını ölçmek için kullanılan bir değişken olup **false** olarak ayarlanmıştır. Eğer **timing** değeri **true** olarak değiştirilirse, sistem performansını ölçmek için kullanılan işlemler etkinleştirilecektir. Bu, disk kullanımı, veri bloğu erişim sırası ve benzeri bilgilerin toplanmasına ve raporlanmasına neden olacaktır.

6. Diski -D 8 ile iki katına arttırıp görelim



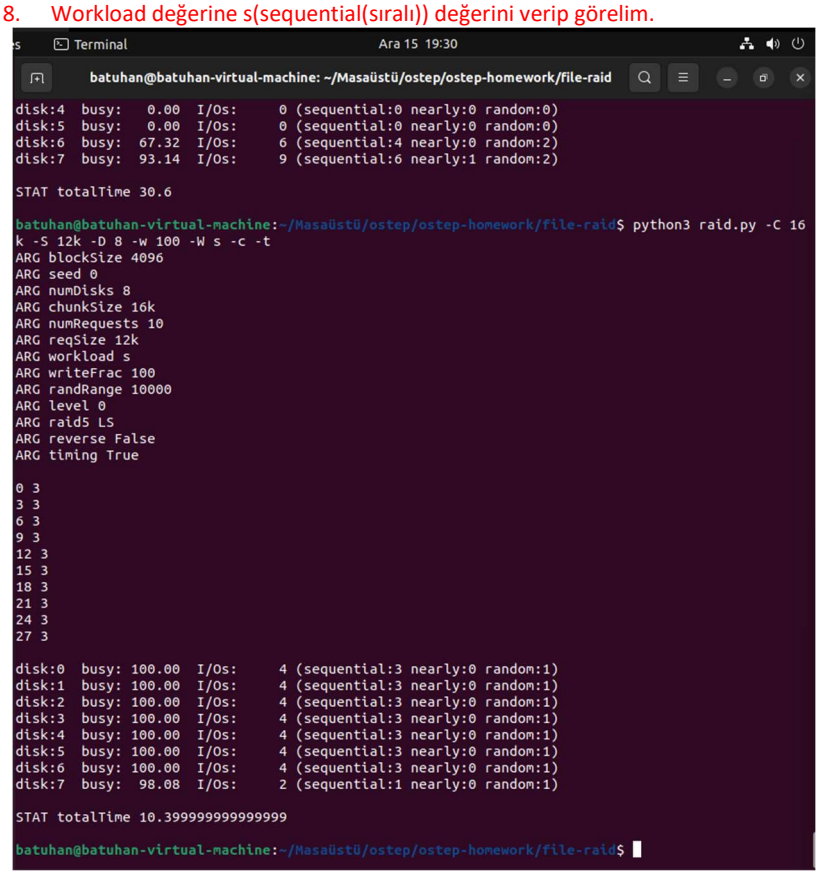
Diskimizi arttırınca totalTime değerinin düştüğünü görüyoruz

7. WriteFrac değerini -w 100 komutu ile 100 yapıp görelim



WriteFrac, verinin bir parçasının dizideki her diske yazılma oranını belirtir. Bu değer 100 yapılırsa, verinin tümü dizideki her diske yazılacaktır. Bu, dizideki her diskin daha fazla yük altına gireceğini ve dolayısıyla daha fazla işlem yapmasını ve daha fazla performans göstermesini sağlar. Ancak, dizideki disklerin tümü aynı anda çalışır halde olacaktır ve bu da sistemin enerji tüketimini artırabilir. Ancak bizim totalTime değerimiz değişmedi nedeni zaten iş yükünün fazla olmaması olabilir.

8. Workload değerine s(sequential(sıralı)) değerini verip görelim.



Workload değeri, verinin nasıl yazılacağını belirtir. Eğer workload değeri "s" olarak ayarlanırsa, veri sıralı bir şekilde yazılacaktır. Bu durumda, veri dizideki diskler arasında sıralı bir şekilde dağıtılacaktır ve her bir disk sırasıyla işlem yapacaktır. Bu, dizideki disklerin daha az yük altına gireceğini ve dolayısıyla daha az işlem yapmasını ve daha düşük bir performans göstermesini sağlar. Ancak, verinin sıralı bir şekilde yazılması sistemin daha iyi bir şekilde organize olmasını sağlar ve bu da okuma işlemlerinde daha hızlı bir performans göstermesine yardımcı olabilir. Sonuçtada gördüğümüz gibi totalTime değeri önceden 30 saniye iken 10 saniyeye düşüyor.