38

Redundant Arrays of Inexpensive Disks (RAIDs) (Bağımsız Disklerin Artıklıklı Dizisi)

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.

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 **Bağımsız Disklerin Artıklıklı Dizisini(**the **Redundant Array of Inexpensive Disks)** tanıtıyoruz. Terim, 1980'lerin sonlarında U.C. 'deki bir grup araştırmacı tarafından tanıtıldı. Berkeley (Profesör David Patterson ve Randy Katz ve daha sonra öğrenci Garth Gibson tarafından yönetilen); Bu sıralarda birçok farklı araştırmacı, daha iyi bir depolama sistemi oluşturmak için birden fazla disk kullanma temel fikrine çok yönlü olarak ulaştı [BG88, K86, K88, PB86,SG86].

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

RAID'ler tek bir diske göre bir dizi avantaj sunar. Bir ilerleme *performanstır* . Birden fazla diski paralel olarak kullanmak, G/Ç sürelerini büyük ölçüde hızlandırabilir. Diğer bir faydası da *kapasitedir* . Large data sets demand largedisks. Son olarak, RAID'ler *güvenilirliği* artırabilir; verileri çok uçlu disklere 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.

TIP: TRANSPARENCY ENABLES DEPLOYMENT(İpucu:Şeffaflık Dağıtımı)
Bir sisteme yeni işlevlerin nasıl ekleneceğini düşünürken, bu işlevlerin **şeffaf** bir şekilde, hiçbir değişiklik gerektirmeyecek şekilde eklenip eklenemeyeceği her zaman göz önünde bulundurulmalıdır. Sistemin geri kalanı 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 bir RAID depolama dizisi kurabilir ve sistemin geri kalanı (ana bilgisayar, işletim sistemi, vb.) kullanmaya başlamak için bir bit değiştirmek zorunda kalmazdı. Bu **dağıtım(deployment)** sorununu çözerek, RAID ilk günden itibaren daha başarılı hale getirildi.

Şaşırtıcı bir şekilde, RAID'ler bu avantajları onları kullanan sistemlere **şeffaf bir şekilde(transparency)** 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. Tion 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 tartışalım. Arayüz, hata modeli ile başlıyoruz ve ardından RAID tasarımının üç önemli eksen boyunca nasıl değerlendirilebileceğini tartışıyoruz: kapasite, güvenilirlik ve performance. Daha sonra RAID tasarımı ve uygulaması için önemli olan bir dizi başka konuyu tartışıyoruz .

38.1 Interface And RAID Internals (Arayüz ve RAID İç Yapıları)

Yukarıdaki bir dosya sistemine, RAID büyük, (umarım) hızlı ve (umarım) güvenilir bir diske benziyor. 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 ardından bir veya daha fazla fiziksel G /Ç vermelidir. bunu yapmak için. 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 (örneğin, SCSI veya SATA) ile ayrı bir donanım kutusu olarak oluşturulur. Bununla birlikte, dahili olarak , RAID'ler, RAID'in çalışmasını yönlendirmek için firmware çalıştıran bir mikrodenetleyiciden, DRAM gibi geçici belleklerden oluşan oldukça karmaşıktır. veri blokları okundukça ve yazıldıkça, ve bazı durumlarda,

arabelleğe geçici olmayan bellek, eşlik hesaplamalarını gerçekleştirmek için güvenli ve hatta belki de özelleştirilmiş mantık yazardır (aşağıda da göreceğimiz gibi bazı RAID seviyelerinde 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 Fault Model (Arıza Model)

RAID'i anlamak ve farklı yaklaşımları karşılaştırmak için aklımızda bir hata modeli olmalıdır. RAID'ler, belirli disk hatalarını tespit etmek 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 erişilemez hale gelen bir günah bloğu hakkında endişelenmemize gerek yoktur (bazen gizli sektör hatası olarak adlandırılır). Bu daha karmaşık (ve ne yazık ki daha gerçekçi) disk hatalarını daha sonra ele alacağız.

38.3 How To Evaluate A RAID(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 $N \cdot B$; Buna karşılık, her bloğun iki kopyasını tutan bir sistemimiz varsa (**yansıtma(mirroring)** olarak adlandırılır), yararlı bir kapasite elde ederiz $(N \cdot B)/2$. Farklı şemalar (örneğin, pariteye dayalı olanlar) aralarında düşme eğilimindedir .

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

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

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

38.4 RAID Level 0: Striping (RAID Level 0: Şeritleme)

İlk RAID seviyesi aslında bir RAID seviyesi değildir, çünkü artıklık yoktur. Bununla birlikte, RAID seviye 0 veya daha iyi bilindiği gibi **şeritleme(Striping)**, performans ve kapasite konusunda mükemmel bir üst sınır görevi görür ve bu nedenle anlamaya değer.

En basit şeritleme şekli, sistemin diskleri boyunca blokları aşağıdaki gibi şeritleyecektir (burada 4 diskli bir dizi 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

Figure 38.1: **RAID-0: Simple Striping(Basit Şeritleme)**

Şekil 38.1'den temel fikri elde edersiniz: dizinin bloklarını diskler arasında hepsini bir kez deneme tarzında yaymak. Bu yaklaşım, dizinin bitişik parçaları için istekler yapıldığında diziden en fazla paralelliği ayıklamak için tasarlanmıştır (büyük, sıralı bir okumada olduğu gibi, inceleme için). Aynı sıradaki bloklara **şerit(stripe)** diyoruz; bu nedenle, 0, 1, 2 ve 3 numaralı bloklar yukarıdaki aynı şerittedir .

Örnekte, bir sonrakine geçmeden önce her diske yalnızca 1 bloğun (her biri 4KB boyutunda) yerleştirildiği basit varsayımını yaptık. Ancak, bu düzenlemenin böyle olmasına gerek yoktur. Örneğin, blokları diskler arasında Şekil 38.2'de olduğu gibi düzenleyebiliriz :

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

Figure 38.2: **Striping With A Bigger Chunk Size(Büyük Yığın Şeritleme)**

Bu örnekte, bir sonraki diske geçmeden önce her diske iki adet 4KB blok yerleştiriyoruz . Bu nedenle, bu RAID dizisinin **yığın boyutu(chunk size)** 8KB'dir ve bir şerit bu nedenle 4 parçadan veya 32 KB'lık veriden oluşur.

ASIDE: THE RAID MAPPING PROBLEM(Raid Eşleme Sorunu)

RAID'in kapasitesini, güvenilirliğini ve performans özelliklerini incelemeden önce , önce **haritalama sorunu(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 çok fazla karmaşıklığa ihtiyacımız yoktur. 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 % number_of_disks
Offset = A / number_of_disks
```

Bunların hepsinin tamsayı işlemleri olduğunu unutmayın (örneğin, 4 / 3 = 1 değil 1.33333...). 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.

Chunk Sizes(Yığın Boyutları)

Yığın boyutu çoğunlukla dizinin performansını etkiler. Örneğin, küçük bir yığın boyutu, birçok dosyanın birçok diskte şeritleneceğini ve böylece tek bir dosyaya okuma ve yazma işlemlerinin paralelliğini artıracağını ima eder; Bununla birlikte, birden fazla diskteki bloklara erişmek için konumlandırma süresi artar, çünkü tüm istek için konumlandırma süresi, isteklerin tüm disklerdeki konumlandırma sürelerinin maksimumuna göre belirlenir. Sürücü.

Öte yandan, büyük bir yığın boyutu, bu tür dosya içi parallelizmi azaltır ve böylece yüksek verim elde etmek için birden fazla eşzamanlı isteğe dayanır. Bununla birlikte, büyük yığın boyutları konumlandırma süresini azaltır; Örneğin, tek bir dosya bir yığının içine sığarsa ve böylece tek bir diske yerleştirilirse, ona erişirken 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.

Back To RAID-0 Analysis(RAID-0 Analizine Geri Dön)

Şimdi şeritlemenin kapasitesini, güvenilirliğini ve performansını değerlendirelim. Kapasite perspektifinden bakıldığında, mükemmeldir: her biri boyuttaki *N* diskler verilir.
B bloklar, şeritleme *N* teslimi Yararlı kapasiteye sahip *B* blokları. Stanttan - güvenilirlik noktası, şeritleme de mükemmel, ancak kötü şekilde: herhangi bir disk başarısızlık veri kaybına yol açacaktır. Son olarak, performans mükemmeldir: tüm diskler, genellikle paralel olarak, kullanıcı G/Ç isteklerine hizmet etmek için kullanılır.

Evaluating RAID Performance(RAID Performansını Değerlendir)

RAID performansını analiz ederken, iki farklı perfor-mance metriği göz önünde bulundurulabilir. Birincisi, *tek istekli gecikme süresidir*. Tek bir G/Ç isteğinin bir RAID'e olan latensitesini anlamak, tek bir mantıksal G/Ç işlemi sırasında ne kadar paralelliğin var olabileceğini ortaya koyduğu için yararlıdır. İkincisi, RAID'in *kararlı durum aktarım hızı* , 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** ve **rastgele(sequential** and **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 istek serisi) 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. İnceleme için, rastgele bir istek akışı önce mantıksal adres 10'da, sonra 550.000 mantıksal adreste 4KB'ye erişebilir, sonra 20.100'de vb. Veritabanı yönetim sistemindeki (DBMS) işlemsel iş yükleri gibi bazı önemli iş yükleri bu tür bir erişim modeli sergiler ve bu nedenle önemli bir değer olarak kabul edilir. Iş yük -ünü.Tabii ki, 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 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 aktarabildiğini varsayacağız rastgele iş yükü. Genel olarak, S, *R'den* çok daha büyüktür (yani, $S \gg R$).

Bu farkı anladığımızdan emin olmak için basit bir egzersiz yapalım. Özellikle, aşağıdaki disk karakteristikleri 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:

Average seek time(Ortalama
arama süresi) 7 ms
Ortalama dönme gecikmesi3ms
Diskin aktarım hızı l50 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 :

$$S = \frac{Amount\ of\ Data}{Time\ to\ access} = \frac{10\ MB}{210\ ms} = 47.62\ MB/s$$

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.

$$R = \frac{Amount\ of\ Data}{Time\ to\ access} = \frac{10\ KB}{10.195\ ms} = 0.981\ MB/s$$

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

Back To RAID-0 Analysis, Again(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.

Kararlı durum sıralı aktarım hızı perspektifinden, sistemin tam bant genişliğini elde etmek için ex-pect yaparız. 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 olanıdır hem de bir üst sınır görevi görecektir diğer RAID düzeyleriyle karşılaştırma .

38.5 RAID Level 1: Mirroring(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; Her kopya elbette 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

Figure 38.3: **Simple RAID-1: Mirroring(Basit RAID-1:Yansıtma)**

Örnekte, disk 0 ve disk 1 aynı içeriğe sahiptir ve disk 2 ve disk 3 de aynıdır; veriler bu yansıtma çiftleri arasında şeritlidir. 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**) olarak adlandırılır, çünkü aynalı çiftler (RAID-1) ve daha sonra bunların üstünde çizgiler (RAID-0) kullanır; başka bir ortak düzenleme, iki büyük şeritleme (RAID-0) dizisi ve ardından bunların üzerine aynalar (RAID-1) içeren RAID-01'dir (veya RAID 0 +1, **şeritlerin yansımasıdır**). 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 . Bununla birlikte, bu yazmaların paralel olarak gerçekleşebileceğini unutmayın; örneğin, mantıksal blok 5'e yazma, aynı anda disk 2 ve 3'e ilerleyebilir.

RAID-1 Analysis(RAID-1 Analizi)

RAID-1'i değerlendirelim. Kapasite açısından bakıldığında, RAID-1 pahalıdır; yansıtma seviyesi = 2 ile, zirvemizin sadece yarısını elde ederiz.Yararlı kapasite B bloklarının N diskleri ile , RAID-1 kullanışlı kapasite ($N \cdot B$)/2.

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 , yansıtılmış bir sistem (yansıtma düzeyi 2 olan) kesin olarak 1 disk hatasını ve hangi disklerin arızalandığına bağlı olarak N / 2 hatalarını tolere edebilir.

Uygulamada, genellikle böyle şeyleri şansa bırakmaktan hoşlanmayız; Bu nedenle çoğu insan yansıtmanın tek bir başarısızlığı ele almak için iyi olduğunu düşünür.Son olarak, performansı analiz ediyoruz. Tek bir okuma isteğinin gecikmesi perspektifinden , tek bir diskteki gecikme süresiyle aynı olduğunu görebiliriz; RAID-1'in yaptığı tek şey,okumayı kopyalarından birine yönlendirmektir.

Bir yazma biraz farklıdır: tamamlanmadan önce tamamlanması için iki fiziksel yazma gerektirir. Bu iki yazma paralel olarak gerçekleşir ve bu nedenle zaman kabaca tek bir yazmanın zamanına eşdeğer olacaktır; Bununla birlikte, mantıksal yazının her iki fiziksel yazının da tamamlanmasını beklemesi gerektiğinden,

ASIDE: THE RAID CONSISTENT-UPDATE PROBLEM(Raid Tutarlı Güncelleme Sorunu)

RAID-1'i analiz etmeden önce, **tutarlı güncelleme(consistent-update problem)** [DAA05] olarak bilinen herhangi bir çok diskli RAID sisteminde ortaya çıkan bir sorunu tartışalım. Sorun, tek bir mantıksal işlem sırasında birden çok diski güncel tutması gereken herhangi bir RAID'e yazma işleminde 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. RAID daha sonra disk 0'a yazmayı yayınlar, ancak RAID diski 1'e istek göndermeden hemen önce bir güç kaybı (veya sistem çökmesi) meydana gelir. Bu talihsiz durumda, disk 0 isteğinin tamamlandığını varsayalım (ancak disk 1'e yapılan isteğin hiçbir zaman yayınlanmadığı için açıkça yapılmadığı açıktır).

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(atomically)** olarak değişmesidir, yani her ikisi de yeni sürüm olarak sona ermeli veya hiçbirini yapmamalıdır.

Bu sorunu çözmenin genel yolu, RAID'in yapmak üzere olduğu şeyi kaydetmek için bir tür **önceden yazma günlüğü(write-ahead log**) kullanmaktır (yani, iki diski belirli bir veri parçasıyla güncellemek). Bu yaklaşımı benimseyerek, bir çarpışma varlığında doğru şeyin gerçekleşmesini sağlayabiliriz ; bekleyen tüm işlemleri RAID'e yeniden oynatan bir **kurtarma(recovery)** prosedürü çalıştırarak , yansıtılmış iki kopyanın (RAID-1 durumunda) eşitlenmemesini sağlayabiliriz .

Son bir not: Her yazmada diske giriş yapmak son derece pahalı olduğundan, çoğu RAID donanımı bu tür bir günlük kaydı gerçekleştirdiği az miktarda geçici olmayan RAM (örneğin, pil destekli) içerir. Böylece, diske günlüğe kaydetmenin yüksek maliyeti olmadan tutarlı güncelleştirme sağlanır.

En kötü durumda arama ve iki isteğin dönme gecikmesi ve bu nedenle (ortalama olarak) tek bir diske yazmaktan biraz daha yüksek olacaktır.

Sabit durum aktarım hızını analiz etmek için, sıralı iş yüküyle başlayalım. Diske sırayla yazarken, her mantıksal yazma iki fiziksel yazma ile sonuçlanmalıdır; örneğin, mantıksal blok 0 yazdığımızda (yukarıdaki şekilde), RAID dahili olarak hem disk 0'a hem de disk 1'e yazacaktır . Böylece, maksimum bant genişliğinin ob-

tutuldu süresince Sıralı yazı Hedef a Yansıtılmış dizi dir ($\frac{N}{2} \cdot S$), veya yarım Ve tepe bant genişliği.

Ne yazık ki, sıralı bir okuma sırasında tam olarak aynı performansı elde ediyoruz. Sıralı bir okumanın daha iyi yapabileceği düşünülebilir, çünkü verilerin her ikisini birden değil, yalnızca bir kopyasını okuması gerekir. Ancak, bunun neden çok yardımcı olmadığını göstermek için bir örnek kullanalım. 0, 1, 2, 3, 4, 5, 6 ve 7 numaralı blokları okumamız gerektiğini düşünün. Diyelim ki 0'dan disk 0'a, 1'den disk 2'ye , 2'den disk 1'e ve 3'ten disk 3'e okundu. 0, 2, 1 numaralı disklere 4, 5, 6 ve 7'ye okumalar vererek devam ediyoruz.

ve sırasıyla 3. Tüm diskleri kullandığımız için, dizinin tam bant genişliğini elde ettiğimizi safça düşünebiliriz.

Bununla birlikte, durumun (zorunlu olarak) böyle olmadığını görmek için, tek bir diskin aldığı istekleri göz önünde bulundurun (örneğin, disk 0). İlk olarak, blok 0 için bir istek alır ; daha sonra, blok 4 için bir istek alır (blok 2'yi atlama). Aslında, her disk diğer her blok için bir istek alır. Atlanan blok üzerinde dönerken, istemciye yararlı bant genişliği sağlamaz. Böylece, her disk yalnızca en yüksek bant genişliğinin yarısını sunacaktır. Ve böylece, sıralı okumak vasiyet sadece edinmek a bant genişliği in ($\frac{N}{} \cdot S$) MB/s.

Rastgele okumalar, yansıtılmış RAID için en iyi durumdur. Bu durumda, okumaları tüm disklere dağıtabilir ve böylece mümkün olan tüm bant genişliğini elde edebilir. Böylece, rastgele okumalar için RAID-1 $N \cdot R$ MB/s.

Son olarak, rastgele yazmalar beklediğiniz gibi çalışır: N · R MB/s. Her mantıksal yazma iki fiziksel yazıya dönüşmelidir ve böylece diskler kullanımda olacak, istemci bunu yalnızca kullanılabilir bant genişliğinin yarısı olarak algılayacaktır. X mantıksal bloğuna yazma, iki farklı fiziksel diske iki paralel yazmaya dönüşse de, birçok küçük yeniden aramanın bant genişliği, şeritleme ile gördüğümüzün yalnızca yarısına ulaşır. Yakında göreceğimiz gibi, mevcut bant genişliğinin yarısını almak aslında oldukça iyi!

38.6 RAID Level 4: Saving Space With Parity(RAID Seviye4:Eşlik ile Yerden Tasarruf Etme)

Şimdi **eşlik(parity)** olarak bilinen bir disk dizisine artıklık eklemek için farklı bir yöntem sunuyoruz. Parite temelli yaklaşımlar daha az kapasite kullanmaya çalışır ve böylece aynalı sistemler tarafından ödenen büyük alan cezasının üstesinden gelir. Ancak bunu bir bedeli 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

Figure 38.4: **RAID-4 With Parity(Raid-4 Parite İle)**

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

Pariteyi hesaplamak için, şeridimizdeki herhangi bir bloğun kaybına dayanmamızı sağlayan matematiksel bir işlev kullanmamız gerekir. **Basit(Basic)** fonksiyon XOR'un püf noktası ortaya çıktı. Belirli bir bit kümesi için, tüm bu bitlerin XOR'u, bitlerde çift sayıda 1 varsa 0 ve tek sayı 1 varsa 1 döndürür. Mesela:

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 sırada sadece

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

bir 1 (C1) ve dolayısıyla XOR 1 (P) olmalıdır. Bunu basit bir şekilde hatırlayabilirsiniz: eşlik biti de dahil olmak üzere herhangi bir satırdaki 1 sayısının çift (tek değil) bir sayı olması gerekir; bu, RAID'in paritenin doğru olması için sürdürmesi gereken **değişmezdir(invariant)**.

Yukarıdaki örnekten, eşlik bilgilerinin bir hatadan kurtulmak için nasıl kullanılabileceğini de tahmin edebilirsiniz. C2 etiketli sütunun kaybolduğunu düşünün. Sütunda hangi değerlerin olması gerektiğini bulmak için, o satırdaki diğer tüm değerleri (XOR'd eşlik biti dahil) okumamız ve doğru cevabı **yeniden oluşturmamız(reconstruct)** yeterlidir. Özellikle, ilk satırın C2 sütunundaki değerinin kaybolduğunu varsayalım (bu bir 1'dir); Bu satırdaki diğer değerleri okuyarak (C0'dan 0, C1'den 0, C3'ten 1 ve P eşlik sütunundan 0), 0, 0, 1 ve 0 değerlerini alırız. XOR'un her satırda çift sayıda 1 tuttuğunu bildiğimiz için, eksik verilerin ne olması gerektiğini biliyoruz : 1. Ve XOR tabanlı bir parite şemasında yeniden yapılanma böyle çalışır! Yeniden yapılandırılmış değeri nasıl hesapladığımıza da dikkat edin: sadece veri bitlerini ve eşlik bitlerini birlikte XOR ediyoruz, aynı şekilde pariteyi ilk etapta hesapladığımız gibi.

Şimdi merak ediyor olabilirsiniz: tüm bu bitleri XORing'den bahsediyoruz ve yine de yukarıdan RAID'in her diske 4KB (veya daha büyük) bloklar yerleştirdiğini biliyoruz; Pariteyi hesaplamak için XOR'u bir grup bloğa nasıl uygularız ? Bunun da kolay olduğu ortaya çıktı. Veri bloklarının her bir bitinde bit bazında XOR gerçekleştirmeniz yeterlidir; her bit bilge XOR'un sonucunu eşlik bloğundaki karşılık gelen bit yuvasına koyun. Örneğin, 4 bit boyutunda bloklarımız varsa (evet, bu hala 4KB'lık bir bloktan biraz daha küçüktür, ancak resmi elde edersiniz), şöyle bir şeye benzeyebilirler :

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 Analysis(RAID-4 Analizi)

Şimdi RAID-4'ü analiz edelim. Kapasite açısından bakıldığında, RAID-4 1 kullanır koruduğu her disk grubu için eşlik bilgileri için disk. Bu nedenle, RAID grubu için kullanışlı kapasitemiz $(N - 1) \cdot B$.

Güvenilirliğin anlaşılması da oldukça kolaydır: RAID-4, 1 disk arızasını tolere eder ve daha fazlasını tolere etmez. Birden fazla disk kaybolursa, kayıp verileri yeniden yapılandırmanın bir 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

Figure 38.5: **Full-stripe Writes In RAID-4(Tam Şeritli Yazma RAID-4)**

Son olarak, performans var. Bu kez, kararlı durum verimini analiz ederek başlayalım. Sıralı okuma performansı , eşlik diski dışındaki tüm diskleri kullanabilir ve böylece en yüksek etkili bant genişliğini sağlayabilir. $(N - 1) \cdot S$ MB/s (kolay bir kasa).

Sıralı yazıların performansını anlamak için, önce bunların nasıl yapıldığını çözmeliyiz. Diske büyük bir veri yığını yazarken, RAID-4 **tam şeritli yazma(full-stripe write**.) olarak bilinen basit bir optimizasyon gerçekleştirebilir . Örneğin, 0, 1, 2 ve 3 bloklarının bir yazma isteğinin parçası olarak RAID'e gönderildiği durumu düşünün (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) paralel olarak yukarıdaki beş diske yazabilir (şekilde gri renkle vurgulanmıştır). Bu nedenle, tam şeritli yazmalar, RAID-4'ün diske yazması için en etkili yoldur .

Tam şeritli yazmayı anladıktan sonra, RAID-4'te sıralı yazmaların performansını hesaplamak kolaydır; etkili bant genişliği de $(N - 1) \cdot S$ MB/s. Eşlik diski sırasında sürekli kullanımda olmasına rağmen işlem, istemci bundan performans avantajı elde etmez.

Şimdi rastgele okumaların performansını analiz edelim. Yukarıdaki şekilden de görebileceğiniz gibi, bir dizi 1 bloklu rastgele okuma, sistemin veri disklerine yayılacak , ancak eşlik diskine yayılmayacaktır. Böylece, etkili performans: $(N - 1) \cdot R$ MB/s.

Son olarak kaydettiğimiz rastgele yazmalar, RAID-4 için en karmaşık durumu sunar. Yukarıdaki örnekte blok 1'in üzerine yazmak istediğimizi düşünün. Devam edip üzerine yazabiliriz, ancak bu bizi bir sorunla karşı karşıya bırakacaktır: P0 parite bloğu artık şeridin doğru parite değerini doğru bir şekilde yansıtmaz; Bu örnekte, P0 da güncelleştirilmelidir. Hem doğru hem de verimli bir şekilde nasıl güncelleyebiliriz?

İki yöntem olduğu ortaya çıktı. **Katkı paritesi olarak bilinen birincisi(additive parity)**, aşağıdakileri yapmamızı gerektirir. Yeni eşlik bloğunun değerini hesaplamak için, şeritteki diğer tüm veri bloklarını paralel olarak (örnekte, 0, 2 ve 3 numaralı bloklar) ve XOR yeni bloğa sahip olanları (1) okuyun. Sonuç, yeni parite bloğunuz olur. Yazmayı tamamlamak için, yeni verileri ve yeni eşliği de paralel olarak ilgili disklere yazabilirsiniz.

Bu teknikle ilgili sorun, disk sayısıyla ölçeklenmesi ve bu nedenle daha büyük RAID'lerde eşlik etmek için çok sayıda okuma gerektirmesidir. Böylece, **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 yeni diyeceğimiz yeni bir değerle yazmak istediğimizi düşünelim. Çıkarma yöntemi üç adımda çalışır. İlk olarak, C2'deki eski verileri (C2 eski = 1) ve eski pariteyi (P eski = 0) okuyoruz. Daha sonra, eski verileri ve yeni verileri karşılaştırırız; eğer aynıysa (örneğin, C2 yeni = C2 eski), o zaman eşlik bitinin de aynı kalacağını biliyoruz (yani, $P_{newi} = P_{oldi}$). Bununla birlikte, eğer farklılarsa, o zaman eski parite bitini mevcut durumunun tersine çevirmeliyiz , yani eğer (P_{oldi} == 1), P new 0 olarak ayarlanır ; eğer (P_{old} == 0), P_{new} 1 olarak ayarlanır. Tüm bu karmaşayı XOR ile düzgün bir şekilde ifade edebiliriz (burada XOR operatörü $\oplus$):

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

Bitlerle değil, bloklarla uğraştığımız için, bu hesaplamayı bloktaki tüm bitler üzerinde gerçekleştiririz (örneğin; her blokta 4096 bayt çarpılır) bayt başına 8 bit ile). Bu nedenle, çoğu durumda, yeni blok eski bloktan farklı olacaktır ve bu nedenle yeni parite bloğu da olacaktır.

Artık toplama paritesi hesaplamasını ne zaman kullanacağımızı ve çıkarma yöntemini ne zaman kullanacağımızı anlayabilmelisiniz. Toplama yönteminin çıkarma yönteminden daha az G / Ç gerçekleştirmesi için sistemde kaç diskin olması gerektiğini düşünün; çapraz geçiş noktası nedir?

Bu performans analizi için, subtracve yöntemini kullandığımızı varsayalım. Bu nedenle, her yazma için RAID'in 4 fiziksel G/Ç (iki okuma ve iki yazma) gerçekleştirmesi gerekir. Şimdi RAID'e gönderilen çok sayıda yazı olduğunu hayal edin; RAID-4 paralel olarak kaç tane performans gösterebilir? Anlamak için, RAID-4 düzenine tekrar 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

Figure 38.6: **Example: Writes To 4, 13, And Respective Parity Blocks**

Şimdi RAID-4'e gönderilen 2 küçük yazı olduğunu hayal edin. yaklaşık aynı zamanda, 4 ve 13 numaralı bloklara (diyagramda işaretlenmiş). Bu disklerin verileri 0 ve 1 numaralı disklerdedir ve bu nedenle verilere okuma ve yazma paralel olarak gerçekleşebilir, bu da iyidir. Ortaya çıkan sorun eşlik diskindedir; Her iki istek de 4 ve 13 için ilgili eşlik bloklarını , parite blokları 1 ve 3'ü (+ ile işaretlenmiş) okumalıdır. sorun artık açıktır: eşlik diski bu tür iş yükü altında bir zorlayıcıdır; Bu nedenle bazen buna eşlik tabanlı RAID'ler için **küçük yazma sorunu** (**small-write problem)** diyoruz.

paralel, eşlik diski herhangi bir paralelliğin gerçekleşmesini önler; sisteme yazılan tüm yazılar eşlik diski nedeniyle serileştirilir. 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. rastgele küçük yazmalar altında RAID-4 verimsizdir;sisteme disk ekledikçe gelişmiyor. RAID-4'teki G/Ç gecikmesini analiz ederek sonlandırıyoruz. Artık bildiğiniz gibi, tek bir okuma (hata olmadığı varsayılarak) yalnızca tek bir diske eşlenir ve bu nedenle gecikme süresi tek bir disk isteğinin gecikmesine eşdeğerdir.

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

38.7 RAID Level 5: Rotating Parity(RAID Düzey 5:Dönen Eşlik)

Küçük yazma problemini ele almak için (en azından kısmen), Patterson, Giboğlu 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

Figure 38.7: **RAID-5 With Rotated Parity(Döndürülmüş Pariteli)**

Gördüğünüz gibi, RAID-4 için eşlik diski zoruluğunu gidermek için her şeridin eşlik bloğu artık diskler arasında döndürülür.

RAID-5 Analysis(RAID-5 Analiz)

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 gecikmesi (okuma veya yazma olsun) da RAID-4 ile 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 eşliği için) ve disk 0 ve disk 2'ye yapılan isteklere dönüşecektir.

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$	$(N-1) \cdot B$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S$[1]	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$[1]	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} R$
Latency				
Read	T	T	T	T
Write	T	T	$2T$	$2T$

Figure 38.8: **RAID Capacity, Reliability, and Performance(RAID Kapasitesi, Güvenilirlik ve Performans)**

(blok 10 ve paritesi için). Böylece paralel olarak ilerleyebilirler. Aslında, genellikle çok sayıda rastgele istek verildiğinde, tüm diskleri eşit şekilde meşgul edebileceğimizi varsayabiliriz. Bu durumda, küçük yazmalar için toplam bant genişliğimiz $\frac{N}{4} \cdot R$ MB/s. Dört kayıp faktörü , her RAID-5 yazmasının hala toplam 4 G/Ç işlemi oluşturmasından kaynaklanmaktadır, bu da eşlik tabanlı RAID kullanmanın maliyetidir.

RAID-5, daha iyi olduğu birkaç durum dışında temelde RAID-4 ile aynı olduğundan,piyasadaki RAID-4'ün yerini neredeyse tamamen almıştır.Olmadığı tek yer, büyük bir yazıdan başka hiçbir şey yapmayacaklarını bilen sistemlerdir, böylece küçük yazma probleminden tamamen kaçınırlar [HLM94]; Bu durumlarda, RAID-4 bazen biraz daha kolay inşa edildiği için kullanılır.

38.8 RAID Comparison: A Summary(RAID Karşılaştırması:Özet)

Şimdi RAID düzeylerinin basitleştirilmiş karşılaştırmasını Şekil 38.8'de özetliyoruz. Analizimizi basitleştirmek için bir dizi ayrıntıyı atladığımızı unutmayın. Örneğin, yansıtılmış 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 rastgele yazma performansından biraz daha az olacaktır. Ayrıca, RAID-4/5'te eşlik diskini güncellerken, eski eşliğin ilk okunması büyük olasılıkla tam bir aramaya neden olacaktır.
ve rotasyon, ancak paritenin ikinci yazısı yalnızca rotasyonla sonuçlanacaktır. Son olarak, yansıtılmış RAID'lere sıralı G/Ç, 2× performans cezası öder diğer yaklaşımlarla karşılaştırıldığında [1].

[1] 1/2 cezası, yansıtma için naif bir okuma/yazma kalıbı varsayar; Her aynanın farklı parçalarına büyük G/Ç istekleri gönderen daha sofistike bir yaklaşım, potansiyel olarak tam bant genişliğine ulaşabilir. Nedenini bulup bulamayacağınızı 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 güvenilirliği umursamıyorsanız, şeritleme açıkça en iyisidir. Bununla birlikte, rastgele G/Ç performansı ve güvenilirliği istiyorsanız, yansıtma en iyisidir; ödediğiniz maliyet kapasite kaybıdır. Kapasite ve güvenilirlik ana hedeflerinizse, 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 mantıklı olanıdır.

38.9 Other Interesting RAID Issues(Diğer Farklı RAID Sorunları)

RAID hakkında düşünürken tartışılabilecek bir dizi farklı fikir var.

Örneğin, orijinal taksonomiden Düzey 2 ve 3 ve birden fazla disk hatasını tolere etmek için Düzey 6 [C+04] dahil olmak üzere birçok RAID tasarımı vardır. Bir disk arızalandığında RAID'in yaptığı da vardır; Bazen arızalı diski doldurmak için etrafta oturan **sıcak bir yedek(hot spare)** vardır.Hata durumunda performansa ve arızalı diskin yeniden yapılandırılması sırasındaki performansa ne olur? Ayrıca, **gizli sektör hatalarını (latent sector errors)** dikkate almak veya **bozulmayı engellemek** (**block corruption)** için daha gerçekçi hata modelleri [B + 08] ve bu tür hataları ele almak için birçok teknik vardır (ayrıntılar için veri bütünlüğü bölümüne bakın). Son olarak, RAID'i bir yazılım katmanı olarak bile oluşturabilirsiniz: bu tür yazılımlarda**, yazılım RAID(soft-ware RAID)** sistemleri daha ucuzdur, ancak tutarlı güncelleme sorunu [DAA05] dahil olmak üzere başka sorunları vardır.

38.10 Summary(Özet)

RAID'i tartıştık. RAID bir dizi bağımsız diski büyük, daha kapasitif ve daha güvenilir tek bir varlığa dönüştürür; Dolaylı olarak, bunu şeffaf bir şekilde yapar ve bu nedenle yukarıdaki donanım ve yazılım değişimden nispeten habersizdir.

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

Homework (ÖDEV) (Simulation)

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

SORULAR(QUESTIONS)

1-Bazı timely RAID eşleme testleri gerçekleştirmek için simülatörü kullanın. Farklı düzeylerde (0, 1, 4, 5) çalıştırın ve bir dizi isteğin eşlemelerini çözüp çözemeyeceğinize bakın. RAID-5 için sol simetrik ve sol asimetrik düzenler arasındaki farkı anlayıp anlayamadığınıza bakın. Yukarıdakinden farklı problemler oluşturmak için bazı farklı rastgele tohumlar kullanın.

left-symmetric left-asymmetric

0 1 2 P 0 1 2 P

4 5 P 3 3 4 P 5

8 P 6 7 6 P 7 8

./raid.py -L 5 -5 LA -c -W seq →Bu raidi çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

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

8444 1
LOGICAL READ from addr:8444 size:4096
  read  [disk 3, offset 2814]
4205 1
LOGICAL READ from addr:4205 size:4096
  read  [disk 3, offset 1401]
5112 1
LOGICAL READ from addr:5112 size:4096
  read  [disk 0, offset 1704]
7837 1
LOGICAL READ from addr:7837 size:4096
  read  [disk 1, offset 2612]
4765 1
LOGICAL READ from addr:4765 size:4096
  read  [disk 1, offset 1588]
9081 1
LOGICAL READ from addr:9081 size:4096
  read  [disk 1, offset 3027]
2818 1
LOGICAL READ from addr:2818 size:4096
  read  [disk 2, offset 939]
6183 1
LOGICAL READ from addr:6183 size:4096
  read  [disk 0, offset 2061]
9097 1
LOGICAL READ from addr:9097 size:4096
  read  [disk 1, offset 3032]
```

./raid.py -L 5 -5 LS -C W seq

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

0 1
LOGICAL READ from addr:0 size:4096
  read  [disk 0, offset 0]
1 1
LOGICAL READ from addr:1 size:4096
  read  [disk 1, offset 0]
2 1
LOGICAL READ from addr:2 size:4096
  read  [disk 2, offset 0]
3 1
LOGICAL READ from addr:3 size:4096
  read  [disk 3, offset 1]
4 1
LOGICAL READ from addr:4 size:4096
  read  [disk 0, offset 1]
5 1
LOGICAL READ from addr:5 size:4096
  read  [disk 1, offset 1]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 2, offset 2]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 2]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]
9 1
```

2-İlk problemle aynı şeyi yapın, ancak bu sefer parça boyutunu -C ile değiştirin. Yığın boyutu eşlemeleri nasıl değiştirir?

./raid.py -L 5 -5 LS -c -W seq -C 8K -n 12

```
ARG randRange 10000
ARG level 5
ARG raid5 LS
ARG reverse False
ARG timing False

0 1
LOGICAL READ from addr:0 size:4096
  read  [disk 0, offset 0]
1 1
LOGICAL READ from addr:1 size:4096
  read  [disk 0, offset 1]
2 1
LOGICAL READ from addr:2 size:4096
  read  [disk 1, offset 0]
3 1
LOGICAL READ from addr:3 size:4096
  read  [disk 1, offset 1]
4 1
LOGICAL READ from addr:4 size:4096
  read  [disk 2, offset 0]
5 1
LOGICAL READ from addr:5 size:4096
  read  [disk 2, offset 1]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 3, offset 2]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 3]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]
9 1
LOGICAL READ from addr:9 size:4096
  read  [disk 0, offset 3]
10 1
LOGICAL READ from addr:10 size:4096
  read  [disk 1, offset 2]
11 1
LOGICAL READ from addr:11 size:4096
  read  [disk 1, offset 3]
```

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

./raid.py -L 5 -5 LS -W seq -C 8K -n 12 -r →Bu raidi çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
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 READ from addr:0 size:4096
  read  [disk 0, offset 0]
1 1
LOGICAL READ from addr:1 size:4096
  read  [disk 0, offset 1]
2 1
LOGICAL READ from addr:2 size:4096
  read  [disk 1, offset 0]
3 1
LOGICAL READ from addr:3 size:4096
  read  [disk 1, offset 1]
4 1
LOGICAL READ from addr:4 size:4096
  read  [disk 2, offset 0]
5 1
LOGICAL READ from addr:5 size:4096
  read  [disk 2, offset 1]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 3, offset 2]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 3]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]
9 1
LOGICAL READ from addr:9 size:4096
```

4- Şimdi ters bayrağı kullanın, ancak bayrakla her isteğin boyutunu artırın -S. 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 sequential); RAID-4 ve RAID-5 hangi istek boyutları için G/Ç açısından çok daha verimlidir?

İsteğin boyutu arttıkça temeldeki G/Ç modeli daha verimli hale gelir. Bunun nedeni, daha büyük istek boyutlarının daha fazla verinin paralel olarak işlenmesine izin vererek performansı artırabilmesidir. RAID-4 ve RAID-5, verileri daha küçük parçalara bölmek ve birden çok diske yaymak için şeritleme kullandıklarından, verilere paralel erişime izin verdiğinden, büyük istek boyutları için verimli olacak şekilde tasarlanmıştır.

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

0 1
LOGICAL READ from addr:0 size:4096
  read  [disk 0, offset 0]
1 1
LOGICAL READ from addr:1 size:4096
  read  [disk 1, offset 0]
2 1
LOGICAL READ from addr:2 size:4096
  read  [disk 2, offset 0]
3 1
LOGICAL READ from addr:3 size:4096
  read  [disk 0, offset 1]
4 1
LOGICAL READ from addr:4 size:4096
  read  [disk 1, offset 1]
5 1
LOGICAL READ from addr:5 size:4096
  read  [disk 2, offset 1]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 0, offset 2]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 1, offset 2]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 2, offset 2]
9 1
```

```
iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$ python3 ./raid.py -L 4 -S 8k -c -W seq
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 8k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 4
ARG raid5 LS
ARG reverse False
ARG timing False

0 2
LOGICAL READ from addr:0 size:8192
  read  [disk 0, offset 0]    read  [disk 1, offset 0]
2 2
LOGICAL READ from addr:2 size:8192
  read  [disk 2, offset 0]    read  [disk 0, offset 1]
4 2
LOGICAL READ from addr:4 size:8192
  read  [disk 1, offset 1]    read  [disk 2, offset 1]
6 2
LOGICAL READ from addr:6 size:8192
  read  [disk 0, offset 2]    read  [disk 1, offset 2]
8 2
LOGICAL READ from addr:8 size:8192
  read  [disk 2, offset 2]    read  [disk 0, offset 3]
10 2
LOGICAL READ from addr:10 size:8192
  read  [disk 1, offset 3]    read  [disk 2, offset 3]
12 2
LOGICAL READ from addr:12 size:8192
  read  [disk 0, offset 4]    read  [disk 1, offset 4]
14 2
LOGICAL READ from addr:14 size:8192
  read  [disk 2, offset 4]    read  [disk 0, offset 5]
16 2
LOGICAL READ from addr:16 size:8192
  read  [disk 1, offset 5]    read  [disk 2, offset 5]
18 2
LOGICAL READ from addr:18 size:8192
```

./raid.py -L 4 -S 12k -c -W seq

```
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 12k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 4
ARG raid5 LS
ARG reverse False
ARG timing False

0 3
LOGICAL READ from addr:0 size:12288
  read  [disk 0, offset 0]    read  [disk 1, offset 0]    read  [disk 2, offset 0]
3 3
LOGICAL READ from addr:3 size:12288
  read  [disk 0, offset 1]    read  [disk 1, offset 1]    read  [disk 2, offset 1]
6 3
LOGICAL READ from addr:6 size:12288
  read  [disk 0, offset 2]    read  [disk 1, offset 2]    read  [disk 2, offset 2]
9 3
LOGICAL READ from addr:9 size:12288
  read  [disk 0, offset 3]    read  [disk 1, offset 3]    read  [disk 2, offset 3]
12 3
LOGICAL READ from addr:12 size:12288
  read  [disk 0, offset 4]    read  [disk 1, offset 4]    read  [disk 2, offset 4]
15 3
LOGICAL READ from addr:15 size:12288
  read  [disk 0, offset 5]    read  [disk 1, offset 5]    read  [disk 2, offset 5]
18 3
LOGICAL READ from addr:18 size:12288
  read  [disk 0, offset 6]    read  [disk 1, offset 6]    read  [disk 2, offset 6]
21 3
LOGICAL READ from addr:21 size:12288
  read  [disk 0, offset 7]    read  [disk 1, offset 7]    read  [disk 2, offset 7]
24 3
LOGICAL READ from addr:24 size:12288
  read  [disk 0, offset 8]    read  [disk 1, offset 8]    read  [disk 2, offset 8]
27 3
LOGICAL READ from addr:27 size:12288
  read  [disk 0, offset 9]    read  [disk 1, offset 9]    read  [disk 2, offset 9]

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

./raid.py -L 4 -S 16k -c -W seq

```
File  Edit  View  Search  Terminal  Help
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 16k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 4
ARG raid5 LS
ARG reverse False
ARG timing False

0 4
LOGICAL READ from addr:0 size:16384
  read  [disk 0, offset 0]    read  [disk 1, offset 0]    read  [disk 2, offset 0]    read  [disk 0, offset 1]
4 4
LOGICAL READ from addr:4 size:16384
  read  [disk 1, offset 1]    read  [disk 2, offset 1]    read  [disk 0, offset 2]    read  [disk 1, offset 2]
8 4
LOGICAL READ from addr:8 size:16384
  read  [disk 2, offset 2]    read  [disk 0, offset 3]    read  [disk 1, offset 3]    read  [disk 2, offset 3]
12 4
LOGICAL READ from addr:12 size:16384
  read  [disk 0, offset 4]    read  [disk 1, offset 4]    read  [disk 2, offset 4]    read  [disk 0, offset 5]
16 4
LOGICAL READ from addr:16 size:16384
  read  [disk 1, offset 5]    read  [disk 2, offset 5]    read  [disk 0, offset 6]    read  [disk 1, offset 6]
20 4
LOGICAL READ from addr:20 size:16384
  read  [disk 2, offset 6]    read  [disk 0, offset 7]    read  [disk 1, offset 7]    read  [disk 2, offset 7]
24 4
LOGICAL READ from addr:24 size:16384
  read  [disk 0, offset 8]    read  [disk 1, offset 8]    read  [disk 2, offset 8]    read  [disk 0, offset 9]
28 4
LOGICAL READ from addr:28 size:16384
  read  [disk 1, offset 9]    read  [disk 2, offset 9]    read  [disk 0, offset 10]    read  [disk 1, offset 10]
32 4
LOGICAL READ from addr:32 size:16384
  read  [disk 2, offset 10]    read  [disk 0, offset 11]    read  [disk 1, offset 11]    read  [disk 2, offset 11]
36 4
LOGICAL READ from addr:36 size:16384
  read  [disk 0, offset 12]    read  [disk 1, offset 12]    read  [disk 2, offset 12]    read  [disk 0, offset 13]

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

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

0 1
LOGICAL READ from addr:0 size:4096
  read  [disk 0, offset 0]
1 1
LOGICAL READ from addr:1 size:4096
  read  [disk 1, offset 0]
2 1
LOGICAL READ from addr:2 size:4096
  read  [disk 2, offset 0]
3 1
LOGICAL READ from addr:3 size:4096
  read  [disk 3, offset 1]
4 1
LOGICAL READ from addr:4 size:4096
  read  [disk 0, offset 1]
5 1
LOGICAL READ from addr:5 size:4096
  read  [disk 1, offset 1]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 2, offset 2]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 2]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]
9 1
LOGICAL READ from addr:9 size:4096
```

./raid.py -L 5 -S 8k -c -W seq

```
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 8k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 5
ARG raid5 LS
ARG reverse False
ARG timing False

0 2
LOGICAL READ from addr:0 size:8192
  read  [disk 0, offset 0]    read  [disk 1, offset 0]
2 2
LOGICAL READ from addr:2 size:8192
  read  [disk 2, offset 0]    read  [disk 3, offset 1]
4 2
LOGICAL READ from addr:4 size:8192
  read  [disk 0, offset 1]    read  [disk 1, offset 1]
6 2
LOGICAL READ from addr:6 size:8192
  read  [disk 2, offset 2]    read  [disk 3, offset 2]
8 2
LOGICAL READ from addr:8 size:8192
  read  [disk 0, offset 2]    read  [disk 1, offset 3]
10 2
LOGICAL READ from addr:10 size:8192
  read  [disk 2, offset 3]    read  [disk 3, offset 3]
12 2
LOGICAL READ from addr:12 size:8192
  read  [disk 0, offset 4]    read  [disk 1, offset 4]
14 2
LOGICAL READ from addr:14 size:8192
  read  [disk 2, offset 4]    read  [disk 3, offset 5]
16 2
LOGICAL READ from addr:16 size:8192
  read  [disk 0, offset 5]    read  [disk 1, offset 5]
18 2
LOGICAL READ from addr:18 size:8192
  read  [disk 2, offset 6]    read  [disk 3, offset 6]

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

mouse pointer inside or press Ctrl+G.

./raid.py -L 5 -S 12k -c -W seq

```
File  Edit  View  Search  Terminal  Help
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 12k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 5
ARG raid5 LS
ARG reverse False
ARG timing False

0 3
LOGICAL READ from addr:0 size:12288
  read  [disk 0, offset 0]    read  [disk 1, offset 0]    read  [disk 2, offset 0]
3 3
LOGICAL READ from addr:3 size:12288
  read  [disk 3, offset 1]    read  [disk 0, offset 1]    read  [disk 1, offset 1]
6 3
LOGICAL READ from addr:6 size:12288
  read  [disk 2, offset 2]    read  [disk 3, offset 2]    read  [disk 0, offset 2]
9 3
LOGICAL READ from addr:9 size:12288
  read  [disk 1, offset 3]    read  [disk 2, offset 3]    read  [disk 3, offset 3]
12 3
LOGICAL READ from addr:12 size:12288
  read  [disk 0, offset 4]    read  [disk 1, offset 4]    read  [disk 2, offset 4]
15 3
LOGICAL READ from addr:15 size:12288
  read  [disk 3, offset 5]    read  [disk 0, offset 5]    read  [disk 1, offset 5]
18 3
LOGICAL READ from addr:18 size:12288
  read  [disk 2, offset 6]    read  [disk 3, offset 6]    read  [disk 0, offset 6]
21 3
LOGICAL READ from addr:21 size:12288
  read  [disk 1, offset 7]    read  [disk 2, offset 7]    read  [disk 3, offset 7]
24 3
LOGICAL READ from addr:24 size:12288
  read  [disk 0, offset 8]    read  [disk 1, offset 8]    read  [disk 2, offset 8]
27 3
LOGICAL READ from addr:27 size:12288
  read  [disk 3, offset 9]    read  [disk 0, offset 9]    read  [disk 1, offset 9]

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

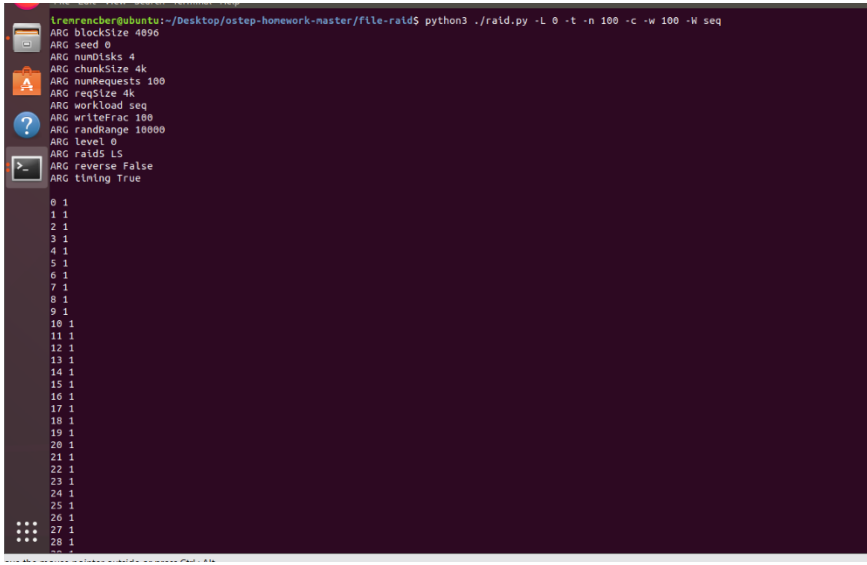
./raid.py -L 5 -S 12k -c -W seq

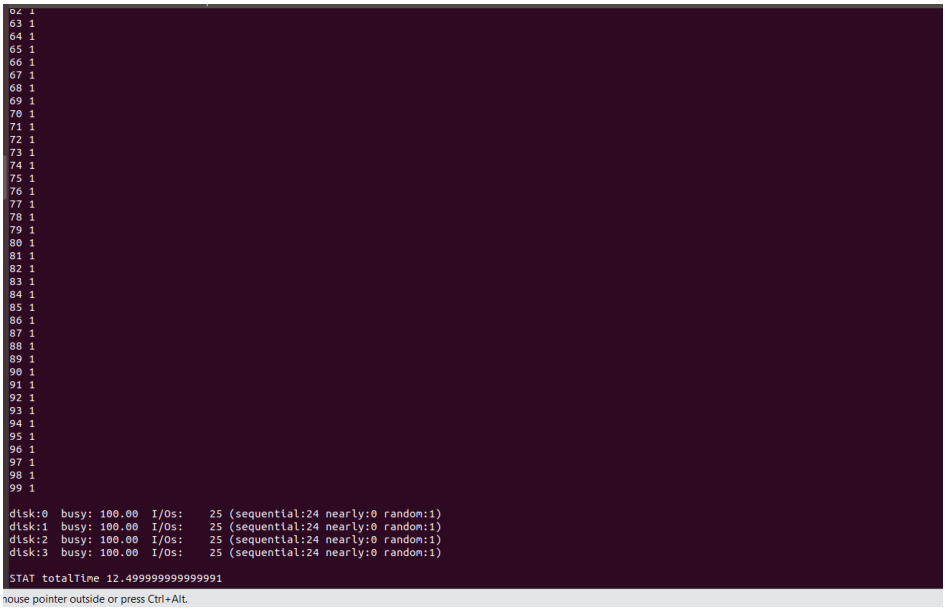
```
File  Edit  View  Search  Terminal  Help
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 16k
ARG workload seq
ARG writeFrac 0
ARG randRange 10000
ARG level 5
ARG raid5 LS
ARG reverse False
ARG timing False

0 4
LOGICAL READ from addr:0 size:16384
  read  [disk 0, offset 0]    read  [disk 1, offset 0]    read  [disk 2, offset 0]    read  [disk 3, offset 1]
4 4
LOGICAL READ from addr:4 size:16384
  read  [disk 0, offset 1]    read  [disk 1, offset 1]    read  [disk 2, offset 2]    read  [disk 3, offset 2]
8 4
LOGICAL READ from addr:8 size:16384
  read  [disk 0, offset 2]    read  [disk 1, offset 3]    read  [disk 2, offset 3]    read  [disk 3, offset 3]
12 4
LOGICAL READ from addr:12 size:16384
  read  [disk 0, offset 4]    read  [disk 1, offset 4]    read  [disk 2, offset 4]    read  [disk 3, offset 5]
16 4
LOGICAL READ from addr:16 size:16384
  read  [disk 0, offset 5]    read  [disk 1, offset 5]    read  [disk 2, offset 6]    read  [disk 3, offset 6]
20 4
LOGICAL READ from addr:20 size:16384
  read  [disk 0, offset 6]    read  [disk 1, offset 7]    read  [disk 2, offset 7]    read  [disk 3, offset 7]
24 4
LOGICAL READ from addr:24 size:16384
  read  [disk 0, offset 8]    read  [disk 1, offset 8]    read  [disk 2, offset 8]    read  [disk 3, offset 9]
28 4
LOGICAL READ from addr:28 size:16384
  read  [disk 0, offset 9]    read  [disk 1, offset 9]    read  [disk 2, offset 10]    read  [disk 3, offset 10]
32 4
LOGICAL READ from addr:32 size:16384
  read  [disk 0, offset 10]    read  [disk 1, offset 11]    read  [disk 2, offset 11]    read  [disk 3, offset 11]
36 4
LOGICAL READ from addr:36 size:16384
  read  [disk 0, offset 12]    read  [disk 1, offset 12]    read  [disk 2, offset 12]    read  [disk 3, offset 13]

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

REE
ASY
CES

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

Buradaki simulator de sadece t için sonuçları direkt gösterdim.





./raid.py -L 1 -t -n 100 -c → Bu raidi çalıştırdığımda aşağıdaki ekran görüntüsünü elde ederiz.
// 278.7

```
8007 1
227 1
1015 1
2208 1
3502 1
5036 1
1009 1
1993 1
7315 1
9184 1
6726 1
580 1
8454 1
2506 1
4423 1
4716 1
5691 1
3114 1
8376 1
5606 1
7415 1
456 1
2401 1
3522 1
3592 1
6337 1
7156 1
4144 1
15 1
3344 1
6373 1
8754 1
4144 1
7018 1
6621 1

disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:1  busy:  86.98  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:2  busy:  97.52  I/Os:    29 (sequential:0 nearly:3 random:26)
disk:3  busy:  65.23  I/Os:    19 (sequential:0 nearly:1 random:18)

STAT totalTime 278.7

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

./raid.py -L 4 -t -n 100 -c // 386.1

```
File  Edit  View  Search  Terminal  Help
8007 1
227 1
1015 1
2208 1
3502 1
5036 1
1009 1
1993 1
7315 1
9184 1
6726 1
580 1
8454 1
2506 1
4423 1
4716 1
5691 1
3114 1
8376 1
5606 1
7415 1
456 1
2401 1
3522 1
3592 1
6337 1
7156 1
4144 1
15 1
3344 1
6373 1
8754 1
4144 1
7018 1
6621 1

disk:0  busy:  78.48  I/Os:    30 (sequential:0 nearly:0 random:30)
disk:1  busy: 100.00  I/Os:    40 (sequential:0 nearly:3 random:37)
disk:2  busy:  76.46  I/Os:    30 (sequential:0 nearly:2 random:28)
disk:3  busy:   0.00  I/Os:     0 (sequential:0 nearly:0 random:0)

STAT totalTime 386.1000000000002

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

./raid.py -L 5 -t -n 100 -c // 276.5

```
8007 1
227 1
1015 1
2208 1
3502 1
5036 1
1009 1
1993 1
7315 1
9184 1
6726 1
580 1
8454 1
2506 1
4423 1
4716 1
5691 1
3114 1
8376 1
5606 1
7415 1
456 1
2401 1
3522 1
3592 1
6337 1
7156 1
4144 1
15 1
3344 1
6373 1
8754 1
4144 1
7018 1
6621 1

disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:1  busy:  95.84  I/Os:    29 (sequential:0 nearly:5 random:24)
disk:2  busy:  87.60  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:3  busy:  65.70  I/Os:    19 (sequential:0 nearly:1 random:18)

STAT totalTime 276.7

iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$
```

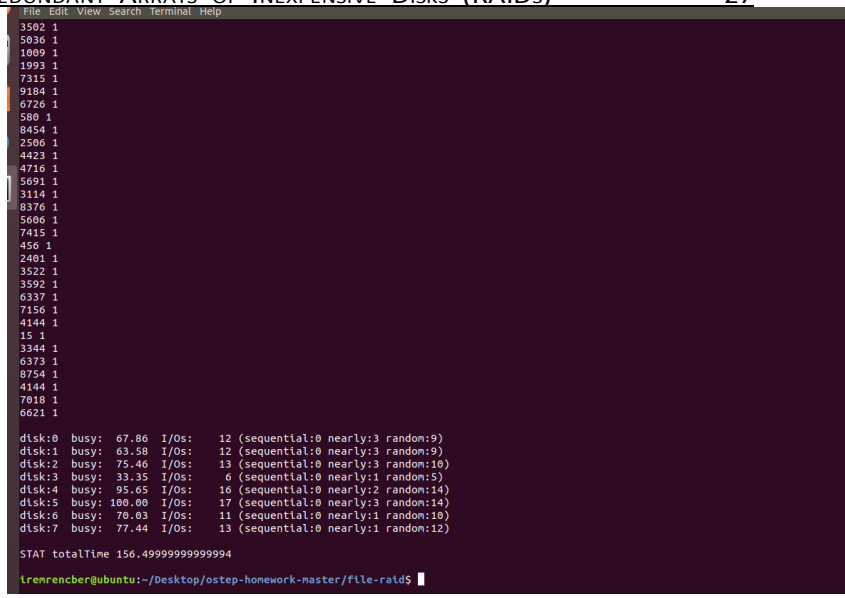
6- Yukarıdakiyle aynısını yapın, ancak disk sayısını artırın. Disk sayısı arttıkça her RAID düzeyinin performansı nasıl ölçeklenir?

Genel olarak, disk sayısı arttıkça RAID düzeylerinin performansı da artar. Ancak bu durum her RAID düzeyi için geçerli değildir ve performans artışının büyüklüğü de farklılık gösterebilir.

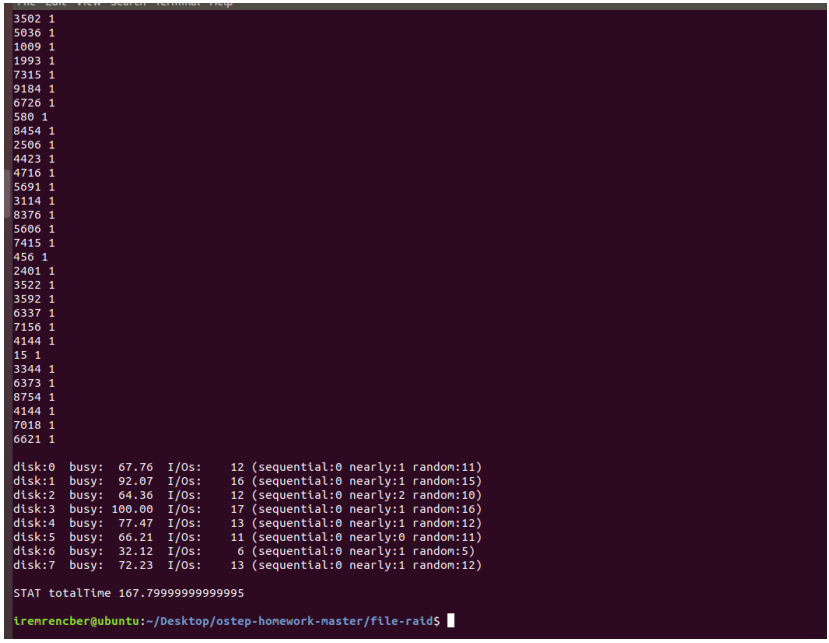
275.7 / 156.5 = 1.76
278.7 / 167.8 = 1.66
386.1 / 165.0 = 2.34
276.5 / 158.6 = 1.74

/raid.py -L 0 -t -n 100 -c -D 8 → Bu raidi çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü elde ederiz. 275.7 / 156.5 = 1.76

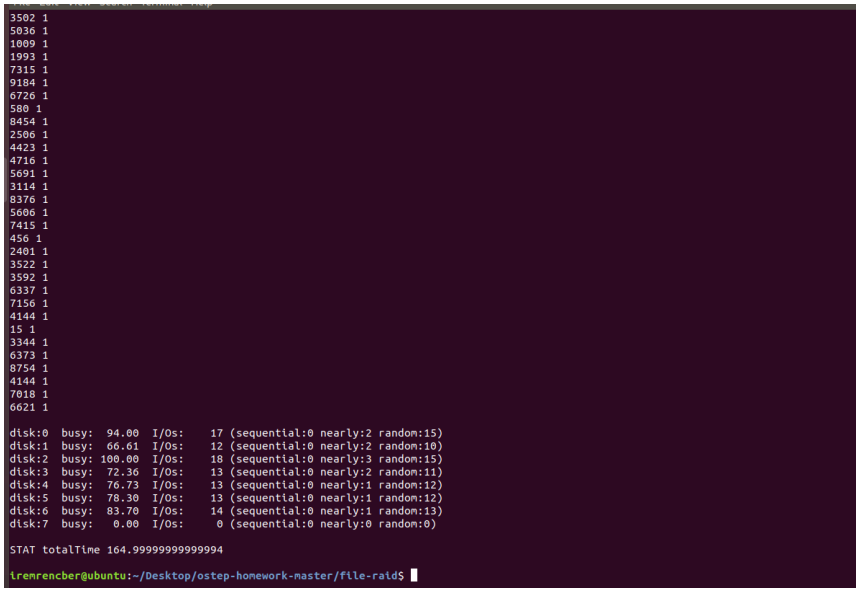
```
iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$ python3 ./raid.py -L 0 -t -n 100 -c -D 8
ARG blockSize 4096
ARG seed 0
ARG numDisks 8
ARG chunkSize 4k
ARG numRequests 100
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 10000
ARG level 0
ARG raid5 LS
ARG reverse False
ARG timing True

8444 1
4205 1
5112 1
7837 1
4765 1
9081 1
2818 1
6183 1
9097 1
8102 1
3101 1
8908 1
4721 1
4141 1
9130 1
4770 1
2604 1
5486 1
7197 1
8248 1
11 1
8676 1
3252 1
1910 1
2386 1
8031 1
804 1
5079 1
```



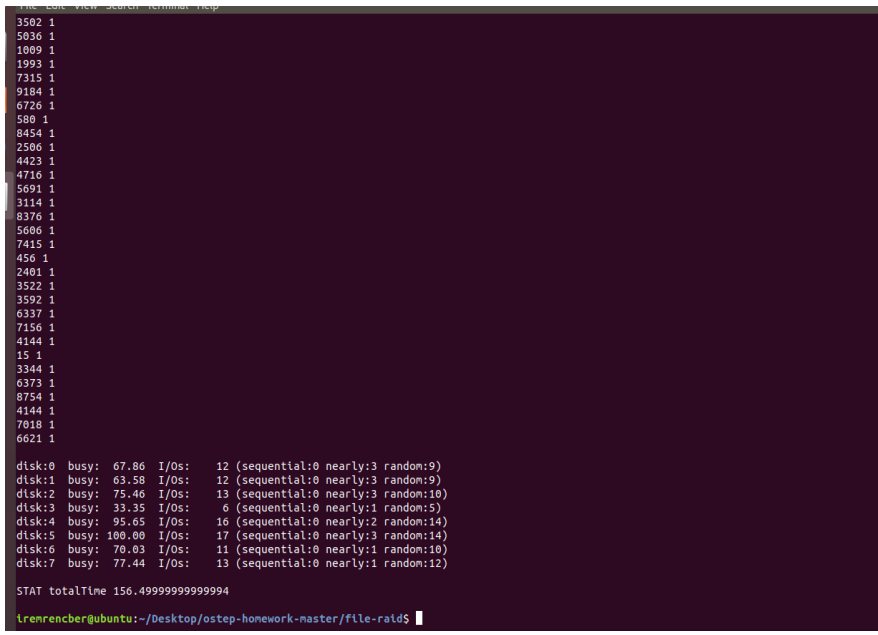
./raid.py -L 1 -t -n 100 -c -D 8 // 278.7 / 167.8 = 1.66



./raid.py -L 4 -t -n 100 -c -D 8 // 386.1 / 165.0 = 2.34



./raid.py -L 5 -t -n 100 -c -D 8 // 276.5 / 158.6 = 1.74



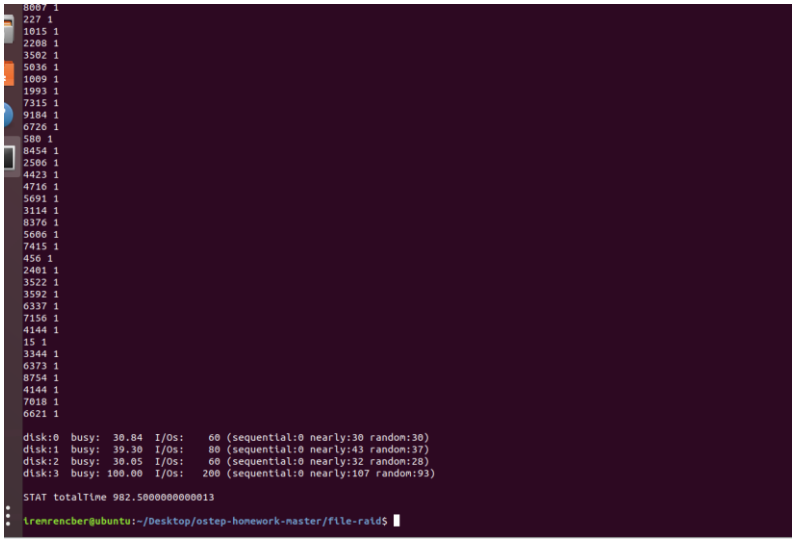
7- Yukarıdakilerin aynısını yapın, ancak okumalar yerine tüm yazmaları(-w 100 kullanın) kullanın. Her bir RAID seviyesinin performansı şimdi nasıl ölçekleniyor? 100 rastgele yazmanın iş yükünü tamamlamanın ne kadar süreceğini kabaca tahmin edebilir misiniz?

// 275.7 100 * 10 / 4
// 509.8 100 * 10 / (4 / 2)
// 275.7 / 156.5 = 1.76 100 * 10 / 8
// 509.8 / 275.7 = 1.85 100 * 10 / (8 / 2)
// 982.5 / 937.8 = 1.05
// 497.4 / 290.9 = 1.71

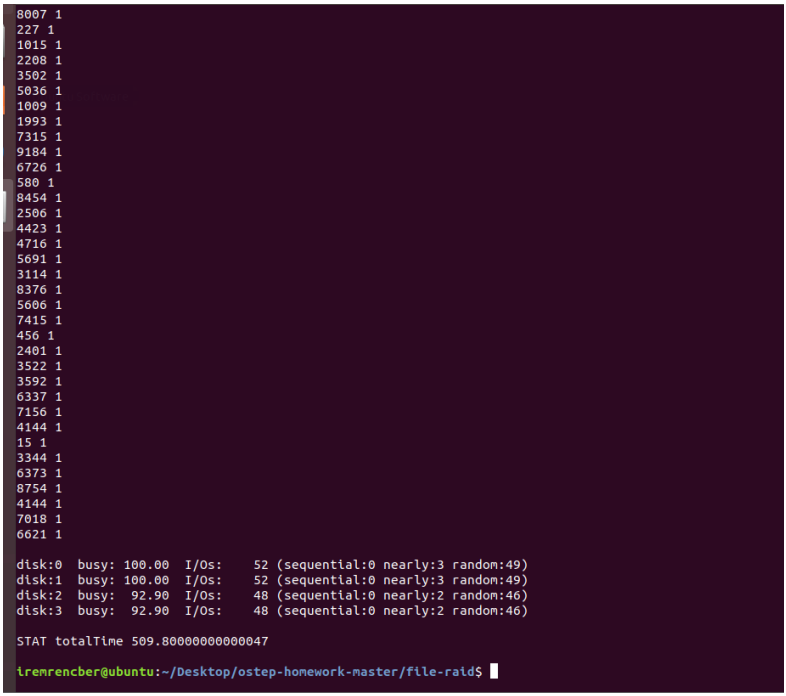
```
iremrencber@ubuntu:~/Desktop/ostep-homework-master/file-raid$ python3 ./raid.py -L 0 -t -n 100 -c -w 100
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 100
ARG reqSize 4k
ARG workload rand
ARG writeFrac 100
ARG randRange 10000
ARG level 0
ARG raid5 LS
ARG reverse False
ARG timing True

8444 1
4205 1
5112 1
7837 1
4765 1
9081 1
2818 1
6183 1
9097 1
8102 1
3101 1
8988 1
4721 1
4341 1
9130 1
4770 1
2604 1
5486 1
7197 1
8248 1
11 1
8676 1
3252 1
1910 1
2386 1
8031 1
```

```
8007 1
227 1
1015 1
2208 1
3502 1
5036 1
1009 1
1993 1
7315 1
9184 1
6726 1
580 1
8454 1
2506 1
4423 1
4716 1
5691 1
3114 1
8376 1
5606 1
7415 1
456 1
2401 1
3522 1
3592 1
6337 1
7156 1
4144 1
15 1
3344 1
6373 1
8754 1
4144 1
7018 1
6621 1

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
```

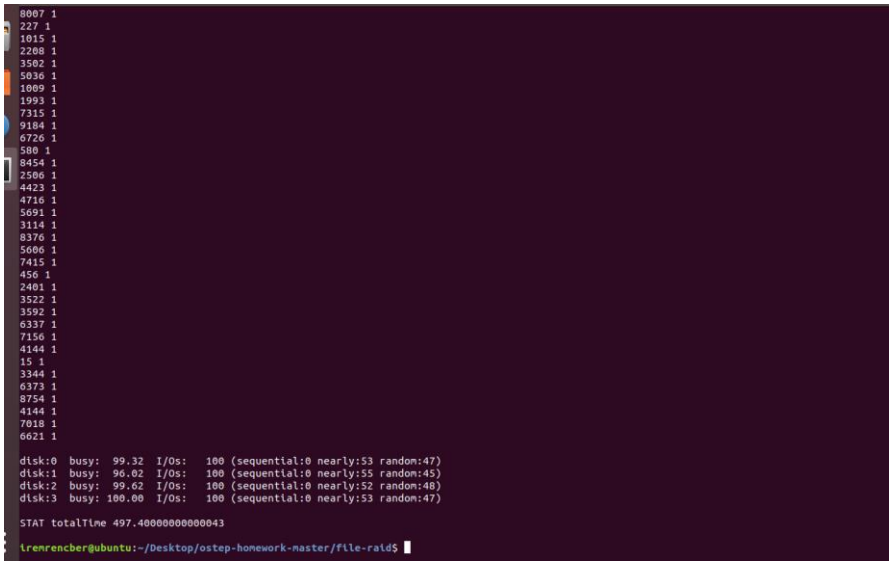
./raid.py -L 1 -t -n 100 -c -w 100



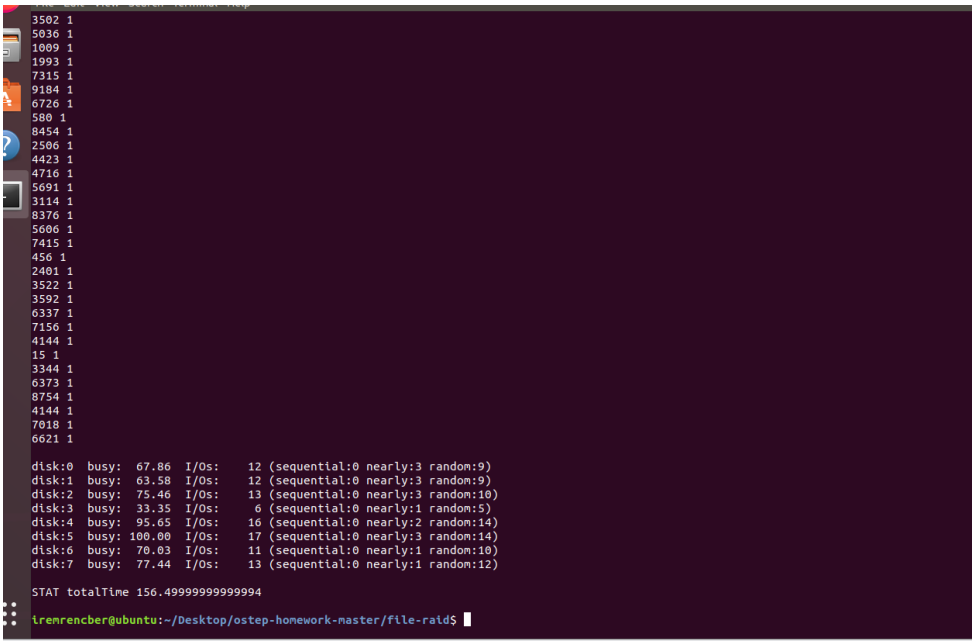
./raid.py -L 4 -t -n 100 -c -w 100



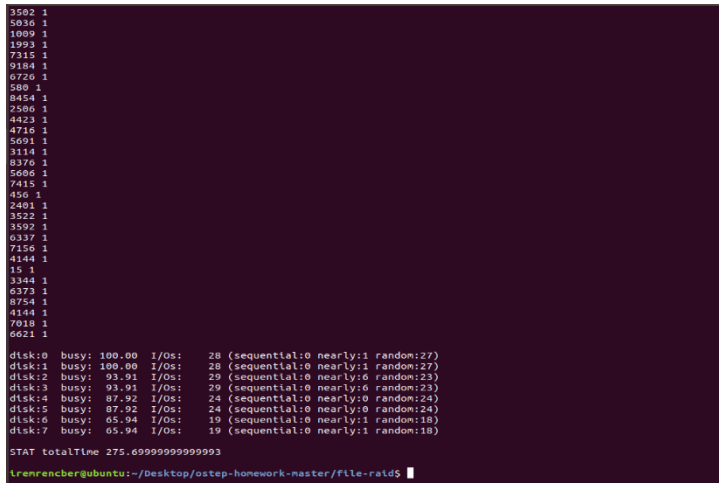
./raid.py -L 5 -t -n 100 -c -w 100 // 497.4

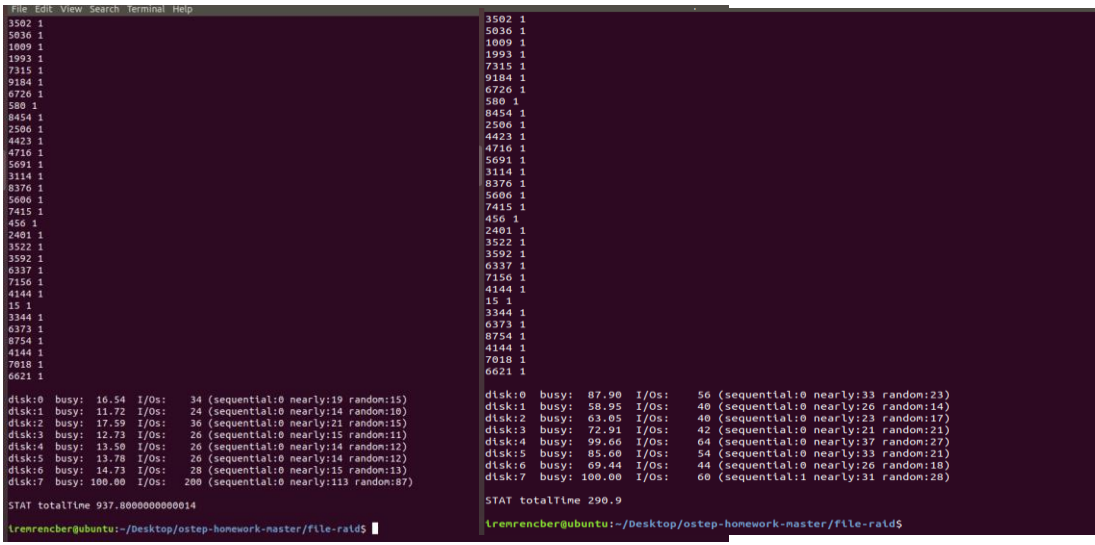


./raid.py -L 0 -t -n 100 -c -D 8 -w 100 // 275.7 / 156.5 = 1.76 100 * 10 / 8

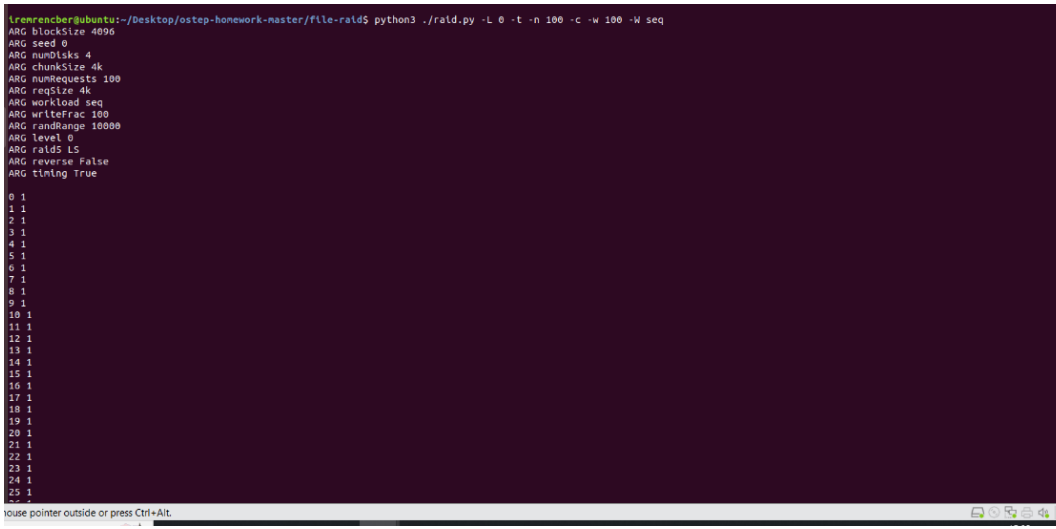


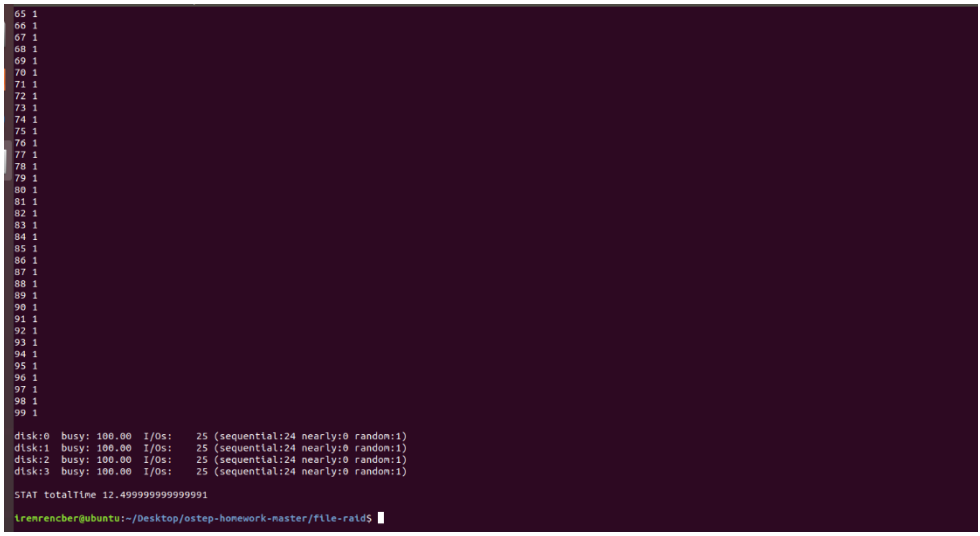
./raid.py -L 1 -t -n 100 -c -D 8 -w 100 // 509.8 / 275.7 = 1.85 100 * 10 / (8 / 2) → 1.Ekran görüntüsü elde etmek için raidi yazıyorum diğerleri içinde aynısını yapıyorum.
./raid.py -L 4 -t -n 100 -c -D 8 -w 100 // 982.5 / 937.8 = 1.05→ Sol 2.Ekran görüntüsü
./raid.py -L 5 -t -n 100 -c -D 8 -w 100 // 497.4 / 290.9 = 1.71→Sağ 2.Ekran görüntüsü



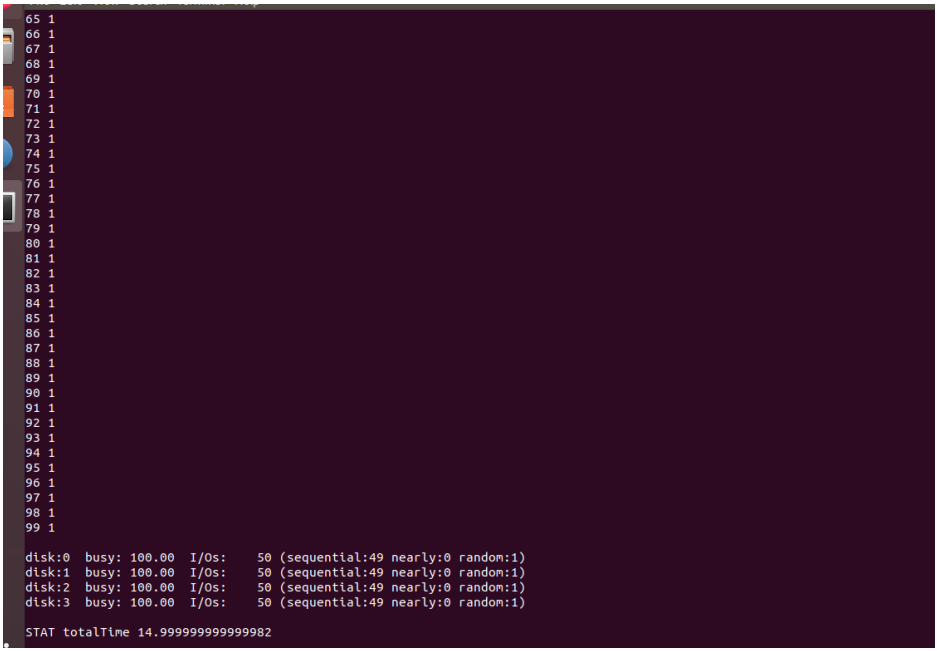


8-Zamanlama modunu son bir kez çalıştırın, ancak bu sefer sıralı bir iş yüküyle(-W sequential). Performans, RAID düzeyine ve okuma ve yazma işlemlerine göre nasıl değişir? Her isteğin boyutunu değiştirirken ne dersiniz? RAID-4 veya RAID-5 kullanırken RAID'e hangi boyutta yazmalısınız?
./raid.py -L 0 -t -n 100 -c -w 100 -W seq // 275.7 / 12.5 = 22

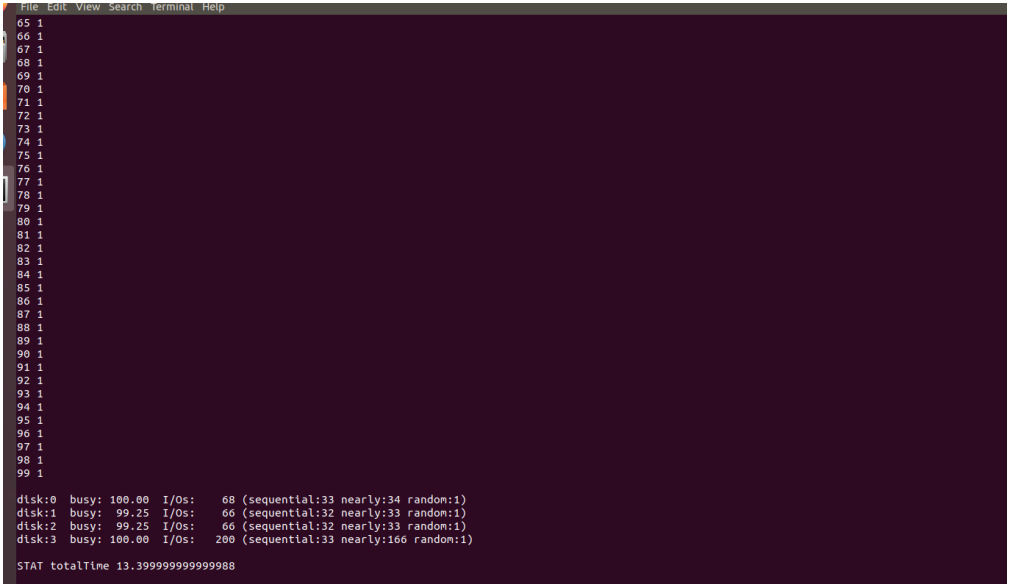




./raid.py -L 1 -t -n 100 -c -w 100 -W seq // 509.8 / 15 = 34



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



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