37.BÖLÜM:

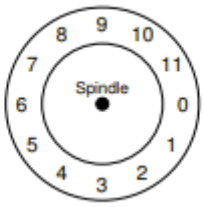
Sabit disk sürücüleri

Son bölüm, bir G/Ç cihazının genel konseptini tanıttı ve işletim sisteminin böyle bir sistemle nasıl etkileşim kurabileceğini gösterdi. Bu bölümde, özellikle bir cihaz hakkında daha fazla ayrıntıya giriyoruz: Sabit disk sürücüsü(hard disk drive.) Bu sürücüler, kalıcı veri depolamanın ana biçimi olmuştur. Onlarca yıldır bilgisayar sistemleri ve dosya sistemi teknolojisindeki (çok yakında) gelişmelerin çoğu, bunların davranışlarına dayanmaktadır. Bu nedenle, onu yöneten dosya sistemi yazılımını oluşturmadan önce bir diskin işleyişinin ayrıntılarını anlamaya değer. Bu ayrıntıların çoğu, Ruemmler ve Wilkes [RW92] ve Anderson'ın mükemmel makalelerinde mevcuttur. Dykes ve Riedel [ADR03].

CRUX: DİSKTE VERİLER NASIL DEPOLANIR VE ERİŞİLİR Modern sabit disk sürücüleri verileri nasıl depolar? Arayüz nedir? Veriler gerçekte nasıl düzenleniyor ve bunlara erişiliyor? Disk zamanlaması performansı nasıl artırır?

37.1 Arayüz:

Modern bir disk sürücüsünün arayüzünü anlayarak başlayalım. Tüm modern sürücüler için temel arabirim basittir. Sürücü, her biri okunabilen veya yazılabilen çok sayıda sektörden (512 bayt blok) oluşur. Sektörler, n sektörlü bir diskte 0'dan n - 1'e kadar numaralandırılmıştır. Böylece diski bir sektör dizisi olarak görebiliriz; 0 ila n - 1 bu nedenle sürücünün adres alanıdır(address space.) Çok sektörlü işlemler mümkündür; aslında birçok dosya sistemi aynı anda 4 KB (veya daha fazla) okur veya yazar. Bununla birlikte, diski güncellerken, sürücü üreticilerinin sağladığı tek garanti, tek bir 512 baytlık yazmanın atomik(atomic) olmasıdır (yani, atomik ya bütünüyle tamamlanır ya da hiç tamamlanmaz); bu nedenle, zamansız bir güç kaybı meydana gelirse, daha büyük bir yazmanın yalnızca bir kısmı tamamlanabilir (bazen yırtık yazma (torn write) olarak adlandırılır).



Şekil 37.1: Yalnızca Tek İzli Bir Disk

Çoğu disk sürücüsü istemcisinin yaptığı, ancak doğrudan arabirimde belirtilmeyen bazı varsayımlar vardır; Schlosser ve Ganger bunu disk sürücülerinin "yazılı olmayan sözleşmesi" [SG04] olarak adlandırdılar. Spesifik olarak, sürücünün adres alanında birbirine yakın iki bloğa erişmenin, birbirinden uzak iki bloğa erişmekten daha hızlı olacağı varsayılabilir. Bitişik bir

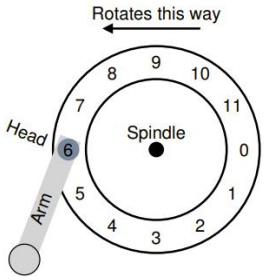
parçadaki (yani sıralı bir okuma veya yazma) bloklara erişmenin en hızlı erişim modu olduğu ve genellikle herhangi bir rasgele erişim modelinden çok daha hızlı olduğu da varsayılabilir.

37.2 Temel Geometri

Modern bir diskin bazı bileşenlerini anlamaya başlayalım. Üzerinde manyetik değişiklikler yaratarak verilerin ısrarla depolandığı dairesel, sert bir yüzey olan bir tabakla başlıyoruz. Bir diskin bir veya daha fazla plakası olabilir; her tabağın her birine yüzey adı verilen 2 kenarı vardır. Bu plakalar genellikle bazı sert malzemelerden (alüminyum gibi) yapılır ve daha sonra sürücünün gücü kapalıyken bile bitleri kalıcı olarak depolamasını sağlayan ince bir manyetik katmanla kaplanır. Plakaların tümü, plakaları sabit (sabit) bir hızda (sürücü açıkken) etrafında döndüren bir motora bağlı olan iş mili etrafında birbirine bağlıdır. Dönüş hızı genellikle dakika başına dönüş (RPM)(rotations per minute (RPM)) cinsinden ölçülür ve tipik modern değerler 7.200 RPM ila 15.000 RPM aralığındadır. Genellikle tek bir dönüşün süresiyle ilgileneceğimizi unutmayın, örneğin, 10.000 Rpm'de dönen bir sürücü, tek bir dönüşün yaklaşık 6 milisaniye (6 MS) sürdüğü anlamına gelir. Veriler, sektörlerin eşmerkezle dairelerinde her yüzeyde kodlanmıştır; böyle bir eşmerkezle daireye iz(track) diyoruz. Tek bir yüzey, bir insan saçı genişliğine sığan yüzlerce iz ile sıkıca bir araya getirilmiş binlerce ve binlerce iz içerir. Yüzeyden okumak ve yazmak için, diskteki manyetik kalıpları algılamamıza (yani okumamıza) veya onlarda bir değişikliğe neden olmamıza (yani yazmamıza) izin veren bir mekanizmaya ihtiyacımız var. Bu okuma ve yazma işlemi disk kafası (disk head) tarafından gerçekleştirilir; sürücünün yüzeyi başına böyle bir kafa vardır. Disk kafası tek bir disk koluna (disk arm) bağlıdır.

37.3 Basit Bir Disk Sürücüsü

Her seferinde bir iz olacak bir model oluşturarak disklerin nasıl çalıştığını anlayalım. Tek izli basit bir diskimiz olduğunu varsayalım (Şekil 37.1).



Şekil 37.2: Tek İz Artı Bir Başlık

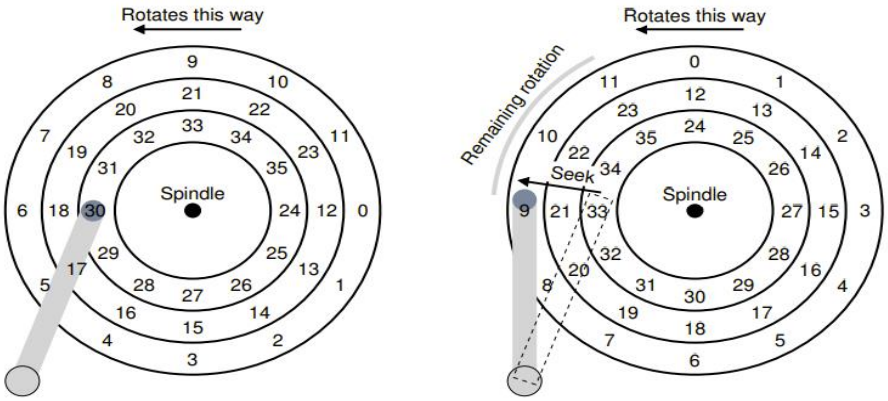
Tipik sektör boyutu, hatırlama) ve bu nedenle 0'dan 11'e kadar rakamlarla adreslenir. Burada sahip olduğumuz tek tabla, bir motorun bağlı olduğu iş milinin etrafında döner. Elbette parkur tek başına çok ilginç değil; bu sektörleri okuyabilmek veya yazabilmek istiyoruz ve bu nedenle şimdi gördüğümüz gibi bir disk koluna bağlı bir disk kafasına ihtiyacımız var (Şekil 37.2). Şekilde, kolun ucuna takılan disk kafası sektör 6'nın üzerinde konumlandırılmış ve yüzey saat yönünün tersine dönmektedir.

Tek İz Gecikmesi: Dönme Gecikmesi

Basit, tek izli diskimizde bir talebin nasıl işleneceğini anlamak için, şimdi blok 0'ı okuma talebi aldığımızı hayal edin. Disk bu talebe nasıl hizmet etmelidir? Basit diskimizde, diskin fazla bir şey yapması gerekmiyor. Özellikle, istenen sektörün disk kafasının altında dönmesini beklemesi gerekir. Bu bekleme, modern sürücülerde yeterince sık gerçekleşir ve G/Ç hizmet süresinin yeterince önemli bir bileşenidir ve özel bir adı vardır: dönüş gecikmesi (bazen dönüş gecikmesi (rotational delay), kulağa garip gelse de). Örnekte, tam dönme gecikmesi (rotational delay) R ise, diskin okuma/yazma başlığının altına 0 gelmesini beklemek için (6'dan başlarsak) yaklaşık R2'lik bir dönme gecikmesine maruz kalması gerekir. Bu tek yoldaki en kötü durum talebi, sektör 5'e yapılacak ve bu tür bir talebe hizmet vermek için neredeyse tam bir dönüş gecikmesine neden olacaktır.

Birden Fazla Parça: Zaman Ara

Şimdiye kadar diskimizde çok gerçekçi olmayan tek bir iz var; modern disklerde elbette milyonlarca var. Şimdi her zamankinden biraz daha gerçekçi disk yüzeyine bakalım, bu üç izli (Şekil 37.3, sol). Şekilde, kafa şu anda en içteki yolun (24 ila 35 arasındaki sektörleri içeren) üzerinde konumlanmıştır; sonraki iz, bir sonraki sektör grubunu (12'den 23'e) içerir ve en dıştaki iz, ilk sektörleri (0'dan 11'e) içerir.



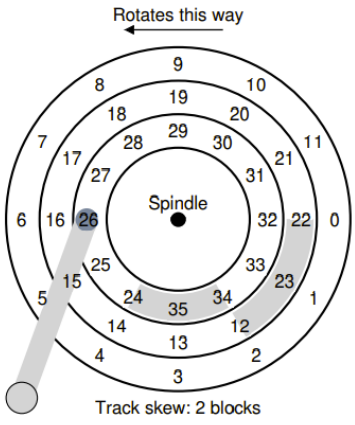
Şekil 37.3: Üç İz Artı Bir Baş (Sağ: Aramalı)

Sürücünün belirli bir sektöre nasıl erişebileceğini anlamak için, artık uzak bir sektöre yönelik bir istekte, örneğin sektör 11'e yapılan bir okumada ne olacağını izliyoruz. Bu okumaya hizmet vermek için, sürücünün önce disk kolunu doğru konuma hareket ettirmesi gerekir. Arama olarak bilinen bir süreçte (bu durumda en dıştaki) izleyin. Döndürmelerle birlikte aramalar, en maliyetli disk işlemlerinden biridir. Aramanın birçok aşaması olduğu

unutulmamalıdır: ilk olarak disk kolu hareket ederken bir hızlanma aşaması; sonra kol tam hızda hareket ederken yavaşlama, ardından kol yavaşlarken yavaşlama; kafa dikkatli bir şekilde doğru ray üzerine yerleştirildiğinden nihayet yerleşme. Yerleşme süresi (settling time) genellikle oldukça önemlidir, örneğin 0,5 ila 2 ms, çünkü sürücünün doğru yolu bulacağından emin olması gerekir (bunun yerine yakınlaştığını hayal edin!). Aramanın ardından, disk kolu, kafayı sağ ray üzerinde konumlandırmıştır. Aramanın bir tasviri Şekil 37.3'te (sağda) bulunur. Gördüğümüz gibi, arama sırasında kol istenen yola hareket ettirildi ve tabla elbette bu durumda yaklaşık 3 sektör döndürüldü. Böylece, sektör 9 disk kafasının altından geçmek üzere ve aktarımı tamamlamak için sadece kısa bir dönme gecikmesine katlanmamız gerekiyor. Sektör 11 disk başlığının altından geçtiğinde, G/Ç'nin aktarım olarak bilinen ve verilerin yüzeyden okunduğu veya yüzeye yazıldığı son aşaması gerçekleşecektir. Ve böylece, G/Ç zamanının tam bir resmine sahibiz: önce bir arama, ardından dönüş gecikmesini beklemek ve son olarak aktarım ve tabla elbette bu durumda yaklaşık 3 sektör dönmüştür. Böylece, sektör 9 disk kafasının altından geçmek üzere ve aktarımı tamamlamak için sadece kısa bir dönme gecikmesine katlanmamız gerekiyor. Sektör 11 disk başlığının altından geçtiğinde, G/Ç'nin aktarım(transfer) olarak bilinen ve verilerin yüzeyden okunduğu veya yüzeye yazıldığı son aşaması gerçekleşecektir. Ve böylece, G/Ç zamanının tam bir resmine sahibiz: önce bir arama, ardından dönüş gecikmesini beklemek ve son olarak aktarım. ve tabla elbette bu durumda yaklaşık 3 sektör dönmüştür. Böylece, sektör 9 disk kafasının altından geçmek üzere ve aktarımı tamamlamak için sadece kısa bir dönme gecikmesine katlanmamız gerekiyor. Sektör 11 disk başlığının altından geçtiğinde, G/Ç'nin aktarım olarak bilinen ve verilerin yüzeyden okunduğu veya yüzeye yazıldığı son aşaması gerçekleşecektir. Ve böylece, G/Ç zamanının tam bir resmine sahibiz: önce bir arama, ardından dönüş gecikmesini beklemek ve son olarak aktarım.

Diğer Bazı Detaylar

Üzerinde çok fazla zaman harcamayacak olsak da, sabit disklerin nasıl çalıştığına dair başka ilginç ayrıntılar da var. Pek çok sürücü, sıralı okumaların yol sınırlarını geçerken bile düzgün bir şekilde hizmet verilebilmesini sağlamak için bir tür iz eğimi kullanır. Basit örnek diskimizde bu, Şekil 37.4'te görüldüğü gibi görünebilir.



Şekil 37.4: Üç İz: 2'lik İz Eğimi

Sektörler genellikle bu şekilde çarpıktır çünkü bir izden diğerine geçerken diskin kafayı yeniden konumlandırması (hatta komşu izlere) için zamana ihtiyacı vardır. Böyle bir eğrilik olmadan, başlık bir sonraki ize hareket ettirilir, ancak istenen bir sonraki blok zaten başlığın altında dönmüş olur ve bu nedenle sürücünün bir sonraki bloğa erişmek için neredeyse tüm dönme gecikmesini beklemesi gerekir. Başka bir gerçeklik de geometrinin bir sonucu olarak, dış izlerin iç izlerden daha fazla sektöre sahip olma eğiliminde olmasıdır; orada daha fazla yer var. Bu izlere genellikle çok bölgeli(multi-zoned) disk sürücüleri denir, burada disk birden çok bölgede düzenlenir ve burada bir bölge bir yüzeydeki ardışık izler kümesidir. Her bölge, iz başına aynı sayıda sektöre sahiptir ve dış bölgeler, iç bölgelere göre daha fazla sektöre sahiptir. Nihayet, Herhangi bir modern disk sürücüsünün önemli bir parçası, bazen iz arabelleği olarak adlandırılan tarihsel nedenlerle önbelleğidir. Bu önbellek(cache), sürücünün diskten okunan veya diske yazılan verileri tutmak için kullanabileceği küçük bir bellek (track buffer) miktarıdır (genellikle yaklaşık 8 veya 16 MB). Örneğin, diskten bir sektör okurken, sürücü o yoldaki tüm sektörleri okumaya karar verebilir ve bunları kendi belleğinde önbelleğe alabilir; bunu yapmak, sürücünün aynı iz üzerindeki sonraki isteklere hızlı bir şekilde yanıt vermesini (immediate reporting) sağlar. Yazmalarda sürücünün bir seçeneği vardır: Verileri belleğine koyduğunda yazmanın tamamlandığını mı yoksa yazma gerçekten diske yazıldıktan sonra mı kabul etmelidir? İlki, geri yazma önbelleği (veya bazen anında raporlama) olarak adlandırılır ve ikincisi, baştan sona yazma (write through) olarak adlandırılır. Geri yazma önbelleği bazen sürücünün "daha hızlı" görünmesini sağlar, ancak tehlikeli olabilir; dosya sistemi veya uygulamalar, verilerin doğru olması için diske belirli bir sırayla yazılmasını gerektiriyorsa, geri yazma önbelleği sorunlara yol açabilir (ayrıntılar için dosya sistemi günlük kaydı hakkındaki bölümü okuyun).

YANDA: BOYUT ANALİZİ (DIMENSIONAL ANALYSIS)

Kimya dersinde, hemen hemen her problemi, birimleri birbirini götürecek şekilde ayarlayarak nasıl çözdüğünüzü ve bunun sonucunda bir şekilde yanıtların ortaya çıktığını hatırlıyor musunuz? Bu kimyasal sihir, boyutsal analizin abartılı adıyla biliniyor ve bilgisayar sistemleri analizinde de yararlı olduğu ortaya çıktı. Boyut analizinin(dimensional analysis) nasıl çalıştığını ve neden yararlı olduğunu görmek için bir örnek yapalım. Bu durumda, bir diskin tek bir dönüşünün ne kadar sürdüğünü milisaniye cinsinden bulmanız gerektiğini varsayalım. Ne yazık ki, size yalnızca diskin RPM'si veya dakikadaki dönüş (rotations per minute) sayısı verilir. Diyelim ki 10K RPM'lik bir diskten bahsediyoruz (yani dakikada 10.000 kez dönüyor). Dönüş başına süreyi milisaniye cinsinden elde edecek şekilde boyutsal analizi nasıl kurarız? Bunun için sol tarafa istenen birimleri koyarak başlıyoruz: Zaman (ms)/ 1 Döndürme.

Daha sonra, mümkün olduğunda birimleri iptal ettiğimizden emin olarak bildiğimiz her şeyi yazarız. İlk önce 1 dakika/ 10.000 Döndürme elde ederiz. (solda olduğu için dönüşü altta tutun), ardından dakikayı saniyeye dönüştürün 60 saniye/1 dakika ve son olarak saniyeleri milisaniye cinsinden dönüştürün 1000 ms/1 saniye. Nihai sonuç, birimleri güzel bir şekilde iptal edilen bu denklemdir:

Zaman (ms)/1 ROT = 6ms/rotasyon

37.4 G/Ç Zamanı: Hesabı Yapmak

Artık diskin soyut bir modeline sahip olduğumuza göre, disk performansını daha iyi anlamak için küçük bir analiz kullanabiliriz. Özellikle, artık G/Ç süresini üç ana bileşenin toplamı olarak gösterebiliriz:

TI/O = Taramak + Trotasyon + Ttransfer

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Tablo 37.1: Disk Sürücüsü Özellikleri: SCSI ve SATA

Genellikle sürücüler arasında karşılaştırma yapmak için daha kolay kullanılan (aşağıda yapacağımız gibi) G/Ç oranının (RI/O) zamandan itibaren kolayca hesaplanabileceğini unutmayın. Aktarımın boyutunu geçen süreye bölmeniz yeterlidir:

RI/O= Boyut Aktarımı/ TI/O

G/Ç süresi için daha iyi bir fikir edinmek için aşağıdaki hesaplamayı yapalım. İlgilendiğimiz iki iş yükü olduğunu varsayın. Rastgele(random) iş yükü olarak bilinen ilki, diskteki rastgele konumlara küçük (örneğin, 4 KB) okumalar gönderir. Rastgele iş yükleri, veritabanı yönetim sistemleri de dahil olmak üzere birçok önemli uygulamada yaygındır. Sıralı iş yükü olarak bilinen ikincisi, atlamadan çok sayıda sektörü diskten arka arkaya okur. Sıralı erişim kalıpları oldukça yaygındır ve bu nedenle de önemlidir. Rastgele ve sıralı(sequential) iş yükleri arasındaki performans farkını anlamak için önce disk sürücüsü hakkında birkaç varsayımda bulunmamız gerekir. Seagate'in birkaç modern diskine bakalım. Cheetah 15K.5 [S09b] olarak bilinen ilki, yüksek performanslı bir SCSI sürücüsüdür. İkincisi, Barracuda [S09a], kapasite için üretilmiş bir sürücüdür. Her ikisine ilişkin ayrıntılar Tablo 37.1'de bulunmaktadır. Gördüğünüz gibi, sürücüler oldukça farklı özelliklere sahiptir ve birçok yönden disk sürücüsü pazarının iki önemli bileşenini güzel bir şekilde özetlemektedir. Birincisi, sürücülerin olabildiğince hızlı dönecek, düşük arama süreleri sağlayacak ve verileri hızlı bir şekilde aktaracak şekilde tasarlandığı "yüksek performanslı" sürücü pazarıdır. İkincisi, bayt başına maliyetin en önemli unsur olduğu “kapasite” pazarıdır; bu nedenle, sürücüler daha yavaştır ancak mevcut alana mümkün olduğu kadar çok bit yerleştirir. Bu rakamlardan, sürücülerin yukarıda belirtilen iki iş yükü altında ne kadar iyi çalışacağını hesaplamaya başlayabiliriz. Rastgele iş yüküne bakarak

başlayalım. Her 4 KB okumanın diskte rastgele bir yerde gerçekleştiğini varsayarsak, bu tür bir okumanın ne kadar süreceğini hesaplayabiliriz. Çita üzerinde:

Taramak = 4 ms, Dönme = 2 ms, Ttransfer = 30 mikro saniye

İPUCU: DİSKLERİ SIRALI OLARAK KULLANIN Mümkün olduğunda, verileri disklere ve disklerden sıralı bir şekilde aktarın. Sıralı mümkün değilse, en azından verileri büyük parçalar halinde aktarmayı düşünün: ne kadar büyükse o kadar iyi. G/Ç küçük rastgele parçalar halinde yapılırsa, G/Ç performansı önemli ölçüde düşer. Ayrıca, kullanıcılar zarar görecektir. Ayrıca, dikkatsiz rastgele G/Ç'lerinizle ne kadar acı çektiğinizi bilerek acı çekeceksiniz.

Ortalama arama süresi (4 milisaniye), üretici tarafından bildirilen ortalama süre olarak alınır; tam bir aramanın (yüzeyin bir ucundan diğer ucuna) muhtemelen iki veya üç kat daha uzun süreceğini unutmayın. Ortalama dönme gecikmesi doğrudan RPM'den hesaplanır. 15000 RPM, 250 RPS'ye (saniyede dönüş) eşittir; bu nedenle, her dönüş 4 ms sürer. Ortalama olarak, disk yarım dönüşle karşılaşacaktır ve bu nedenle ortalama süre 2 ms'dir. Son olarak, aktarım süresi, en yüksek aktarım hızı üzerinden yalnızca aktarım boyutudur; burada yok denecek kadar küçüktür (30 mikro saniye; sadece 1 milisaniye elde etmek için 1000 mikro saniyeye ihtiyacımız olduğunu unutmayın!). Böylece, yukarıdaki denklemimize göre Cheetah için TI/O kabaca 6 ms'ye eşittir. G/Ç oranını hesaplamak için transfer boyutunu ortalama süreye böleriz ve böylece yaklaşık 0,66 MB/sn'lik rastgele iş yükü altında Cheetah için RI/O'ya ulaşır. Barracuda için aynı hesaplama, yaklaşık 13,2 ms'lik bir TI/O, iki kattan fazla daha yavaş ve dolayısıyla yaklaşık 0,31 MB/s'lik bir oran verir. Şimdi sıralı iş yüküne bakalım. Burada çok uzun bir transferden önce tek bir arama ve rotasyon olduğunu varsayabiliriz. Basit olması için aktarım boyutunun 100 MB olduğunu varsayalım. Böylece Barracuda ve Cheetah için TI/O sırasıyla yaklaşık 800 ms ve 950 ms'dir. Dolayısıyla G/Ç oranları, sırasıyla 125 MB/sn ve 105 MB/sn'lik en yüksek aktarım hızlarına çok yakındır. Tablo 37.2 bu sayıları özetlemektedir. Tablo bize bir dizi önemli şeyi gösteriyor. Birincisi ve en önemlisi, rastgele ve sıralı iş yükleri arasında sürücü performansında büyük bir boşluk vardır. Çita için yaklaşık 200 kat ve Barracuda için 300 kat fazla fark. Ve böylece bilgi işlem tarihindeki en bariz tasarım ipucuna ulaşıyoruz. Daha incelikli ikinci bir nokta: Üst düzey "performans" sürücüler ile düşük uç "kapasite" sürücüler arasında büyük bir performans farkı vardır. Bu nedenle (ve diğerleri), insanlar genellikle ikincisini olabildiğince ucuza almaya çalışırken birincisi için en yüksek doları ödemeye isteklidir.

	Cheetah	Barracuda
$R_{I/O}$ Random	0.66 MB/s	0.31 MB/s
$R_{I/O}$ Sequential	125 MB/s	105 MB/s

Tablo 37.2: Disk Sürücüsü Performansı: SCSI ve SATA

BİR KENARA: “ORTALAMA” ARAMA HESAPLAMASI Birçok kitap ve makalede, ortalama disk arama süresinin kabaca tam arama süresinin üçte biri olduğunu göreceksiniz. Bu nereden

geliyor? Zamana değil, ortalama arama mesafesine dayalı basit bir hesaplamadan kaynaklandığı ortaya çıktı. Diski 0'dan N'ye kadar bir dizi iz olarak hayal edin. Herhangi iki x ve y izi arasındaki arama mesafesi böylece aralarındaki farkın mutlak değeri olarak hesaplanır: |x − y|. Ortalama arama mesafesini hesaplamak için yapmanız gereken tek şey, önce tüm olası arama mesafelerini toplamaktır:

37.5 Disk Zamanlama

G/Ç'nin yüksek maliyeti nedeniyle, işletim sistemi tarihsel olarak diske verilen G/Ç'lerin sırasına karar vermede rol oynamıştır. Daha spesifik olarak, bir dizi G/Ç isteği verildiğinde, disk planlayıcı (disk scheduler)istekleri inceler ve hangisinin sıralanacağına karar verir [SCO90, JW91]. Her işin uzunluğunun genellikle bilinmediği iş programlamanın aksine, disk programlama ile bir "iş"in (yani, disk talebinin) ne kadar süreceği konusunda iyi bir tahminde bulunabiliriz. Bir talebin aranmasını ve olası dönüş gecikmesini tahmin ederek, disk zamanlayıcı her bir talebin ne kadar süreceğini bilir ve böylece (açgözlülükle) önce hizmete girmesi en az zaman alacak olanı seçer. Böylece, disk zamanlayıcı, çalışmasında SJF (önce en kısa iş) ilkesini izlemeye çalışacaktır. ssf: Önce En Kısa Arama Süresi İlk disk planlama yaklaşımı, en kısa arama zamanı ilk (SSTF) olarak bilinir (aynı zamanda en kısa önce arama veya SSF olarak da adlandırılır). SSTF, G/Ç isteklerinin sırasını yola göre sıralar ve istekleri önce tamamlamak için en yakın yoldaki istekleri seçer. Örneğin, başın mevcut pozisyonunun iç yolun üzerinde olduğunu varsayarsak ve sektör 21 (orta yol) ve 2 (dış yol) için taleplerimiz varsa, talebi önce 21'e gönderir, tamamlanmasını bekleriz, ve ardından talebi 2'ye gönderin (Şekil 37.5). SSTF bu örnekte iyi çalışıyor, önce orta yolu, sonra dış yolu arıyor. Bununla birlikte, aşağıdaki nedenlerden dolayı SSTF her derde deva değildir. İlk olarak, sürücü geometrisi ana bilgisayar işletim sisteminde mevcut değildir; bunun yerine, bir dizi blok görür. Neyse ki, bu sorun oldukça kolay bir şekilde düzeltildi. SSTF yerine.

İkinci sorun daha temel: açlık. Yukarıdaki örneğimizde, başın şu anda konumlandığı iç yola sürekli bir istek akışı olup olmadığını hayal edin. Diğer parçalara yapılan istekler, saf bir SSTF yaklaşımı tarafından tamamen göz ardı edilir (principle of SJF (shortest job first)). Ve böylece sorunun özü:

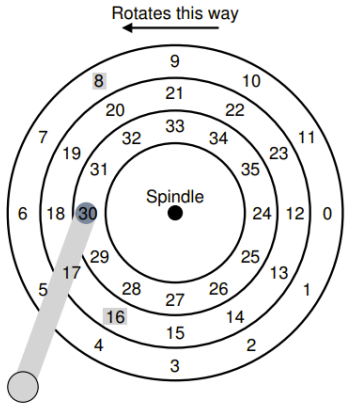
CRUX: DİSK AÇILMASI NASIL BAŞA ÇEKİLİR SSTF benzeri zamanlamayı nasıl uygulayabilir, ancak aç kalmayı nasıl önleyebiliriz?

Asansör (a.k.a. SCAN veya C-SCAN)

Bu sorunun cevabı bir süre önce geliştirildi (örneğin [CKR72]'ye bakın) ve nispeten basittir. Başlangıçta SCAN olarak adlandırılan algoritma, izler boyunca sırayla disk hizmet istekleri arasında hareket eder. Disk boyunca tek bir geçişe tarama diyelim. Bu nedenle, diskin bu taramasında zaten hizmet verilmiş bir yolda bir blok için bir istek gelirse, hemen işleme alınmaz, bunun yerine bir sonraki taramaya kadar kuyruğa alınır. SCAN'ın hepsi aynı şeyi yapan bir dizi çeşidi vardır. Örneğin, Coffman ve ark. tarama yaparken hizmet verilecek kuyruğu donduran F-SCAN'ı tanıttı [CKR72]; bu eylem, tarama sırasında gelen istekleri daha sonra hizmet verilmek üzere bir kuyruğa yerleştirir. Bunu yapmak, geç gelen (ancak daha yakın) isteklerin hizmetini geciktirerek uzaktaki isteklerin aç kalmasını önler. C-SCAN, Dairesel SCAN'ın kısaltması olan başka bir yaygın varyanttır. Algoritma, disk boyunca tek bir yönde tarama yapmak yerine, dıştan içe ve sonra içten dışa vb. asansör algoritması, çünkü yukarı veya aşağı giden bir asansör gibi davranır ve yalnızca hangi katın daha yakın olduğuna bağlı

olarak katlara hizmet vermez. 10. kattan 1. kata inerken birisi 3'te binip 4'e bassa ve asansör 1'den "yakın" olduğu için 4'e çıksa ne kadar can sıkıcı olurdu bir düşünün! Gördüğünüz gibi asansör algoritması gerçek hayatta kullanıldığında asansörlerde kavga çıkmasını engelliyor. Disklerde, sadece aç kalmayı önler. Ne yazık ki, SCAN ve kuzenleri en iyi zamanlama teknolojisini temsil etmez. Özellikle, SCAN (veya hatta SSTF), SJF ilkesine olabildiğince sıkı sıkıya bağlı değildir. Özellikle, dönüşü göz ardı ederler. Ve böylece, başka bir dönüm noktası:

CRUX: DİSK DÖNDÜRME MALİYETLERİ NASIL HESAPLANIR? Hem aramayı hem de döndürmeyi hesaba katarak SJF'ye daha yakın olan bir algoritmayı nasıl uygulayabiliriz?



Şekil 37.6: SSTF: Bazen Yeterince İyi Değil

SPTF: Önce En Kısa Konumlandırma Süresi

Sorunumuzun çözümü olan en kısa konumlandırma süresi veya SPTF zamanlaması (bazen en kısa erişim süresi veya SATF olarak da adlandırılır) konusunu tartışmadan önce, sorunu daha ayrıntılı olarak anladığımızdan emin olalım. Şekil 37.6 bir örnek sunar. Örnekte, kafa şu anda iç rayda sektör 30'un üzerinde konumlanmıştır. Bu nedenle programlayıcının bir sonraki talebi için sektör 16'yı (orta yol üzerinde) mi yoksa sektör 8'i (dış yol üzerinde) mi programlaması gerektiğine karar vermesi gerekir. Peki bundan sonra hangisine hizmet etmeli? Cevap, elbette, “duruma göre değişir”. Mühendislikte, değiş tokuşların mühendisin yaşamının bir parçası olduğunu yansıtan "duruma bağlıdır" ifadesinin neredeyse her zaman yanıt olduğu ortaya çıktı; bu tür özdeyişler de bir tutamda iyidir, örneğin, patronunuzun sorusuna bir yanıt bilmiyorsanız, bu taşı denemek isteyebilirsiniz. Yine de, neden bağlı olduğunu bilmek neredeyse her zaman daha iyidir, burada tartıştığımız şey de budur. Burada bağlı olduğu şey, dönmeye kıyasla aramanın göreli süresidir. Örneğimizde, arama süresi dönme gecikmesinden çok daha yüksekse, SSTF (ve varyantları) gayet iyi. Ancak, aramanın döndürmeden biraz daha hızlı olduğunu hayal edin. O halde, bizim örneğimizde, servis talebi 8'in altından geçmeden önce tüm yol boyunca dönmesi gereken servis 16'ya giden orta hatta daha kısa arama yapmaktansa, dış hat üzerinde servis talebi 8'e daha uzağa gitmek daha mantıklı olacaktır. disk kafası. Modern sürücülerde, yukarıda gördüğümüz gibi, hem arama hem de döndürme kabaca eşdeğerdir (tabii ki tam isteklere bağlı olarak) ve bu nedenle SPTF kullanışlıdır ve performansı artırır. Ancak, bir işletim sisteminde uygulamak daha da zordur, iz sınırlarının nerede olduğu veya disk kafasının şu anda nerede olduğu (dönme anlamında) genellikle iyi bir fikre sahip değildir. Bu nedenle, SPTF genellikle aşağıda açıklanan bir sürücü içinde gerçekleştirilir.

İPUCU: HER ZAMAN BAĞLIDIR (LIVNY YASASI) Meslektaşımız Miron Livny'nin her zaman söylediği gibi, hemen hemen her soru “duruma göre değişir” şeklinde yanıtlanabilir. Ancak, bu şekilde çok fazla soru yanıtlarsanız, insanlar size soru sormaktan tamamen vazgeçecekleri için dikkatli kullanın. Örneğin, biri "öğle yemeğine gitmek ister misin?" diye sorar. Cevap veriyorsunuz: “duruma göre değişir, geliyor musunuz?”

Diğer Planlama Sorunları

Temel disk işlemleri, zamanlama ve ilgili konuların bu kısa açıklamasında ele almadığımız birçok başka konu var. Böyle bir sorun şudur: Modern sistemlerde disk zamanlaması nerede yapılır? Daha eski sistemlerde, tüm zamanlamayı işletim sistemi yapıyordu; Bekleyen istekleri inceledikten sonra, işletim sistemi en iyisini seçer ve diske verir. Bu istek tamamlandığında, bir sonraki seçilir ve bu böyle devam ederdi. O zamanlar diskler daha basitti, hayat da öyle. Modern sistemlerde, diskler çok sayıda bekleyen isteği karşılayabilir ve karmaşık dahili programlayıcılara sahiptir (bunlar SPTF'yi doğru bir şekilde uygulayabilir; disk denetleyicinin içinde, tam kafa konumu dahil ilgili tüm ayrıntılar mevcuttur). Böylece, işletim sistemi planlayıcısı genellikle en iyi birkaç isteğin ne olduğunu düşünürse onu seçer (örneğin 16) ve hepsini diske gönderir; disk daha sonra söz konusu isteklere mümkün olan en iyi (SPTF) sırada hizmet vermek için baş pozisyonuna ilişkin dahili bilgisini ve ayrıntılı yol düzeni bilgisini kullanır. Disk zamanlayıcılar tarafından gerçekleştirilen bir diğer önemli ilgili görev, G/Ç birleştirmedir(I/O merging.) Örneğin, Şekil 37.6'daki gibi blok 33'ü, ardından 8'i ve ardından 34'ü okumak için bir dizi talep hayal edin. Bu durumda, programlayıcı blok 33 ve 34'e yönelik talepleri tek bir iki bloklu talepte birleştirmelidir; zamanlayıcının yaptığı herhangi bir yeniden sıralama, birleştirilmiş istekler üzerine gerçekleştirilir. Birleştirme, diske gönderilen isteklerin sayısını azalttığı ve böylece genel giderleri azalttığı için işletim sistemi düzeyinde özellikle önemlidir. Modern zamanlayıcıların ele aldığı son bir sorun şudur: sistem diske bir G/Ç vermeden önce ne kadar beklemelidir? Biri safça, diskin tek bir G/Ç'ye sahip olduğunda bile hemen sürücüye istek göndermesi gerektiğini düşünebilir; bu yaklaşıma işi koruma denir, çünkü hizmet verme istekleri varsa disk asla boşta kalmaz. Bununla birlikte, ileriye dönük disk zamanlaması üzerine yapılan araştırmalar, işi korumayan bir yaklaşım olarak adlandırılan bir yaklaşımda bazen biraz [ID01] beklemenin daha iyi olduğunu göstermiştir. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz). Tek bir G/Ç'ye bile sahip olduğunda, isteği hemen sürücüye göndermelidir; bu yaklaşıma işi koruma denir, çünkü hizmet verme istekleri varsa disk asla boşta kalmaz. Bununla birlikte, ileriye dönük disk zamanlaması (anticipatory disk scheduling) üzerine yapılan araştırmalar, işi korumayan bir yaklaşım olarak adlandırılan bir yaklaşımda bazen biraz [ID01] beklemenin daha iyi olduğunu göstermiştir. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz). Tek bir G/Ç'ye bile sahip olduğunda, isteği hemen sürücüye göndermelidir; bu yaklaşıma işi koruma denir, çünkü hizmet verme istekleri varsa disk asla boşta kalmaz. Bununla birlikte, ileriye dönük disk zamanlaması üzerine yapılan araştırmalar, işi

korumayan bir yaklaşım olarak adlandırılan bir yaklaşımda bazen biraz [ID01] beklemenin daha iyi olduğunu göstermiştir. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz). Hizmet verme istekleri varsa disk asla boşta kalmayacağından. Bununla birlikte, ileriye dönük disk zamanlaması üzerine yapılan araştırmalar, işi korumayan bir yaklaşım olarak adlandırılan bir yaklaşımda bazen biraz [ID01] beklemenin daha iyi olduğunu göstermiştir. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz). Hizmet verme istekleri varsa disk asla boşta kalmayacağından. Bununla birlikte, ileriye dönük disk zamanlaması (anticipatory disk scheduling) üzerine yapılan araştırmalar, işi korumayan bir yaklaşım olarak adlandırılan bir yaklaşımda bazen biraz [ID01] beklemenin daha iyi olduğunu göstermiştir. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz). Ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz). Ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz).

37.6 Özet

Disklerin nasıl çalıştığının bir özetini sunduk. Özet, aslında ayrıntılı bir işlevsel modeldir; gerçek sürücü tasarımına giren inanılmaz fizik, elektronik ve malzeme bilimini açıklamaz. Bu türden daha fazla ayrıntıyla ilgilenenler için farklı bir ana dal (veya belki de yan dal) öneriyoruz; bu modelden memnun olanlar için, aferin! Artık bu inanılmaz cihazların üzerine daha ilginç sistemler inşa etmek için modeli kullanmaya devam edebiliriz.

References

[ADR03] “More Than an Interface: SCSI vs. ATA” Dave Anderson, Jim Dykes, Erik Riedel FAST ’03, 2003 One of the best recent-ish references on how modern disk drives really work; a must read for anyone interested in knowing more.

[CKR72] “Analysis of Scanning Policies for Reducing Disk Seek Times” E.G. Coffman, L.A. Klimko, B. Ryan SIAM Journal of Computing, September 1972, Vol 1. No 3. Some of the early work in the field of disk scheduling.

[ID01] “Anticipatory Scheduling: A Disk-scheduling Framework To Overcome Deceptive Idleness In Synchronous I/O” Sitaram Iyer, Peter Druschel SOSP ’01, October 2001 A cool paper showing how waiting can improve disk scheduling: better requests may be on their way!

[JW91] “Disk Scheduling Algorithms Based On Rotational Position” D. Jacobson, J. Wilkes Technical Report HPL-CSP-91-7rev1, Hewlett-Packard (February 1991) A more modern take on disk scheduling. It remains a technical report (and not a published paper) because the authors were scooped by Seltzer et al. [SCO90].

[RW92] “An Introduction to Disk Drive Modeling” C. Ruemmler, J. Wilkes IEEE Computer, 27:3, pp. 17-28, March 1994 A terrific introduction to the basics of disk operation. Some pieces are out of date, but most of the basics remain.

[SCO90] “Disk Scheduling Revisited” Margo Seltzer, Peter Chen, John Ousterhout USENIX 1990 A paper that talks about how rotation matters too in the world of disk scheduling.

[SG04] “MEMS-based storage devices and standard disk interfaces: A square peg in a round hole?”

Steven W. Schlosser, Gregory R. Ganger FAST ’04, pp. 87-100, 2004 While the MEMS aspect of this paper hasn’t yet made an impact, the discussion of the contract between file systems and disks is wonderful and a lasting contribution.

[S09a] “Barracuda ES.2 data sheet” http://www.seagate.com/docs/pdf/datasheet/disc/ds cheetah 15k 5.pdf A data sheet; read at your own risk. Risk of what? Boredom.

[S09b] “Cheetah 15K.5” http://www.seagate.com/docs/pdf/datasheet/disc/ds barracuda es.pdf See above commentary on data sheets.

SORULAR:

Bu ev ödevi, modern bir sabit diskin nasıl çalıştığını size alıştırmak için disk.py'yi kullanır. Pek çok farklı seçeneğe sahiptir ve diğer simülasyonların çoğundan farklı olarak, disk hareket halindeyken tam olarak ne olduğunu size gösteren bir grafik animatöre sahiptir. Ayrıntılar için BENİOKU'ya bakın.

1. Aşağıdakiler için arama, döndürme ve aktarma sürelerini hesaplayın istek kümeleri: -a 0, -a 6, -a 30, -a 7,30,8 ve son olarak -a 10,11,12,13.

Bu tür istek kümelerini hesaplamak için;

- -a 0: Bu istek kümesinde yer alan kontrolü, uçuş süresi veya uçuş süresi için yeterli değildir. Bu nedenle, arama, döndürme ve aktarma süresi hesaplanamaz.
- -a 6:

  Bu Süre (dakika) = (6)
- -30

  Süre (dakika)
- -a 7,30,8: Bu istek kümesinde yer alan yönü, uçuş süresi veya uçuş süresi için yeterli değildir. Bu nedenle, arama, döndürme ve aktarma süresi hesaplanamaz.
- -a

2. Yukarıdaki isteklerin aynısını yapın, ancak arama oranını farklı değerlerle değiştirin: -S 2, -S 4, -S 8, -S 10, -S 40, -S 0.1. Zaman nasıl değişir?

Arama oranını farklı değerlere değiştirirsek, ses dosyası içindeki belirli konumları arama süreleri de buna göre değişir. Arama hızı, ses dosyasının oynatılma hızını belirler; daha yüksek arama hızı daha hızlı oynatmayı ve daha düşük arama hızı daha yavaş oynatmayı sağlar. Örneğin, -S 2 seçeneğini kullanırsak, ses dosyası normal hızın iki katı hızda çalınır ve dosya içinde belirli konumların aranması, varsayılan arama hızının kullanılmasına kıyasla daha hızlı olur. Öte yandan, -S 0.1 seçeneğini kullanırsak, ses dosyası daha yavaş çalınır ve dosya içinde belirli konumların aranması, varsayılan arama hızının kullanılmasına kıyasla daha yavaş olur. Sağlanan farklı arama oranı değerleri için arama sürelerinin nasıl değişeceğinin bir özeti aşağıdaki gibidir:
-S 2: Arama, varsayılan arama oranına göre daha hızlı olacaktır.
-S 4: Arama, varsayılan arama hızına kıyasla daha da hızlı olacaktır.
-S 8: Arama, varsayılan arama oranına göre çok daha hızlı olacaktır.
-S 10: Arama, varsayılan arama hızına kıyasla daha da hızlı olacaktır.
-S 40: Arama, varsayılan arama oranına göre çok hızlı olacaktır.
-S 0.1: Arama, varsayılan arama hızına göre daha yavaş olacaktır.
Arama sürelerindeki gerçek değişikliklerin, belirli ses dosyasına ve kullanılan oynatıcı yazılımının veya donanımının özelliklerine bağlıdırlar.

3. Yukarıdaki isteklerin aynısını yapın, ancak dönüş hızını değiştirin: -R 0.1, -R 0.5, -R 0.01. Zaman nasıl değişir?

Dönme hızı -0,1, -0,5 veya -0,01 olarak değiştirilirse, Dünya'nın bir dönüşü tamamlaması için geçen süre değişir.

-0,1'lik bir dönüş hızı için, Dünya'nın bir dönüşü veya 36.000 saniyeyi tamamlaması 10 kat daha uzun sürecektir.

-0,5'lik bir dönüş hızı için, Dünya'nın bir dönüşü tamamlaması 2 kat daha uzun, yani 7.200 saniye sürer.

-0,01'lik bir dönüş hızı için, Dünya'nın bir dönüşü tamamlaması 1.000 kat daha uzun, yani 3.600.000 saniye sürer.

Dünyanın gerçek dönüş hızının saniyede yaklaşık 0,004 derece veya saniyede yaklaşık 0,000011 radyan olduğunu dikkate almalıyız. Yukarıda verilen değerleri yalnızca açıklama amaçlıdır kullanabiliriz ve bu bilgiler Dünya'nın gerçek dönüş hızını yansıtmaz.

4. Bazı istek akışlarının FIFO'dan daha iyi bir politika ile daha iyi sunulacağını fark etmiş olabilirsiniz. Örneğin, istek akışı -a 7,30,8 ile istekler hangi sırayla işlenmelidir? Şimdi aynı iş yükünde en kısa arama süresi ilk (SSTF) planlayıcısını (-p SSTF) çalıştırın; Her talebin yerine getirilmesi ne kadar sürmelidir (arama, döndürme, aktarma)?

En kısa arama süresi ilk (SSTF) zamanlayıcısı, bir dizi disk isteği için toplam arama süresini en aza indirmeyi amaçlayan bir disk zamanlama algoritmasıdır. Bunu, isteklere disk kafasının hareket etmesi gereken mesafeyi en aza indirecek sırayla hizmet vererek yapar.

İstek akışı -a 7,30,8 için, SSTF programlayıcı istekleri şu sırayla işler: 8, 7, 30. Bunun nedeni, 8. parça için isteğe hizmet vermenin en az miktarda arama süresi gerektirmesi ve ardından 7. parça için istek ve son olarak 30. parça için istek.

Her istek için toplam arama süresi, dönüş süresi ve aktarım süresi, kullanılan diskin dönme hızı, veri aktarım hızı ve yapılan isteklerin boyutu gibi özelliklerine bağlı olacaktır. Ayrıca, diskte istenen verileri bulmak ve bunlara erişmek için gereken süre gibi, isteklere hizmet verilmesiyle ilgili herhangi bir ek yüke de bağlı olacaktır.

Disk ve yapılan istekler hakkında daha fazla bilgi olmadan, akıştaki her istek için tam arama zamanı, dönüş süresi ve aktarım süresini belirlemek mümkün değildir.

5. Şimdi aynı şeyi yapın, ancak önce en kısa erişim zamanı (SATF) planlayıcıyı (-p SATF) kullanın. -a 7,30,8 ile belirtilen istek kümesi için herhangi bir fark yaratır mı? SATF'nin SSTF'den belirgin şekilde daha iyi performans gösterdiği bir dizi istek bulun; Gözle görülür bir fark oluşmasının şartları nelerdir?

SATF programlayıcı kullanılarak, -a 7,30,8 ile belirtilen küme için isteklerin sırası şu şekilde olur:7, 8, 30. Bu sıra ile SSTF zamanlayıcı tarafından üretilen sipariş arasındaki fark, SATF zamanlayıcının her zaman önce en kısa erişim süresine sahip istek, oysa SSTF planlayıcısı önce en kısa arama süresine sahip isteği seçer.

Genel olarak, kısa erişim sürelerine sahip çok sayıda istek ve uzun erişim sürelerine sahip az sayıda istek olduğunda, SATF planlayıcısı SSTF planlayıcısından daha iyi performans gösterecektir. Bunun nedeni, SATF planlayıcının istekleri her zaman kısa erişim süreleriyle önceliklendirmesidir, bu da daha az toplam arama ve daha düşük genel arama süresi ile sonuçlanabilir.

SATF'nin SSTF'den önemli ölçüde daha iyi performans gösterdiği bir dizi istek bulmak için, kısa erişim süreli birçok isteğin ve uzun erişim süreli birkaç isteğin olduğu bir senaryo düşünebiliriz. Örneğin, şu istek setine sahipsek: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ve disk kafasının mevcut konumu 50 ise, SSTF kullanan isteklerin sırası zamanlayıcı şöyle olur: 60, 40, 70, 30, 80, 20, 90, 10, 100. Bu durumda toplam arama süresi: 10 + 10 + 20 + 20 + 30 + 30 + 40 + 40 + 50 = 240.

Öte yandan, SATF planlayıcı kullanıldığında, isteklerin sırası şöyle olacaktır: 60, 70, 80, 90, 100, 40, 30, 20, 10. Bu durumda toplam arama süresi: 10 + 20 + 30 olacaktır. + 40 + 50 + 10 + 20 + 30 + 40 = 220. Bu örnekte, SATF planlayıcısı SSTF planlayıcısından önemli ölçüde daha iyi performans gösteriyor çünkü istekleri kısa erişim süreleriyle öncelik sırasına koyabiliyor ve bu da daha düşük bir toplam arama süresi sağlıyor.

6. -a 10,11,12,13 istek akışının disk tarafından özellikle iyi işlenmediğini fark etmiş olabilirsiniz. Nedenmiş? Bu sorunu çözmek için bir iz eğriliği (-o skew, burada skew negatif olmayan bir tam sayıdır) tanıtabilir misiniz? Varsayılan arama oranı göz önüne alındığında, bu istek kümesi için toplam süreyi en aza indirmek için sapma ne olmalıdır? Peki ya farklı arama oranları (örn; -S 2, -S 4)? Genel olarak, arama oranı ve sektör düzeni bilgileri göz önüne alındığında, çarpıklığı bulmak için bir formül yazabilir misiniz?

İstek akışının (10, 11, 12, 13) disk tarafından iyi işlenmemiş olmasının birkaç nedeni olabilir:

1. İstekler, diskin bunları verimli bir şekilde işlemesi için en uygun sırada olmayabilir. Örneğin, diskte istekler birbirinden uzaksa, disk kafasının bunlar arasında hareket etmesi daha uzun sürer ve isteklerin tamamlanması için geçen toplam süre artar.
2. Diskin dönme hızı, disk önbelleğinin boyutu veya disk denetleyicisinin iş yükü gibi diğer faktörler nedeniyle disk, isteklere yeterince hızlı hizmet veremeyebilir.

Bu sorunları çözmek için, istekleri tamamlamak için toplam süreyi en aza indirecek şekilde yeniden sıralamak için izleme çarpıklığını kullanabiliriz. İz eğriliği, isteklere daha verimli bir şekilde hizmet verilebilmesi için diskteki her bir izin başlangıç konumunu dengeleme işlemidir.

Varsayılan arama hızına sahip bir dizi istek için toplam süreyi en aza indirmek için, eğrilik, disk kafasının istekler arasında seyahat etmesi gereken mesafeyi en aza indirecek şekilde diskteki konuma ayarlanmalıdır. Bu konum, diskteki sektörlerin belirli düzenine ve isteklerin konumlarına bağlı olacaktır. Farklı arama oranları için, optimum sapma, arama oranının belirli değerlerine ve sektör düzenine bağlı olacaktır. Genel olarak, bu parametrelere göre en uygun eğimi hesaplamak için bir formül kullanabiliriz. Bu algoritma, "yükselticinin" (yani, disk

kafasının) isteklere hizmet vermek için kat etmesi gereken toplam mesafeyi en aza indirerek optimum eğimi hesaplar.

7. Çok bölgeli diskler, dış izlere daha fazla sektör yerleştirir. Bu diski bu şekilde yapılandırmak için -z bayrağı ile çalıştırın. Spesifik olarak, -z 10,20,30 ile çalıştırılan bir diske karşı bazı istekler çalıştırmayı deneyin (sayılar, iz başına bir sektör tarafından kaplanan açısal alanı belirtir; bu örnekte, dış yol her 10 derecede bir sektörle paketlenir, her 20 derecede bir orta yol ve her 30 derecede bir sektöre sahip iç yol). Bazı rasgele istekler çalıştırın (örn. -a -1 -A 5,-1,0; bu, -a -1 bayrağı aracılığıyla rasgele isteklerin kullanılması gerektiğini ve 0 ile maksimum arasında değişen beş isteğin oluşturulacağını belirtir) ve arama, döndürme ve aktarma sürelerini hesaplayıp hesaplayamayacağınıza bakın. Farklı rastgele tohumlar kullanın (-s 1, -s 2, vb.). Dış, orta ve iç hatlardaki bant genişliği (birim zaman başına sektörler olarak) nedir?

-z bayrağıyla çalışan bir disk için arama, döndürme ve aktarım sürelerini hesaplamak için aşağıdaki adımları kullanmalıyız:

1. -a -1 ve -A bayraklarını kullanarak, örneğinizde belirtildiği gibi (örneğin, **`-a -1 -A 5,-1,0`**) bir dizi rasgele istek oluşturun. Bu, diskin performansını test etmek için kullanabileceğiniz bir dizi istek üretecektir.
2. Her istek için arama süresini, dönüş süresini ve aktarım süresini hesaplayın. Arama süresi, disk kafasının istek için doğru yola gitmesi için geçen süredir. Dönme süresi, diskin başlık altında istenen sektörü döndürmesi için geçen süredir. Aktarım süresi, istenen sektörü okumak veya yazmak için geçen süredir.
3. Her iz için bant genişliğini (birim zaman başına sektörler olarak) hesaplamak için, aktarılan sektör sayısını bunları aktarmak için harcanan toplam süreye bölün. Örneğin, dış izden 10 saniyelik bir süre içinde 1000 sektör aktarırsanız, dış iz için bant genişliği saniyede 100 sektör olur.

8. Zamanlama pencereleri, bir sonraki hizmetin hangi sektöre sunulacağını belirlemek için bir diskin aynı anda kaç sektör isteğini inceleyebileceğini belirler. Çok sayıda istekten rastgele bazı iş yükleri oluşturun (örneğin, -A 1000,-1,0, belki farklı tohumlarla) ve zamanlama penceresi 1'den istek sayısına (örneğin, , -w 1'den -w 1000'e kadar ve aradaki bazı değerler). Mümkün olan en iyi performansa ulaşmak için ne kadar büyük bir zamanlama penceresi gerekiyor? Bir grafik yapın ve görün. İpucu: -c bayrağını kullanın ve bunları daha hızlı çalıştırmak için grafikleri -G ile açmayın. Zamanlama penceresi 1 olarak ayarlandığında, hangi politikayı kullandığınız fark eder mi?

Rastgele iş yükleri oluşturmak için fio (Flexible I/O Tester) gibi bir araç kullanabiliriz. Fio, istek sayısını, her isteğin boyutunu ve kullanılacak zamanlama politikasını belirtmenize olanak tanır. '**`-c`** '**`Zamanlama penceresi boyutunu' -w 'belirtmek`** için bayrağı ve zamanlama politikasını belirtmek için bayrağı kullanabiliriz.

Örneğin, 1 zamanlama penceresi ve SATF ilkesiyle 1000 isteklik bir iş yükü oluşturmak için aşağıdaki komutu kullanabiliriz:

```
fio --name=workload --size=1G --iodepth=1000 --numjobs=1 --rw=randrw --bs=4k --
ioengine=libaio --direct=1 --group_reporting --norandommap --refill_buffers --
time_based --runtime=60 --output=workload.log --c=1 --w=satf
```

9. Bir programlayıcıda açlıktan kaçınmak önemlidir. SATF gibi bir poliçe verildiğinde belirli bir sektörün çok uzun süre ertelenmesi gibi bir dizi talep aklınıza geliyor mu? Bu sıra göz önüne alındığında, sınırlı bir SATF veya BSATF zamanlama yaklaşımı kullanırsanız nasıl bir performans gösterir? Bu yaklaşımda, zamanlama penceresini (örn. -w 4) ve BSATF ilkesini (-p BSATF) belirtirsiniz; zamanlayıcı, yalnızca geçerli penceredeki tüm isteklere hizmet verildiğinde sonraki istek penceresine geçecektir. Bu açlık sorununu çözüyor mu? SATF ile karşılaştırıldığında performansı nasıl? Genel olarak, bir disk performans ve açlıktan kaçınma arasındaki bu değiş tokuşu nasıl yapmalıdır?

SATF çizelgeleme politikası ile belirli bir sektörün uzun süre ertelenmesine neden olacak bir dizi istek oluşturmak için, aynı anda diğer sektörler için istekler gönderirken, tamamı o sektörü hedefleyen bir dizi istek gönderebiliriz. Bu, planlayıcının sürekli olarak hedef sektöre yönelik taleplere hizmet verme ile diğer sektörlere yönelik taleplere hizmet verme arasında geçiş yapmasına ve hedef sektöre yönelik isteklerin gecikmesine neden olur.

Sınırlı bir SATF veya BSATF zamanlama yaklaşımı kullanmak, belirli bir pencerede hizmet verilebilecek isteklerin sayısını sınırlayarak potansiyel olarak açlık sorununu çözebilir. Örneğin, zamanlama penceresi 4'e ayarlanırsa ve BSATF ilkesi kullanılırsa, zamanlayıcı bir sonraki pencereye geçmeden önce bir seferde en fazla 4 isteğe hizmet verir. Bu, tek bir sektörün çok sayıda istekle hedeflenmesini önleyecektir, çünkü hedeflenen sektör için tüm isteklere hizmet verilebilmesi için planlayıcı bir sonraki pencereye geçecektir.

BSATF programlayıcının SATF programlayıcıya kıyasla performansı, belirli iş yüküne ve planlama penceresinin boyutuna bağlı olacaktır. Genel olarak, daha büyük bir zamanlama penceresi, aynı anda daha fazla talebin karşılanmasına izin verdiği için daha iyi performansa yol açabilir. Ancak, çok sayıda istekle tek bir sektör hedeflenirse bu, açlığın artmasına da yol açabilir.

Genel olarak, bir disk, zamanlama algoritmasını ve kullanılacak zamanlama parametrelerini dikkatlice seçerek performans ve açlıktan kaçınma arasındaki dengeyi sağlamaya çalışmalıdır. Örneğin, bir disk, çok sayıda küçük istek içeren iş yükleri için SATF gibi daha ayrıntılı bir zamanlama yaklaşımı kullanabilir ve daha az sayıda daha büyük istek içeren iş yükleri için BSATF gibi daha ayrıntılı bir yaklaşım kullanabilir. İstekler. Ek olarak disk, seçilen zamanlama yaklaşımının performans ve açlık özelliklerini izleyebilir ve istenen dengeyi elde etmek için gereken parametreleri ayarlayabilir.

10. Şimdiye kadar incelediğimiz tüm planlama politikaları, bir dizi istek üzerinden en uygun programı aramak yerine basitçe bir sonraki en iyi seçeneği seçtikleri için açgözlüdür. Bu açgözlü yaklaşımın optimal olmadığı bir dizi istek bulabilir misiniz?

Evet, açgözlü yaklaşımın optimal olmadığı bir dizi istek bulmak mümkündür.

Bir örnek verecek olursak;

Başlangıç ve bitiş zamanları aşağıdaki gibi olan üç talebimiz olduğunu varsayalım:

- İstek 1: başlangıç zamanı 0, bitiş zamanı 3
- Talep 2: başlangıç zamanı 2, bitiş zamanı 5
- İstek 3: başlangıç zamanı 4, bitiş zamanı 7

Açgözlü yaklaşımı kullanarak, önce İstek 1'i, ardından İstek 2'yi ve son olarak İstek 3'ü planlardık. Alınan toplam süre 7 olur.

Ancak bunun yerine önce İstek 3'ü, ardından İstek 1'i ve son olarak İstek 2'yi planlarsak, alınan toplam süre 6 olur. Daha az zaman aldığı için bu daha iyi bir programdır.

Bu nedenle, bu durumda açgözlü yaklaşım optimal değildir.

Açgözlü yaklaşımın her zaman yetersiz olmayabileceğini ve birçok durumda iyi bir çözüm üretebileceğini bilmeliyiz. Ancak, optimum çözümü bulamadığı durumlar vardır ve alternatif yaklaşımları da değerlendirmek önemlidir.