Bağımsız Disklerin Yedekli Dizisi (RAIDs)

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

CRUX: 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 uyum içinde kullanan bir teknik olan RAID [P+88] olarak bilinen Yedekli Ucuz Disk Dizisini tanıtıyoruz. Terim, 1980'lerin sonunda U.C.'de bir grup araştırmacı tarafından tanıtıldı. Berke-
ley (Profesörler David Patterson ve Randy Katz ve ardından öğrenci Garth Gibson liderliğinde); bu sıralarda birçok farklı araştırmacı, daha iyi bir depolama sistemi oluşturmak için birden çok disk kullanma temel fikrine aynı anda vardı [BG88, K86,K88,PB86,SG86].

Harici olarak, bir RAID bir diske benzer: okunabilen veya yazılabilen bir grup blok. Dahili olarak RAID, sistemi yönetmek için birden fazla disk, bellek (hem uçucu hem de olmayan) ve bir veya daha

fazla işlemciden oluşan karmaşık bir canavardır. Bir donanım RAID'i, bir grup diski yönetme görevi için uzmanlaşmış bir bilgisayar sistemine çok benzer.

RAID'ler, tek bir diske göre çok sayıda avantaj sunar. Bir avantaj performanstır. Birden çok diski paralel olarak kullanmak, G/Ç sürelerini büyük ölçüde hızlandırabilir. Diğer bir avantaj ise kapasitedir. Büyük veri kümeleri, büyük diskler gerektirir. Son olarak, RAID'ler güvenilirliği artırabilir; veriyi çoklu ortama yaymak tip diskler (RAID teknikleri olmadan), verileri tek bir diskin kaybına karşı savunmasız hale getirir; bir tür yedeklilik ile RAID'ler bir diskin kaybını tolere edebilir ve hiçbir sorun yokmuş gibi çalışmaya devam edebilir.

İPUCU: ŞEFFAFLIK DAĞITIMI SAĞLAR
Bir sisteme nasıl yeni işlevsellik ekleneceğini düşünürken, bu tür işlevlerin sistemin geri kalanında herhangi bir değişiklik gerektirmeyecek şekilde şeffaf bir şekilde eklenip eklenemeyeceği her zaman dikkate alınmalıdır. gerektiren bir
mevcut yazılımın tamamen yeniden yazılması (veya radikal donanım değişiklikleri), bir fikrin etki şansını azaltır. RAID mükemmel bir örnektir ve şeffaflığı kesinlikle başarısına katkıda bulunmuştur; yöneticiler bir SCSI diski yerine SCSI tabanlı bir RAID depolama dizisi kurabilir ve sistemin geri kalanının (ana bilgisayar, işletim sistemi vb.) bir bit bile değiştirmesi gerekmez.
kullanmaya başlamak için. Bu dağıtım sorununu çözerek, RAID yapıldı
ilk günden daha başarılı

Şaşırtıcı bir şekilde, RAID'ler bu avantajları onları kullanan sistemlere şeffaf bir şekilde sağlar, yani bir RAID, ana bilgisayar sistemi için büyük bir disk gibi görünür. Şeffaflığın güzelliği, elbette, kişinin bir diski RAID ile değiştirmesine ve tek bir yazılım satırını değiştirmemesine olanak sağlamasıdır; işletim sistemi ve istemci uygulamaları değiştirilmeden çalışmaya devam eder. Bu şekilde şeffaflık, konuşlandırılabilirliği büyük ölçüde artırır.RAID, kullanıcıların ve yöneticilerin yazılım uyumluluğu endişesi olmadan kullanmak üzere bir RAID koymasına olanak tanır.

Şimdi RAID'lerin bazı önemli yönlerini tartışacağız. Arayüz, hata modeli ile başlıyoruz ve ardından bir 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 ele alacağız.

38.1 Arayüz ve RAID Dahili Bileşenleri

Yukarıdaki bir dosya sistemine RAID, büyük, (umarız) hızlı ve (umarız) 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 ardından bunu yapmak için bir veya daha fazla fiziksel G/Ç göndermelidir. . Aşağıda ayrıntılı olarak tartışacağımız gibi, bu fiziksel G/Ç'lerin tam doğası RAID düzeyine bağlıdır. Ancak, basit bir örnek olarak, her bloğun iki kopyasını (her biri ayrı bir diskte) tutan bir RAID düşünün; ne zaman böyle bir ikizlenmiş RAID sistemine yazarken, RAID'inverilen her bir mantıksal G/Ç için iki fiziksel G/Ç.

Bir RAID sistemi genellikle bir ana bilgisayara standart bir bağlantıyla (örn. SCSI veya SATA) ayrı bir donanım kutusu olarak oluşturulur. Bununla birlikte dahili olarak, RAID'ler oldukça karmaşıktır; RAID'in çalışmasını yönlendirmek için aygıt yazılımını çalıştıran bir mikrodenetleyiciden, DRAM gibi geçici bellekten veri bloklarını okunurken ve yazılırken arabelleğe almaktan ve bazı durumlarda, güvenli bir şekilde ara belleğe yazmak için geçici olmayan bellek ve hatta eşlik hesaplamaları yapmak için özel mantık (aşağıda da göreceğimiz gibi, bazı RAID seviyelerinde kullanışlıdır). Yüksek düzeyde, RAID daha çok özelleşmiş bir bilgisayar sistemidir: bir işlemcisi, belleği ve diskleri vardır; ancak uygulamaları çalıştırmak yerine RAID'i çalıştırmak için tasarlanmış özel yazılımları çalıştırır.

38.2 Arıza Modeli

RAID'i anlamak ve farklı yaklaşımları karşılaştırmak için aklımızda bir hata modeli olmalıdır. RAID'ler, belirli türdeki disk hatalarını 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 basit ve arıza-durdurma arıza modeli [S84] olarak adlandırıldı. Bu modelde, bir disk tam olarak iki durumdan birinde olabilir: çalışıyor veya arızalı. Çalışan bir disk ile tüm bloklar okunabilir veya yazılabilir. Aksine, bir disk arızalandığında, kalıcı olarak kaybolduğunu varsayalım.

Fail-stop modelinin kritik bir yönü, arıza tespiti hakkında ne varsaydığıdır. Spesifik olarak, bir disk arızalandığında bunun kolayca tespit edildiğini varsayarız. Örneğin, bir RAID dizisinde, RAID denetleyici donanımının (veya yazılımının) bir disk arızalandığında hemen gözlemleyebileceğini varsayarız.

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 maalesef daha gerçekçi) disk hatalarını daha sonra ele alacağız.

38.3 Bir RAID Nasıl Değerlendirilir?

Yakında göreceğimiz gibi, bir RAID oluşturmaya yönelik bir dizi farklı yaklaşım vardır. Bu yaklaşımların her biri, güçlü ve zayıf yönlerini anlamak için değerlendirilmeye değer farklı özelliklere sahiptir.

Spesifik olarak, her bir RAID tasarımını üç eksen boyunca değerlendireceğiz. İlk eksen kapasitedir; her biri B bloklu bir dizi N disk verildiğinde, RAID istemcileri için ne kadar yararlı kapasite kullanılabilir? Fazlalık olmadan cevap N · B'dir; tersine, her bloğun iki kopyasını tutan bir sistemimiz varsa (yansıtma denir), (N · B)/2'lik bir yararlı kapasite elde ederiz. Farklı şemalar (örneğin, parite tabanlı olanlar) ikisinin arasına girme eğilimindedir.

Değerlendirmenin ikinci ekseni güvenilirliktir. 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. Performansı değerlendirmek biraz zordur, çünkü büyük ölçüde disk dizisine sunulan iş yüküne bağlıdır. Bu nedenle, performansı değerlendirmeden önce, dikkate alınması gereken bir dizi tipik iş yükü sunacağız.

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

38.4 RAID 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, performans ve kapasite açısından mükemmel bir üst sınır görevi görür ve bu nedenle anlaşılmaya değerdir.

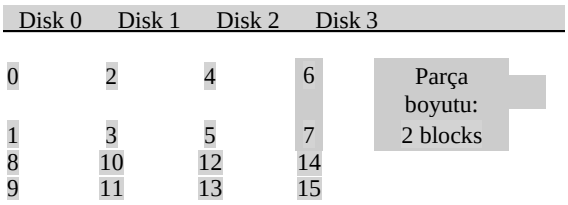
Şeritlemenin en basit biçimi, blokları sistemin diskleri boyunca şu şekilde şeritleyecektir (burada 4 diskli bir dizi varsayın):

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

Şekil 38.1: RAID-0: Basit Şeritleme

Şekil 38.1'den temel fikri elde edersiniz: dizinin bloklarını diskler boyunca sırayla sırayla dağıtın. Bu yaklaşım, istekler yapıldığında diziden en fazla paralelliği çıkarmak için tasarlanmıştır. Dizinin bitişik parçaları (örneğin, büyük, sıralı okumada olduğu gibi). 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:



Şekil 38.2: Daha Büyük Parça Boyutuyla Çizgi Oluşturma

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.

KENARA: RAID MAPPING SORUNU

RAID'in kapasitesini, güvenilirliğini ve performans özelliklerini incelemeden önce, eşleme sorunu dediğimiz konuyu bir kenara bırakıyoruz. Bu sorun, tüm RAID dizilerinde ortaya çıkar; Basitçe söylemek gerekirse, okuma veya yazma için mantıksal bir blok verildiğinde, RAID tam olarak hangi fiziksel diske ve ofsete erişileceğini nasıl bilir?

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

Disk = A / Disklerin_sayısı %'si
Ofset = A / disklerin_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 14. blok için bir istek geldiğini düşünün. 4 disk olduğu göz önüne alındığında, bu, ilgilendiğimiz diskin (% 14 4 = 2): disk 2 olduğu anlamına gelir. Kesin blok şu şekilde hesaplanır (14 / 4 = 3): blok 3. Böylece blok 14, tam olarak bulunduğu üçüncü diskin (disk 2, 0'dan başlayarak) dördüncü bloğunda (blok 3, 0'dan başlayarak) bulunmalıdır.

Farklı yığın boyutlarını desteklemek için bu denklemlerin nasıl değiştirileceğini düşünebilirsiniz. Dene! Çok zor değil.

Yığın Boyutları

Yığın boyutu çoğunlukla dizinin performansını etkiler. Örneğin, küçük bir öbek boyutu, birçok dosyanın birçok diskte sıralanacağını ve böylece okuma ve yazma işlemlerinin tek bir dosyaya paralelliğini artıracağını ima eder; ancak birden çok diskteki bloklara erişim için konumlandırma süresi

artar çünkü tüm istek için konumlandırma süresi, tüm sürücülerdeki isteklerin maksimum konumlandırma süreleri tarafından belirlenir.

Öte yandan, büyük yığın boyutu, bu tür dosya içi paralelliği azaltır ve bu nedenle, yüksek verim elde etmek için birden çok eşzamanlı isteğe güvenir. Ancak, büyük parça boyutları konumlandırma süresini azaltır; örneğin, tek bir dosya bir yığının içine sığarsa ve bu nedenle tek bir diske yerleştirilirse, ona erişim sırasında oluşan konumlandırma süresi, yalnızca tek bir diskin konumlandırma süresi olacaktır.

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

RAID-0 Analizine Geri Dön

Şimdi şeritlemenin kapasitesini, güvenilirliğini ve performansını değerlendirelim. Kapasite açısından mükemmel: her boyutta N disk verildi B blokları, şeritleme N ·B faydalı kapasite bloğu sağlar. Standdan-güvenilirlik açısından, şeritleme de mükemmeldir, ancak kötü bir şekilde: herhangi bir disk başarısızlık 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 ölçüsü dikkate alınabilir. İlki, tek istek gecikmesidir. Bir RAID'e yapılan tek bir G/Ç talebinin gecikmesini anlamak, tek bir mantıksal G/Ç işlemi sırasında ne kadar paralellik 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 odak noktası olacaktır.

Verimi daha ayrıntılı olarak anlamak için, ilgilenilen bazı iş yüklerini ortaya koymamız gerekiyor. Bu tartışma için iki tür iş yükü olduğunu varsayacağız: sıralı ve rastgele. sıralı ile iş yükü, diziye yapılan isteklerin büyük bitişik parçalar halinde geldiğini varsayarız; örneğin, x bloğunda başlayan ve blokta (x+1 MB) biten 1 MB veriye erişen bir istek (veya istek dizisi), sıralı kabul edilir. Sıralı iş yükleri birçok ortamda yaygındır (bir anahtar sözcük 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 diskteki farklı bir rasgele konuma yapıldığını varsayıyoruz. Örneğin, rastgele bir istek akışı 4KB'ye önce mantıksal adres 10'da, sonra mantıksal adres 550.000'de, ardından 20.100'de vb. erişebilir. Bir 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.

Elbette, gerçek iş yükleri o kadar basit değildir ve genellikle sıralı ve rastgele görünen bileşenlerin yanı sıra ikisi arasındaki davranışların bir karışımına sahiptir. Basitlik için, sadece bu iki olasılığı göz önünde bulunduruyoruz. Sizin de görebileceğiniz gibi sıralı ve rastgele iş yükleri, bir diskten çok farklı performans özellikleriyle sonuçlanacaktır. Sıralı erişimle, bir disk en verimli modunda çalışır, dönüşü aramak ve beklemek için çok az zaman harcar ve zamanının çoğunu veri aktarmak için harcar.

Rastgele erişimde bunun tam tersi doğrudur: çoğu zaman rotasyonu aramak ve beklemek için harcanır ve veri aktarımı için nispeten daha az zaman harcanır. Analizimizde bu farkı yakalamak için, bir diskin sıralı bir iş yükü altında S MB/s hızında ve bir sıralı iş yükü altındayken R MB/s hızında veri aktarabileceğini varsayacağız rastgele iş yükü Genel olarak, S, R'den çok daha büyüktür (yani, S ≫ R).

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

Ortalama arama süresi 7 ms Ortalama dönüş gecikmesi 3 ms

Disk aktarım hızı 50 MB/s

S'yi hesaplamak için, önce tipik bir 10 MB aktarımda zamanın nasıl harcandığını bulmamız gerekir. Önce 7 ms aramaya, ardından 3 ms dönmeye harcıyoruz. Sonunda transfer başlar; 50 MB/s'de 10 MB, aktarımda harcanan saniyenin 1/5'ine veya 200 ms'ye yol açar. Böylece, her 10 MB istek için, isteği tamamlamak için 210 ms harcıyoruz. S'yi hesaplamak için, sadece bölmemiz gerekiyor:

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

Gördüğümüz gibi, veri aktarımı için harcanan çok zaman nedeniyle S, diskin tepe bant genişliğine çok yakındır (arama ve döndürme maliyetleri amortize edilmiştir).

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

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

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

Yeniden RAID-0 Analizine Dön

Şimdi şeritleme performansını değerlendirelim. Yukarıda söylediğimiz gibi, genellikle iyidir. Gecikme açısından, örneğin, tek bloklu bir isteğin gecikmesi, tek bir diskinkiyle hemen hemen aynı olmalıdır; ne de olsa RAID-0, bu isteği disklerinden birine yönlendirecektir.

Kararlı durum sıralı iş hacmi açısından, sistemin tam bant genişliğini elde etmeyi beklerdik. Böylece verim, N (disk sayısı) ile S (tek bir diskin sıralı bant genişliği) çarpımına eşittir. Çok sayıda rasgele G/Ç için, hepsini tekrar kullanabiliriz. diskler ve böylece N · R MB/s elde edin. Aşağıda göreceğimiz gibi, bu değerler I/Os hem hesaplaması en basit olanıdır hem de üst sınır olarak hizmet edecektir.

diğer RAID düzeyleriyle karşılaştırma.

38.5 RAID Düzey 1: Yansıtma

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

Tipik bir ikizlenmiş sistemde, RAID'in her mantıksal blok için bunun iki fiziksel kopyasını tuttuğunu 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: İkizleme

Örnekte, disk 0 ve disk 1 aynı içeriğe sahiptir ve disk 2 ve disk 3 de aynıdır; veriler bu ayna çiftleri boyunca şeritlenir. Aslında, blok kopyaları diskler arasında yerleştirmenin birkaç farklı yolu olduğunu fark etmişsinizdir. Yukarıdaki düzenleme ortak bir
bir ve bazen RAID-10 (veya RAID 1+0, ayna şeridi) olarak adlandırılır
çünkü yansıtılmış çiftler (RAID-1) ve ardından bunların üzerinde şeritler (RAID-0) kullanır; diğer bir yaygın düzenleme RAID-01'dir (veya RAID 0+1, aynalı şeritler), iki büyük şeritleme (RAID-0) dizisi ve ardından bunların üzerinde ikizlemeler (RAID-1) içerir. Şimdilik, yukarıdaki düzeni varsayarak yansıtma hakkında konuşacağız.

Yansıtılmış bir diziden bir bloğu okurken, RAID'in bir seçeneği vardır: her iki kopyayı da okuyabilir. Örneğin, RAID'e mantıksal blok 5'e bir okuma verilirse, onu disk 2'den veya disk 3'ten okumak serbesttir. Bununla birlikte, bir blok yazarken böyle bir seçenek yoktur: RAID, mantıksal bloğun her iki kopyasını da güncellemelidir. verilerin güvenilirliğini korumak için. Ancak, bu yazma işlemlerinin paralel olarak gerçekleşebileceğini unutmayın; örneğin, 5. mantıksal bloğa bir yazma aynı anda 2. ve 3. disklere ilerleyebilir.

RAID-1 Analizi

RAID-1'i değerlendirelim. Kapasite açısından RAID-1 pahalıdır; yansıtma düzeyi = 2 ile, en yüksek yararlı kapasitemizin yalnızca yarısını elde ederiz. B bloklu N disk ile RAID-1'in faydalı kapasitesi (N · B)/2'dir. Güvenilirlik açısından, RAID-1 iyi iş çıkarıyor. Herhangi bir diskin arızasını tolere edebilir. RAID-1'in biraz şansla bundan daha iyisini yapabileceğini de fark edebilirsiniz. Yukarıdaki şekilde, disk 0 ve disk 2'nin ikisinin de başarısız olduğunu hayal edin. Böyle bir durumda veri kaybı olmaz! Daha genel olarak, ikizlenmiş bir sistem (iki yansıtma seviyesiyle) 1 disk arızasını kesin olarak ve hangi disklerin arızalandığına bağlı olarak N/2 arızaya kadar tolere edebilir. Pratikte genellikle bu tür şeyleri şansa bırakmayı sevmeyiz; bu nedenle çoğu kişi, yansıtmanın tek bir arızayı halletmek 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,

BİR KENARA: RAID TUTARLI GÜNCELLEME SORUNU

RAID-1'i analiz etmeden önce, tutarlı güncelleme sorunu [DAA05] olarak bilinen herhangi bir çok diskli RAID sisteminde ortaya çıkan bir sorunu tartışalım. Sorun, tek bir mantıksal işlem sırasında birden çok diski güncellemesi gereken herhangi bir RAID'e yazma sırasında ortaya çıkar. Bu durumda bize yansıtılmış bir disk dizisi düşündüğümüzü varsayalım.Yazmanın RAID'e verildiğini ve ardından RAID'in bunun iki diske, disk 0 ve disk 1'e yazılması gerektiğine karar verdiğini hayal edin. RAID daha sonra diske yazma 0'ı verir, ancak RAID diske istek gönderemeden hemen önce 1, 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 olmasıdır; disk 0'daki kopya yeni sürümdür ve disk 1'deki kopya eskidir. Olmasını istediğimiz şey, her iki diskin durumunun atomik olarak değişmesi, yani ya her ikisi de yeni sürüm olarak bitmeli ya da hiçbiri.Bu sorunu çözmenin genel yolu, bazılarının önden yazma günlüğünü kullanmaktır.Bunu yapmadan önce RAID'in ne yapmak üzere olduğunu (yani, iki diski belirli bir veri parçasıyla güncellemek) kaydetmek naziktir. Bu yaklaşımı benimseyerek, bir çarpışma durumunda doğru şeyin olmasını sağlayabiliriz; bekleyen tüm işlemleri yeniden oynatan bir kurtarma prosedürü çalıştırarak RAID, iki aynalanmış 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. en kötü durum araması ve iki isteğin dönüş gecikmesi ve dolayısıyla (ortalama olarak) tek bir diske yazmaya göre biraz daha yüksek olacaktır.

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 (yukarıdaki şekilde) yazdığımızda, RAID dahili olarak bunu hem disk 0'a hem de disk 1'e yazar.

yansıtılmış bir diziye sıralı yazma sırasında elde edilen değer (N · S) veya yarısıdır.

tepe bant genişliği.

Ne yazık ki sıralı bir okumada 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. 4, 5, 6 ve 7'nin okumalarını 0, 2, 1 disklerine vererek devam ediyoruz, ve sırasıyla 3. 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 böylece, ardışık okuma yalnızca (N · S) MB/sn'lik bir bant genişliği elde edecektir.

Rastgele okumalar, ikizlenmiş bir RAID için en iyi durumdur. Bu durumda, biz okumaları tüm disklere dağıtabilir ve böylece tam olası bant genişliğini elde edebilir. Böylece, rastgele okumalar için RAID-1, N · R MB/s sunar. Son olarak, rastgele yazma işlemleri beklediğiniz gibi çalışır: N ·R MB/sn. Her mantıksal yazma, iki fiziksel yazmaya dönüşmelidir ve böylece 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 şeritleme ile 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 olarak bilinen bir disk dizisine artıklık eklemenin farklı bir yöntemini sunuyoruz. Pariteye dayalı yaklaşımlar daha az kapasite kullanmaya çalışır ve böylece aynalı sistemler tarafından ödenen 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, çift sayıda bit varsa, tüm bu bitlerin XOR'u 0 döndürür. Bitlerde 1, tek sayıda 1 varsa 1. 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 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: 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.

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 yeterlidir (dahil XOR'd eşlik biti) ve doğru yanıtı yeniden oluşturun. Özellikle, varsayalım C2 sütunundaki ilk satırın değeri kaybolur (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 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 XORing'den 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 bit-bazlı XOR'un sonucunu parite bloğundaki karşılık gelen bit yuvasına yerleştirin. Örneğin, 4 bit boyutunda bloklarımız olsaydı (evet, bu hala 4 KB'lik 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 1 kullanır koruduğu her disk grubu için eşlik bilgileri için disk. Dolayısıyla, bir RAID grubu için yararlı kapasitemiz (N − 1) · 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 böylece etkin bir tepe bant genişliği sağlar. (N − 1) · S MB/s (kolay bir durum).

Sıralı yazmaların performansını anlamak için önce nasıl yapıldığını anlayın. 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 şeritli 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 aynı zamanda (N − 1) · S MB/sn. Eşlik diski, işlem sırasında sürekli kullanımda olmasına rağmen işlem, müşteri 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 rasgele yazmalar en önemlileri sunar.
RAID-4 için ilginç bir durum. 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 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. Böylece, çıkarıcı parite yöntemi.
Örneğin, bu bit dizisini hayal edin (4 veri biti, bir eşlik):

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

C2 bitinin üzerine C2new 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 çevirmeliyiz, yani eğer (Pold == 1),

Pnew 0 olarak ayarlanacaktır; eğer (Pold == 0), Pnew 1 olarak set edilecek. Tüm bu karışıklığı XOR ile düzgün bir şekilde ifade edebiliriz (burada ⊕, XOR operatörüdür):

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

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 farklı olacaktır. eski bloktan daha fazla ve dolayısıyla yeni parite bloğu da 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 Yazma

Şimdi, şu adreste RAID-4'e gönderilen 2 küçük yazı olduğunu hayal edin: ∗yaklaşık aynı zamanda, 4. ve 13. bloklara (şemada ile işaretlenmiştir). Bu disklerin verileri 0 ve 1 disklerindedir ve 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. Umarım, sorun artık açıktır: eşlik diski bu tür iş yükü altında bir darboğazdır; bu nedenle bazen buna parite tabanlı RAID'ler için küçük yazma sorunu diyoruz. Böylece, veri disklerine erişilebilse bile paralel, 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 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: Döndürülmüş Parite ile RAID-5

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 ve disk 4'e (blok 1 ve paritesi için) ve disk 0 ve disk 2'ye (blok 10 ve paritesi için) yapılan isteklere dönüşecektir. Böylece paralel ilerleyebilirler. Aslında, çok sayıda rasgele istek verildiğinde, tüm diskleri eşit şekilde meşgul tutabileceğimizi varsayabiliriz. Eğer durum buysa, o zaman küçük yazmalar için toplam bant genişliğimiz N · R MB/s olacaktır. Faktör dört kaybın nedeni, her RAID-5 yazmanın hala toplam 4 tane oluşturmasıdır. Basitçe eşlik tabanlı RAID kullanmanın maliyeti olan G/Ç işlemleri.

	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	$N \cdot R$	$(N/2) \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{1}{4} \cdot R$
Latency				
Read	T	T	T	T
Write	T	T	$2T$	$2T$

Şekil 38.8: RAID Kapasitesi, Güvenilirlik ve Performans

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, böylece küçük yazma sorununu tamamen ortadan kaldıran sistemlerdir [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, RAID-4/5'te eşlik diskini güncellerken, eski eşliğin ilk okuması muhtemelen tam aramaya neden olacaktır.
ve dönüş, ancak paritenin ikinci yazılması yalnızca dönüşle sonuçlanacaktır. Son olarak, yansıtılmış RAID'lere yönelik sıralı G/Ç, 2 kat performans cezası öder
diğer yaklaşımlarla karşılaştırıldığında1.

11/2 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. 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 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 ve 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 vardır. Ne Başarısız durumdaki performansa ve arızalı diskin yeniden oluşturulması sırasındaki performansa ne oluyor? Gizli sektör hatalarını hesaba katmak veya bozulmayı engellemek [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 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 yapıyor ve bu nedenle yukarıdaki donanım ve yazılım, değişiklikten nispeten habersiz.

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.

Referanslar

[B+08] "An Analysis of Data Corruption in the Storage Stack", yazan Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci- Dusseau. FAST '08, San Jose, CA, Şubat 2008. Disklerin gerçekte ne sıklıkta verilerinizi bozduğunu analiz eden kendi çalışmamız. Sık değil ama bazen! Ve bu nedenle, güvenilir bir depolama sisteminin dikkate alması gereken bir şey.

[BJ88] D. Bitton ve J. Gray tarafından yazılan “Disk Gölgeleme”. VLDB 1988. "Gölgeleme" olarak adlandırılan yansıtmayı tartışan ilk makalelerden biri.

[CL95] Peter M. Chen ve Edward K. Lee tarafından yazılan “RAID seviye 5 disk dizisinde şeritleme”. SIGMETRICS 1995. Bir RAID-5 disk dizisindeki bazı önemli parametrelerin güzel bir analizi.

[C+04] "Row-Diagonal Parity for Double Disk Failure Correction", yazan P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, S. Sankar. FAST '04, Şubat 2004. Eşlik için iki diskli bir RAID sistemiyle ilgili ilk makale olmasa da, söz konusu fikrin yeni ve oldukça anlaşılır bir versiyonudur. Daha fazlasını öğrenmek için okuyun. [DAA05] Timothy E. Denehy, A. Arpaci-Dusseau, R. Arpaci-Dusseau tarafından yazılan “Yazılım RAID'i için Dergi Kılavuzluğunda Yeniden Eşitleme”. FAST 2005. Tutarlı güncelleme sorunu üzerine kendi çalışmamız. Burada, yukarıdaki dosya sisteminin günlük kaydı makinesini altındaki yazılım RAID ile entegre ederek Yazılım RAID için çözüyoruz.

[HLM94] Dave Hitz, James Lau, Michael Malcolm tarafından yazılan “NFS Dosya Sunucusu Cihazı için Dosya Sistemi Tasarımı”. USENIX Kış 1994, San Francisco, California, 1994. Depolamada dönüm noktası niteliğindeki bir ürünü, her yere yazılabilen dosya düzenini veya NetApp dosya sunucusunun temelini oluşturan WAFL dosya sistemini tanıtan seyrek kağıt.

[K86] M.Y. Kim. Bilgisayarlarda IEEE İşlemleri, Cilt C-35: 11, Kasım 1986. RAID ile ilgili en eski çalışmalardan bazıları burada bulunur.

[K88] “Küçük Disk Dizileri – Yüksek Performansa Gelişen Yaklaşım”, F. Kurzweil. Bahar COMPCON '88'de sunum, 1 Mart 1988, San Francisco, California. Başka bir erken RAID referansı.

[P+88] “Yedek Ucuz Disk Dizileri” D. Patterson, G. Gibson, R. Katz. SIG- MOD 1988. Bu, ünlü yazarlar Patterson, Gibson ve Katz tarafından yazılan RAID makalesi olarak kabul edilir. Kağıt o zamandan beri birçok zaman testi ödülü kazandı ve RAID adının kendisi de dahil olmak üzere RAID çağını başlattı!

[PB86] “Paralel İkincil Depolama Sistemlerinde Hata Toleransı Sağlama”, A. Park, K. Balasubramaniam. Department of Computer Science, Princeton, CS-TR-O57-86, Kasım 1986. RAID üzerine bir başka erken çalışma.

[SG86] “Disk Şeritleme”, K. Salem, H. Garcia-Molina. IEEE Uluslararası Veri Mühendisliği Konferansı, 1986. Ve evet, başka bir erken dönem RAID çalışması. SIGMOD'da RAID makalesi yayınlandığında tahtadan çıkmış bir sürü bunlardan var.

[S84] "Bizans Generalleri İş Başında: Fail-Stop İşlemcileri Uygulamak", F.B. Schneider. ACM Transactions on Computer Systems, 2(2):145154, Mayıs 1984. Son olarak, RAID ile ilgili olmayan bir makale! Bu makale aslında sistemlerin nasıl başarısız olduğu ve bir şeyin arıza-durdurma şeklinde davranmasının nasıl sağlanacağı ile ilgilidir.

Ödev (Simülasyon)

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

Sorular

1. Bazı temel 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.
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 bayrağı kullanın, ancak -S bayrağıyla her isteğin boyutunu 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 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?

CEVAPLAR

2.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 raid.py -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 0
ARG raid5 LS
ARG reverse False
ARG timing False

8444 1
LOGICAL READ from addr:8444 size:4096
  read  [disk 0, offset 2111]

4205 1
LOGICAL READ from addr:4205 size:4096
  read  [disk 1, offset 1051]

5112 1
LOGICAL READ from addr:5112 size:4096
  read  [disk 0, offset 1278]

7837 1
LOGICAL READ from addr:7837 size:4096
  read  [disk 1, offset 1959]

4765 1
LOGICAL READ from addr:4765 size:4096
  read  [disk 1, offset 1191]

9081 1
LOGICAL READ from addr:9081 size:4096
  read  [disk 1, offset 2270]

2818 1
LOGICAL READ from addr:2818 size:4096
  read  [disk 2, offset 704]

6183 1
LOGICAL READ from addr:6183 size:4096
  read  [disk 3, offset 1545]

9097 1
LOGICAL READ from addr:9097 size:4096
  read  [disk 1, offset 2274]

8102 1
LOGICAL READ from addr:8102 size:4096
  read  [disk 2, offset 2025]
```

3.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 raid.py -r
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 0
ARG raid5 LS
ARG reverse True
ARG timing False

8444 1
LOGICAL OPERATION is ?
  read  [disk 0, offset 2111]

4205 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 1051]

5112 1
LOGICAL OPERATION is ?
  read  [disk 0, offset 1278]

7837 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 1959]

4765 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 1191]

9081 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 2270]

2818 1
LOGICAL OPERATION is ?
  read  [disk 2, offset 704]

6183 1
LOGICAL OPERATION is ?
  read  [disk 3, offset 1545]

9097 1
LOGICAL OPERATION is ?
  read  [disk 1, offset 2274]

8102 1
LOGICAL OPERATION is ?
  read  [disk 2, offset 2025]
```

4.a.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 -s  raid.py
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 0
ARG raid5 LS
ARG reverse False
ARG timing False

8444 1
LOGICAL READ from addr:8444 size:4096
  Physical reads/writes?

4205 1
LOGICAL READ from addr:4205 size:4096
  Physical reads/writes?

5112 1
LOGICAL READ from addr:5112 size:4096
  Physical reads/writes?

7837 1
LOGICAL READ from addr:7837 size:4096
  Physical reads/writes?

4765 1
LOGICAL READ from addr:4765 size:4096
  Physical reads/writes?

9081 1
LOGICAL READ from addr:9081 size:4096
  Physical reads/writes?

2818 1
LOGICAL READ from addr:2818 size:4096
  Physical reads/writes?

6183 1
LOGICAL READ from addr:6183 size:4096
  Physical reads/writes?

9097 1
LOGICAL READ from addr:9097 size:4096
  Physical reads/writes?

8102 1
LOGICAL READ from addr:8102 size:4096
  Physical reads/writes?
```

4.b.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 raid.py -W sequential
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 4k
ARG workload sequential
ARG writeFrac 0
ARG randRange 10000
ARG level 0
ARG raid5 LS
ARG reverse False
ARG timing False

0 1
LOGICAL READ from addr:0 size:4096
  Physical reads/writes?

1 1
LOGICAL READ from addr:1 size:4096
  Physical reads/writes?

2 1
LOGICAL READ from addr:2 size:4096
  Physical reads/writes?

3 1
LOGICAL READ from addr:3 size:4096
  Physical reads/writes?

4 1
LOGICAL READ from addr:4 size:4096
  Physical reads/writes?

5 1
LOGICAL READ from addr:5 size:4096
  Physical reads/writes?

6 1
LOGICAL READ from addr:6 size:4096
  Physical reads/writes?

7 1
LOGICAL READ from addr:7 size:4096
  Physical reads/writes?

8 1
LOGICAL READ from addr:8 size:4096
  Physical reads/writes?

9 1
LOGICAL READ from addr:9 size:4096
  Physical reads/writes?
```

5.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 raid.py -t
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 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

Estimate how long the workload should take to complete.
- Roughly how many requests should each disk receive?
- How many requests are random, how many sequential?
```

7.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 raid.py -w 100
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 4k
ARG workload rand
ARG writeFrac 100
ARG randRange 10000
ARG level 0
ARG raid5 LS
ARG reverse False
ARG timing False

8444 1
LOGICAL WRITE to  addr:8444 size:4096
  Physical reads/writes?

4205 1
LOGICAL WRITE to  addr:4205 size:4096
  Physical reads/writes?

5112 1
LOGICAL WRITE to  addr:5112 size:4096
  Physical reads/writes?

7837 1
LOGICAL WRITE to  addr:7837 size:4096
  Physical reads/writes?

4765 1
LOGICAL WRITE to  addr:4765 size:4096
  Physical reads/writes?

9081 1
LOGICAL WRITE to  addr:9081 size:4096
  Physical reads/writes?

2818 1
LOGICAL WRITE to  addr:2818 size:4096
  Physical reads/writes?

6183 1
LOGICAL WRITE to  addr:6183 size:4096
  Physical reads/writes?

9097 1
LOGICAL WRITE to  addr:9097 size:4096
  Physical reads/writes?

8102 1
LOGICAL WRITE to  addr:8102 size:4096
  Physical reads/writes?
```

8.

```
gulsen@berkan-virtual-machine:~/Desktop/ostep/file-raid$ python3 raid.py -w 100
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 10
ARG reqSize 4k
ARG workload rand
ARG writeFrac 100
ARG randRange 10000
ARG level 0
ARG raid5 LS
ARG reverse False
ARG timing False

8444 1
LOGICAL WRITE to  addr:8444 size:4096
  Physical reads/writes?

4205 1
LOGICAL WRITE to  addr:4205 size:4096
  Physical reads/writes?

5112 1
LOGICAL WRITE to  addr:5112 size:4096
  Physical reads/writes?

7837 1
LOGICAL WRITE to  addr:7837 size:4096
  Physical reads/writes?

4765 1
LOGICAL WRITE to  addr:4765 size:4096
  Physical reads/writes?

9081 1
LOGICAL WRITE to  addr:9081 size:4096
  Physical reads/writes?

2818 1
LOGICAL WRITE to  addr:2818 size:4096
  Physical reads/writes?

6183 1
LOGICAL WRITE to  addr:6183 size:4096
  Physical reads/writes?

9097 1
LOGICAL WRITE to  addr:9097 size:4096
  Physical reads/writes?

8102 1
LOGICAL WRITE to  addr:8102 size:4096
  Physical reads/writes?
```