Andrew Dosya Sistemi (AFS)

Andrew Dosya Sistemi, 1980'lerde [H+88] Carnegie-Mellon Üniversitesi'nde (CMU) 1 tanıtıldı. Carnegie-Mellon Üniversitesi'nden (kısaca "Satya") tanınmış Profesör M. Satyanarayanan liderliğindeki ana
Bu projenin amacı basitti: **Ölçek(scale)**. Spesifik olarak, dağıtılmış bir dosya sistemi, bir sunucunun olabildiğince çok istemciyi destekleyebileceği şekilde nasıl tasarlanabilir?
İlginç bir şekilde, tasarım ve uygulamanın ölçeklenebilirliği etkileyen çok sayıda yönü vardır. En önemlisi, istemciler ve sunucular arasındaki protokolün tasarımıdır. Örneğin, NFS'de **protokol(protocol)**, istemcileri, önbelleğe alınan içeriklerin değişip değişmediğini belirlemek için düzenli olarak sunucuyla kontrol etmeye zorlar; her kontrol sunucu kaynaklarını (CPU ve ağ bant genişliği dahil) kullandığından, bunun gibi sık kontroller bir sunucunun yanıt verebileceği istemci sayısını sınırlayacak ve böylece ölçeklenebilirliği sınırlayacaktır.
AFS aynı zamanda NFS'den farklıdır, çünkü başından beri makul, kullanıcı tarafından görülebilen davranış birinci sınıf bir endişeydi. NFS'de, önbellek tutarlılığını açıklamak zordur çünkü bu, istemci tarafı önbellek zaman aşımı aralıkları da dahil olmak üzere doğrudan düşük düzeyli uygulama ayrıntılarına bağlıdır. AFS'de önbellek tutarlılığı basittir ve kolayca anlaşılır: dosya açıldığında, bir istemci genellikle sunucudan en son tutarlı kopyayı alır.

- AFS Versiyon 1

AFS'nin iki versiyonunu [H+88, S+85] tartışacağız. İlk sürüm (AFSv1 olarak adlandıracağız, ancak aslında orijinal sistem ITC dağıtılmış dosya sistemi [S+85] olarak adlandırılıyordu) bazı temel tasarımlara sahipti, ancak istenildiği gibi ölçeklenmedi, bu da yeniden -tasarım ve son protokol (AFSv2 veya sadece AFS olarak adlandıracağız) [H+88]. Şimdi ilk versiyonu tartışıyoruz.

Başlangıçta "Carnegie-Mellon Üniversitesi" olarak anılmasına rağmen, CMU daha sonra kısa çizgiyi kaldırdı ve böylece modern biçim olan "Carnegie Mellon Üniversitesi" doğdu. 80'lerin başındaki çalışmalardan türetilen AFS olarak, CMU'yu orijinal tam tireli biçiminde ifade ediyoruz. Gerçekten sıkıcı ayrıntılarla ilgileniyorsanız, daha fazla ayrıntı için https://www.quora.com/When-did-Carnegie-Mellon-University-remove-the-hyphen-in-the-university-name adresine bakın


```
Test Yetkilendirmesi  Bir dosyanın değişip değişmediğini test
edin(önbelleğe alınmış girişleri doğrulamak için kullanılır).


Dosya Durumunu Al     Bir dosya için istatistik bilgisini al
              Dosyanın içeriğini getir.
Mağaza        Bu dosyayı sunucuda saklayın.
Dosya Statünü Ayarla   Bir dosya için istatistik
              bilgisini ayarlayın.
Liste Dizini   Bir dizinin içeriğini listeleme
```

Şekil 50.1: **AFSv1 Protokolünde Öne Çıkanlar**

AFS'nin tüm sürümlerinin temel ilkelerinden biri, bir dosyaya erişen istemci makinenin yerel diskinde tüm dosyayı **önbelleğe almaktır(whole-file caching).** Bir dosyayı açtığınızda(), dosyanın tamamı (varsa) sunucudan alınır ve yerel diskinizdeki bir dosyada saklanır. Sonraki uygulama okuma() ve yazma() işlemleri, dosyanın depolandığı yerel dosya sistemine yönlendirilir; bu nedenle, bu işlemler ağ iletişimi gerektirmez ve hızlıdır. Son olarak, kapatıldığında(), dosya (değiştirilmişse) sunucuya geri boşaltılır. Blokları önbelleğe alan (tüm dosyaları değil, NFS elbette tüm dosyanın her bloğunu önbelleğe alabilir) ve bunu istemci belleğinde (yerel diskte değil) yapan NFS ile bariz zıtlıklara dikkat edin.

Ayrıntılara biraz daha girelim. Bir istemci uygulaması ilk kez open() öğesini çağırdığında, AFS istemci tarafı kodu (AFS tasarımcıları buna **Venüs(Venus)** adını verir) sunucuya bir Fetch protokol mesajı gönderir. Fetch protokolü mesajı, istenen dosyanın tüm yol adını (örneğin, /home/remzi/notes.txt) dosya sunucusuna (Vice olarak adlandırdıkları grup) iletir ve daha sonra yol adını geçer, bulur. istenen dosyayı seçin ve tüm dosyayı müşteriye geri gönderin. İstemci tarafı kodu daha sonra dosyayı istemcinin yerel diskinde önbelleğe alır (yerel diske yazarak). Yukarıda söylediğimiz gibi, sonraki read() ve write() sistem çağrıları AFS'de kesinlikle yereldir (sunucu ile hiçbir iletişim gerçekleşmez); sadece dosyanın yerel kopyasına yönlendirilirler. read() ve write() çağrıları yerel bir dosya sistemine yapılan çağrılar gibi hareket ettiğinden, bir bloğa erişildikten sonra istemci belleğinde de önbelleğe alınabilir. Bu nedenle AFS, yerel diskinde bulunan blokların kopyalarını önbelleğe almak için istemci belleğini de kullanır. Son olarak, bittiğinde, AFS istemcisi dosyanın değiştirilip değiştirilmediğini (yani, yazmak için açılıp açılmadığını) kontrol eder; öyleyse, kalıcı depolama için tüm dosya ve yol adını sunucuya göndererek yeni sürümü bir Mağaza protokolü mesajıyla sunucuya geri gönderir.

Dosyaya bir sonraki erişimde, AFSv1 bunu çok daha verimli bir şekilde yapar. Spesifik olarak, istemci tarafı kodu, dosyanın değişip değişmediğini belirlemek için önce sunucuyla bağlantı kurar (TestAuth protokol mesajını kullanarak). Değilse, istemci yerel olarak önbelleğe alınan kopyayı kullanır ve böylece bir ağ aktarımından kaçınarak performansı artırır. Yukarıdaki şekil, AFSv1'deki bazı protokol mesajlarını göstermektedir. Protokolün bu erken sürümünün yalnızca dosya içeriklerini önbelleğe aldığını unutmayın; örneğin dizinler yalnızca sunucuda tutuluyordu.

İPUCU: ÖNLEYİN SONRA YAPIN (PATTERSON YASASI)

Danışmanlarımızdan biri olan David Patterson (RISC ve RAID ünlüsü), söz konusu sorunu çözmek için yeni bir sistem kurmadan önce bir sistemi ölçmemiz ve sorunu göstermemiz için bizi her zaman teşvik ederdi. İçgüdü yerine deneysel kanıtlar kullanarak, sistem kurma sürecini daha bilimsel bir çabaya dönüştürebilirsiniz. Bunu yapmanın ayrıca, geliştirilmiş sürümünüz geliştirilmeden önce sistemi tam olarak nasıl ölçeceğinizi düşünmenizi sağlamak gibi yan faydaları da vardır. Sonunda yeni sistemi kurmaya başladığınızda, sonuç olarak iki şey daha iyidir: Birincisi, gerçek bir sorunu çözdüğünüzü gösteren kanıtınız var; ikincisi, artık yeni sisteminizi yerinde ölçmenin, en son teknolojiyi gerçekten iyileştirdiğini göstermenin bir yolunu buldunuz. Ve biz buna **Patterson Yasası(Patterson's Law)** diyoruz.

- Sürüm 1 ile ilgili sorunlar

AFS'nin bu ilk sürümüyle ilgili birkaç önemli sorun, tasarımcıları dosya sistemlerini yeniden düşünmeye sevk etti. Sorunları ayrıntılı olarak incelemek için, AFS tasarımcıları neyin yanlış olduğunu bulmak için mevcut prototiplerini **ölçmek(measurement)** için çok zaman harcadılar. Bu tür deneyler iyi bir şeydir çünkü ölçüm, sistemlerin nasıl çalıştığını ve onları nasıl geliştireceğimizi anlamanın anahtarıdır; somut, iyi verilerin elde edilmesi bu nedenle sistem inşasının gerekli bir parçasıdır. Yazarlar çalışmalarında AFSv1 ile ilgili iki ana sorun buldular:

- **Yol geçiş maliyetleri çok yüksek(Path-traversal costs are too high):** Bir Fetch or Store protokol isteği gerçekleştirirken, istemci tüm yol adını (ör. /home/remzi/notes.txt) sunucuya iletir. Sunucu, dosyaya erişmek için, tam bir yol adı geçişi gerçekleştirmeli, önce evi bulmak için kök dizine, ardından remzi'yi bulmak için evde aramalı ve bu şekilde, en sonunda istenen dosya olana kadar yol boyunca devam etmelidir. bulunan Birçok istemcinin sunucuya aynı anda erişmesiyle, AFS tasarımcıları, sunucunun CPU zamanının çoğunu yalnızca dizin yollarında yürümekle geçirdiğini gördüler.
- **İstemci çok fazla TestAuth protokol mesajı veriyor(Client issues too many TestAuth protocol messages):** NFS ve onun GETATTR protokol mesajlarının fazlalığı gibi, AFSv1 de yerel bir dosyanın (veya onun istatistik bilgilerinin) TestAuth protokol mesajıyla geçerli olup olmadığını kontrol etmek için büyük miktarda trafik oluşturdu. Bu nedenle, sunucular zamanlarının çoğunu istemcilere bir dosyanın önbelleğe alınmış kopyalarını kullanmanın uygun olup olmadığını anlatmakla geçirdiler. Çoğu zaman, yanıt dosyanın değişmediğiydi.

AFSv1 ile ilgili aslında iki sorun daha vardı: yük, sunucular arasında dengelenmemişti ve sunucu, istemci başına tek bir ayrı işlem kullanıyordu, bu da içerik değiştirme ve diğer ek yüklere neden oluyordu. Yük

dengesizlik sorunu, bir yöneticinin yükü dengelemek için sunucular arasında taşıyabileceği **hacimler(volumes)** getirilerek çözüldü; AFSv2'de bağlam değiştirme sorunu, sunucunun işlemler yerine iş parçacığı ile oluşturulmasıyla çözüldü. Bununla birlikte, alan adına, burada sistemin ölçeğini sınırlayan yukarıdaki iki ana protokol sorununa odaklanıyoruz.

- Protokolün Geliştirilmesi

Yukarıdaki iki sorun, AFS'nin ölçeklenebilirliğini sınırladı; sunucu CPU'su sistemin darboğazı haline geldi ve her sunucu aşırı yüklenmeden yalnızca 20 istemciye hizmet verebildi. Sunucular çok fazla TestAuth mesajı alıyordu ve Fetch veya Store mesajları aldıklarında dizin hiyerarşisinde gezinmek için çok fazla zaman harcıyorlardı. Böylece, AFS tasarımcıları bir sorunla karşı karşıya kaldılar:

CRUX: **Ö**LÇEKLENEBİLİR **B**İR **D**OSYA **P**ROTOKOLÜ **N**ASIL **T**ASARLANIR

Sunucu etkileşimlerinin sayısını en aza indirmek için protokol nasıl yeniden tasarlanmalı, yani TestAuth mesajlarının sayısı nasıl azaltılabilir? Ayrıca, bu sunucu etkileşimlerini verimli kılmak için protokolü nasıl tasarlayabilirler? Bu sorunların her ikisine de saldıran yeni bir protokol, çok daha ölçeklenebilir bir AFS sürümü ile sonuçlanacaktır.

- AFS Sürüm 2

AFSv2, istemci/sunucu etkileşimlerinin sayısını azaltmak için **geri arama(callback)** kavramını tanıttı. Geri arama, sunucudan istemciye, istemcinin önbelleğe aldığı bir dosya değiştirildiğinde sunucunun istemciyi bilgilendireceğine dair basit bir sözdür. **Bu durumu(state)** sisteme ekleyerek, istemcinin artık önbelleğe alınmış bir dosyanın hala geçerli olup olmadığını öğrenmek için sunucuyla iletişime geçmesi gerekmez. Bunun yerine, sunucu aksini söyleyene kadar dosyanın geçerli olduğunu varsayar; **yoklamaya(polling)** karşı kesintiler**(interrupts)** arasındaki analojiye dikkat edin.

AFSv2 ayrıca bir istemcinin hangi dosyayla ilgilendiğini belirtmek için yol adları yerine bir **dosya tanımlayıcısı(file identifier)** (FID) (NFS **dosya tanıtıcısına(file handle)** benzer) kavramını da tanıttı. AFS'deki bir FID, bir birim tanımlayıcısı, bir dosya tanımlayıcısı ve bir " uniquifier" (bir dosya silindiğinde birimin ve dosya kimliklerinin yeniden kullanılmasını sağlamak için). Böylece, tüm yol adlarını sunucuya göndermek ve sunucunun istenen dosyayı bulmak için yol adını yürümesine izin vermek yerine, müşteri yol adını her seferinde tek parça olarak yürütür, sonuçları önbelleğe alır ve böylece sunucu üzerindeki yükü umarım azaltır.

Örneğin, bir istemci /home/remzi/notes.txt dosyasına eriştiyse ve home, / üzerine bağlanan AFS diziniyse (yani, / yerel kök dizindi, ancak home ve alt dizini AFS'deyse), istemci önce ana sayfanın dizin içeriğini alın, bunları yerel disk önbelleğine koyun ve evde bir geri arama ayarlayın. Ardından, müşteri dizini

Client (C_1)	**Server**
fd = open("/home/remzi/notes.txt", ...);	
Send Fetch (home FID, "remzi")	
	Al Yanıtı getir remzi'yi yerel disk önbelleğine yaz, remzi Send Fetch'in (remzi FID, "notes.txt") geri arama durumu kaydı
	Al Yanıtı getir note.txt'yi yerel diske yaz önbellek kaydı note.txt'nin geri arama durumu önbelleğe alınmış note.txt'nin yerel açık()'ı uygulamaya dosya tanıtıcısını döndürür
Getirme isteği al remzi'yi ev dizininde ara remzi'de geri arama(C1) oluştur remzi'nin içeriğini ve FID'yi iade et	

Getirme isteği al
remzi dizininde note.txt'yi arayın, note.txt'de geri arama (C1) kurun, note.txt'nin içeriğini ve FID'yi döndürün

read(fd, buffer, MAX);
perform local `read()` on cached copy

close(fd);
do local `close()` on cached copyif file has changed, flush to server

fd = open("/home/remzi/notes.txt", ...);
Foreach dir (home, remzi)
if (callback(dir) == VALID)
use local copy for lookup(dir)else
Fetch (as above)
if (callback(notes.txt) == VALID)open local cached copy
return file descriptor to itelse
Fetch (as above) then open and return fd

Şekil 50.2: **Dosya Okuma: İstemci Tarafı ve Dosya Sunucusu Eylemleri**

remzi, yerel disk önbelleğine koyun ve remzi'de bir geri arama ayarlayın. Son olarak müşteri, note.txt'yi alır, bu normal dosyayı yerel diskte önbelleğe alır, bir geri arama ayarlar ve son olarak çağıran uygulamaya bir dosya tanıtıcı döndürür. Özet için bkz. Şekil 50.2.

Bununla birlikte, NFS'den temel farkı, bir dizin veya dosyanın her getirilmesinde, AFS istemcisinin sunucuyla bir geri arama oluşturmasıdır.

KENARA: ÖNBELLEK TUTARLILIĞI HER DERE DEDE DEĞİLDİR

Dağıtılmış dosya sistemlerini tartışırken, dosya sistemlerinin sağladığı önbellek tutarlılığından çok şey yapılır. Ancak bu temel tutarlılık, birden çok istemciden dosya erişimine ilişkin tüm sorunları çözmez. Örneğin, birden çok istemcinin kodu teslim etme ve teslim alma işlemlerini gerçekleştirdiği bir kod deposu oluşturuyorsanız, tüm işi sizin yerinize yapması için temeldeki dosya sistemine güvenemezsiniz; bunun yerine, bu tür eşzamanlı erişimler gerçekleştiğinde "doğru" şeyin gerçekleşmesini sağlamak için açık **dosya düzeyinde kilitleme(file-level locking)** kullanmanız gerekir. Aslında, eşzamanlı güncellemeleri gerçekten önemseyen herhangi bir uygulama, çakışmaların üstesinden gelmek için ekstra makineler ekleyecektir. Bu bölümde ve bir öncekinde açıklanan temel tutarlılık, öncelikle gündelik kullanım için yararlıdır, yani bir kullanıcı farklı bir istemcide oturum açtığında, dosyalarının makul bir sürümünün orada görünmesini bekler. Bu protokollerden daha fazlasını beklemek, kendinizi başarısızlığa, hayal kırıklığına ve gözyaşlarıyla dolu hayal kırıklığına hazırlıyor.

böylece sunucunun önbelleğe alınmış durumundaki bir değişikliği istemciye bildirmesini sağlamak. Fayda açıktır: /home/remzi/notes.txt dosyasına ilk erişim birçok istemci-sunucu mesajı oluştursa da (yukarıda açıklandığı gibi), aynı zamanda tüm dizinlerin yanı sıra note.txt dosyası için geri aramalar oluşturur ve böylece sonraki erişimler tamamen yereldir ve hiçbir sunucu etkileşimi gerektirmez. Bu nedenle, istemcide bir dosyanın önbelleğe alındığı yaygın durumda, AFS yerel disk tabanlı bir dosya sistemiyle neredeyse aynı şekilde davranır. Bir dosyaya birden çok kez erişilirse, ikinci erişim bir dosyaya yerel olarak erişmek kadar hızlı olmalıdır.

- Önbellek Tutarlılığı

NFS'den bahsederken, önbellek tutarlılığının dikkate aldığımız iki yönü vardı: **güncelleme görünürlüğü(update visibility)** ve **önbellek bayatlığı(cache staleness)**. Güncelleme görünürlüğü ile soru şudur: sunucu bir dosyanın yeni sürümü ile ne zaman güncellenecektir? Önbellek bayatlığı ile ilgili soru şudur: Sunucu yeni bir sürüme sahip olduğunda, istemcilerin önbelleğe alınmış eski bir kopya yerine yeni sürümü görmesi ne kadar sürer?

Geri aramalar ve tam dosya önbelleğe alma nedeniyle, AFS tarafından sağlanan önbellek tutarlılığının tanımlanması ve anlaşılması kolaydır. Dikkate alınması gereken iki önemli durum vardır: farklı makinelerdeki işlemler arasındaki tutarlılık ve aynı makinedeki işlemler arasındaki tutarlılık.

AFS, farklı makineler arasında güncellemeleri sunucuda görünür kılar ve önbelleğe alınmış kopyaları aynı anda, yani güncellenen dosya kapatıldığında geçersiz kılar. İstemci bir dosyayı açar ve ardından ona yazar (belki art arda). Sonunda kapatıldığında, yeni dosya sunucuya aktarılır (ve böylece görünür). Bu noktada sunucu, önbelleğe alınmış kopyaları olan tüm istemciler için geri aramaları "keser"; mola, her müşteriyle iletişime geçilerek ve dosyada sahip olduğu geri aramanın artık geçerli olmadığı bildirilerek gerçekleştirilir.

Şekil 50.3: **Önbellek Tutarlılık Zaman Çizelgesi**

geçerli. Bu adım, istemcilerin artık dosyanın eski kopyalarını okumamasını sağlar; bu istemcilerde sonraki açmalar, dosyanın yeni sürümünün sunucudan yeniden getirilmesini gerektirecektir (ve ayrıca dosyanın yeni sürümünde yeniden bir geri arama oluşturmaya da hizmet edecektir).

AFS, aynı makinedeki işlemler arasında bu basit modele bir istisna yapar. Bu durumda, bir dosyaya yazılanlar diğer yerel işlemler tarafından hemen görülebilir (yani, bir işlemin en son güncellemelerini görmek için bir dosya kapatılana kadar beklemesi gerekmez). Bu, tek bir makine kullanımının tam olarak beklediğiniz gibi davranmasını sağlar, çünkü bu davranış tipik UNIX semantiğine dayanır. Yalnızca farklı bir makineye geçerken daha genel AFS tutarlılık mekanizmasını tespit edebilirsiniz.

Daha fazla tartışmaya değer ilginç bir makineler arası durum var. Spesifik olarak, farklı makinelerdeki süreçlerin aynı anda bir dosyayı değiştirdiği ender bir durumda, AFS doğal olarak **son yazan kazanır(last writer wins)** yaklaşımı olarak bilinen yaklaşımı kullanır (buna belki de **son yaklaşan kazanır(last closer wins)** denmelidir). Spesifik olarak, hangi müşteri en son close()'u çağırırsa sunucudaki tüm dosyayı en son günceller ve böylece "kazanan" olur.

dosya, yani başkalarının görmesi için sunucuda kalan dosya. Sonuç, bir istemci veya diğeri tarafından bütünüyle oluşturulmuş bir dosyadır. NFS gibi blok tabanlı bir protokolden farkı not edin: NFS'de, her istemci dosyayı güncellerken ayrı blokların yazma işlemleri sunucuya aktarılabilir ve bu nedenle sunucudaki son dosya bir karışım olarak sonuçlanabilir. her iki istemciden gelen güncellemelerin sayısı. Çoğu durumda, böyle karışık bir dosya çıktısı pek anlamlı olmaz, yani bir JPEG görüntüsünün iki müşteri tarafından parçalar halinde değiştirildiğini hayal edin; sonuçta ortaya çıkan yazma karışımı muhtemelen geçerli bir JPEG oluşturmayacaktır.

Bu farklı senaryolardan birkaçını gösteren bir zaman çizelgesi Şekil 50.3'te görülebilir. Sütunlar, İstemci1'deki iki işlemin (P1 ve P2) davranışını ve onun önbellek durumunu, İstemci2'deki bir işlemin (P3) ve onun önbellek durumunu ve sunucunun (Sunucu) davranışını, tümü hayali olarak adlandırılan tek bir dosya üzerinde çalışır. , F. Sunucu için şekil, soldaki işlem tamamlandıktan sonra dosyanın içeriğini gösterir. Baştan sona okuyun ve her okumanın neden yaptığı sonuçları döndürdüğünü anlayıp anlayamadığınızı görün. Sağdaki bir yorum alanı, takılırsanız size yardımcı olacaktır.

- Kilitlenme Kurtarma

  Yukarıdaki açıklamadan, kilitlenme kurtarmanın NFS'den daha karmaşık olduğunu hissedebilirsiniz. Haklısın. Örneğin, bir sunucunun (S) bir istemciyle (C1) iletişim kuramadığı kısa bir süre olduğunu hayal edin, örneğin C1 istemcisi yeniden başlatılırken. C1 mevcut değilken, S ona bir veya daha fazla geri arama geri çağırma mesajı göndermeye çalışmış olabilir; örneğin, C1'in F dosyasını kendi yerel diskinde önbelleğe aldığını ve ardından C2'nin (başka bir istemci) F'yi güncelleştirdiğini, böylece S'nin dosyayı yerel önbelleklerinden kaldırmak için önbelleğe alan tüm istemcilere mesaj göndermesine neden olduğunu hayal edin. C1, yeniden başlatılırken bu kritik mesajları kaçırabileceğinden, sisteme yeniden katıldığında C1'in tüm önbellek içeriğini şüpheli olarak ele alması gerekir. Bu nedenle, F dosyasına bir sonraki erişimde, C1 önce sunucuya (bir TestAuth protokol mesajı ile) F dosyasının önbelleğe alınmış kopyasının hala geçerli olup olmadığını sormalıdır; öyleyse, C1 kullanabilir; değilse, C1 sunucudan daha yeni sürümü getirmelidir.

  Bir çökmeden sonra sunucu kurtarma da daha karmaşıktır. Ortaya çıkan sorun, geri aramaların bellekte tutulmasıdır; bu nedenle, bir sunucu yeniden başlatıldığında, hangi istemci makinenin hangi dosyalara sahip olduğu hakkında hiçbir fikri yoktur. Bu nedenle, sunucu yeniden başlatıldığında, sunucunun her müşterisi sunucunun çöktüğünü fark etmeli ve tüm önbellek içeriklerini şüpheli olarak değerlendirmeli ve

(yukarıdaki gibi) kullanmadan önce bir dosyanın geçerliliğini yeniden sağlamalıdır. Bu nedenle, sunucu çökmesi büyük bir olaydır, çünkü her müşterinin çökmeden zamanında haberdar olması veya bir istemcinin eski bir dosyaya erişmesi riskini alması gerekir. Böyle bir kurtarmayı uygulamanın birçok yolu vardır; örneğin, sunucu yeniden çalışır durumdayken her istemciye bir mesaj ("önbellek içeriğinize güvenmeyin!" diyen!) , Olarak adlandırılan). Gördüğünüz gibi, daha ölçeklenebilir ve mantıklı bir önbelleğe alma modeli oluşturmanın bir maliyeti var; NFS ile istemciler bir sunucu çökmesini neredeyse hiç fark etmediler.

İş yoğunluğu	NFS	AFS	AFS / NFS
1. Küçük dosya, sıralı okuma	$N_s \cdot L_{net}$	$N_s \cdot L_{net}$	1
2. Küçük dosya, sıralı yeniden okuma	$N_s \cdot L_{mem}$	$N_s \cdot L_{mem}$	1
3. Orta dosya, sıralı okuma	$N_m \cdot L_{net}$	$N_m \cdot L_{net}$	1
4. Orta dosya, sıralı yeniden okuma	$N_m \cdot L_{mem}$	$N_m \cdot L_{mem}$	1
5. Büyük dosya, sıralı okuma	$N_L \cdot L_{net}$	$N_L \cdot L_{net}$	1
6. Büyük dosya, sıralı yeniden okuma	$N_L \cdot L_{net}$	$N_L \cdot L_{disk}$	$\frac{L_{disk}}{L_{net}}$
7. Büyük dosya, tek okuma	L_{net}	$N_L \cdot L_{net}$	N_L
8. Küçük dosya, sıralı yazma	$N_s \cdot L_{net}$	$N_s \cdot L_{net}$	1
9. Büyük dosya, sıralı yazma	$N_L \cdot L_{net}$	$N_L \cdot L_{net}$	1
10. Büyük dosya, sıralı üzerine yazma	$N_L \cdot L_{net}$	$2 \cdot N_L \cdot L_{net}$	2
11. Büyük dosya, tek yazma	L_{net}	$2 \cdot N_L \cdot L_{net}$	$2 \cdot N_L$

Şekil 50.4: **Karşılaştırma: AFS ve NFS**

- AFSv2'nin Ölçeği ve Performansı

Yeni protokol yürürlükteyken, AFSv2 ölçüldü ve orijinal sürümden çok daha ölçeklenebilir olduğu bulundu. Aslında, her sunucu yaklaşık 50 istemciyi destekleyebilir (yalnızca 20 yerine). Diğer bir fayda da, istemci tarafı performansının genellikle yerel performansa oldukça yaklaşmasıydı, çünkü ortak durumda, tüm dosya erişimleri yereldi; dosya okumaları genellikle yerel disk önbelleğine (ve potansiyel olarak yerel belleğe) gitti. Yalnızca bir müşteri yeni bir dosya oluşturduğunda veya mevcut bir dosyaya yazdığında, sunucuya bir Mağaza mesajı gönderme ve böylece dosyayı yeni içeriklerle güncelleme ihtiyacı vardı. Ayrıca yaygın dosya sistemi erişim senaryolarını NFS ile karşılaştırarak AFS performansı hakkında bir bakış açısı kazanalım. Şekil 50.4 (sayfa 9)nitel karşılaştırmamızın sonuçlarını gösterir.

Şekilde, farklı boyutlardaki dosyalar için tipik okuma ve yazma kalıplarını analitik olarak inceliyoruz. Küçük dosyaların içinde Ns blokları vardır; orta dosyalar Nm bloklara sahiptir; büyük dosyalar NL bloklarına sahiptir. Küçük ve orta ölçekli dosyaların bir müşterinin belleğine sığdığını varsayıyoruz; büyük dosyalar yerel bir diske sığar ancak istemci belleğine sığmaz.

Ayrıca, analiz amacıyla, bir dosya bloğu için ağ üzerinden uzak sunucuya erişimin Lnet zaman birimleri aldığını varsayıyoruz. Yerel belleğe erişim için Lmem gerekir ve yerel diske erişim için Ldisk gerekir. Genel varsayım şudur: Lnet > Ldisk > Lmem.

Son olarak, bir dosyaya ilk erişimin herhangi bir önbelleğe çarpmadığını varsayıyoruz. İlgili önbellek dosyayı tutmak için yeterli kapasiteye sahipse, sonraki dosya erişimlerinin (yani "yeniden okumalar") önbelleklerde etkili olacağını varsayıyoruz.

Şeklin sütunları, belirli bir işlemin (örneğin, küçük bir sıralı dosya okuması) kabaca NFS veya AFS'de aldığı süreyi gösterir. En sağdaki sütun, AFS'nin NFS'ye oranını gösterir.

Aşağıdaki gözlemleri yapıyoruz. İlk olarak, birçok durumda, her sistemin performansı kabaca eşdeğerdir. Örneğin, bir dosyayı ilk kez okurken (örn. İş Yükleri 1, 3, 5), dosyayı yeniden alma zamanı

mote sunucusu hakimdir ve her iki sistemde de benzerdir. Dosyayı yerel diske yazması gerektiğinden AFS'nin bu durumda daha yavaş olacağını düşünebilirsiniz; ancak, bu yazma işlemleri yerel (istemci tarafı) dosya sistemi önbelleği tarafından arabelleğe alınır ve bu nedenle söz konusu maliyetler muhtemelen gizlenir. Benzer şekilde, yerel önbelleğe alınmış kopyadan AFS okumalarının daha yavaş olacağını düşünebilirsiniz, çünkü yine AFS önbelleğe alınmış kopyayı diskte depolar. Ancak, AFS burada yine yerel dosya sistemi önbelleğinden yararlanır; AFS'deki okumalar büyük olasılıkla istemci tarafı bellek önbelleğine isabet eder ve performans NFS'ye benzer olur.

İkinci olarak, büyük dosya sıralı yeniden okuma sırasında (İş Yükü 6) ilginç bir fark ortaya çıkar. AFS büyük bir yerel disk önbelleğine sahip olduğundan, dosyaya tekrar erişildiğinde dosyaya oradan

erişecektir. NFS, aksine, yalnızca istemci belleğindeki blokları önbelleğe alabilir; sonuç olarak, büyük bir dosya (yani, yerel bellekten daha büyük bir dosya) yeniden okunursa, NFS istemcisinin tüm dosyayı uzak sunucudan yeniden getirmesi gerekir. Bu nedenle, AFS, uzaktan erişimin gerçekten daha yavaş olduğu varsayılarak, bu durumda Lnet faktörü kadar NFS'den daha hızlıdır.

yerel diskten daha Ayrıca, bu durumda NFS'nin ölçek üzerinde de etkisi olan sunucu yükünü artırdığını not ediyoruz.

Üçüncüsü, sıralı yazma işlemlerinin (yeni dosyaların) her iki sistemde de benzer şekilde gerçekleştirilmesi gerektiğini not ediyoruz (İş Yükleri 8, 9). AFS, bu durumda dosyayı yerel önbelleğe alınmış kopyaya yazacaktır; dosya kapatıldığında, AFS istemcisi protokole göre sunucuya yazmaları zorlar. NFS, istemci tarafındaki bellek baskısı nedeniyle bazı blokları sunucuya zorlayarak istemci belleğindeki yazmaları arabelleğe alacaktır, ancak NFS kapatıldığında temizleme tutarlılığını korumak için dosya kapatıldığında bunları kesinlikle sunucuya yazacaktır. Tüm verileri yerel diske yazdığı için AFS'nin burada daha yavaş olacağını düşünebilirsiniz. Ancak, yerel bir dosya sistemine yazdığının farkına varın; bu yazma işlemleri önce sayfa önbelleğine ve yalnızca daha sonra (arka planda) diske işlenir ve bu nedenle AFS, performansı artırmak için istemci tarafı işletim sistemi bellek önbelleğe alma altyapısının avantajlarından yararlanır.

Dördüncüsü, AFS'nin sıralı dosya üzerine yazma işleminde daha kötü performans gösterdiğini not ediyoruz (İş Yükü 10). Şimdiye kadar yazan iş yüklerinin de yeni bir dosya oluşturduğunu varsaydık; bu durumda, dosya vardır ve üzerine yazılır. Üzerine yazma, AFS için özellikle kötü bir durum olabilir, çünkü istemci önce eski dosyayı bütünüyle getirir, sonra üzerine yazar. NFS, aksine, basitçe blokların üzerine yazacak ve böylece

ilk (işe yaramaz) read2.

Son olarak, büyük dosyalar içindeki küçük bir veri alt kümesine erişen iş yükleri, NFS'de AFS'den çok daha iyi performans gösterir (İş yükleri 7, 11). Bu durumlarda, dosya açıldığında AFS protokolü tüm dosyayı getirir; ne yazık ki, yalnızca küçük bir okuma veya yazma işlemi gerçekleştirilir. Daha da kötüsü, eğer dosya değiştirilirse, dosyanın tamamı sunucuya geri yazılır ve per-

[2]Burada, NFS yazma işlemlerinin blok boyutunda ve blok hizalı olduğunu varsayıyoruz; değillerse, NFS istemcisinin de önce bloğu okuması gerekirdi. Ayrıca dosyanın O TRUNC bayrağıyla açılmadığını varsayıyoruz; öyle olsaydı, AFS'deki ilk açılış, yakında kesilecek olan dosyanın içeriğini getirmezdi.

KENAR: İŞ YÜKÜNÜN ÖNEMİ

Herhangi bir sistemi değerlendirmenin zorluklarından biri, **iş yükü(workload)** seçimidir. Bilgisayar sistemleri çok farklı şekillerde kullanıldığından, aralarından seçim yapabileceğiniz çok çeşitli iş yükleri vardır. Makul tasarım kararları almak için depolama sistemi tasarımcısı hangi iş yüklerinin önemli olduğuna nasıl karar vermelidir?

AFS tasarımcıları, dosya sistemlerinin nasıl kullanıldığını ölçme konusundaki deneyimlerini göz önünde bulundurarak, belirli iş yükü varsayımlarında bulundular; özellikle, çoğu dosyanın sıklıkla paylaşılmadığını ve bütünlüklerine sırayla erişildiğini varsaydılar. Bu varsayımlar göz önüne alındığında, AFS tasarımı mükemmel bir anlam ifade ediyor.

Ancak, bu varsayımlar her zaman doğru değildir. Örneğin, bir günlüğe periyodik olarak bilgi ekleyen bir uygulama hayal edin. Mevcut büyük bir dosyaya küçük miktarlarda veri ekleyen bu küçük günlük yazma işlemleri, AFS için oldukça sorunludur. Bir işlem veri tabanındaki rastgele güncellemeler gibi başka birçok zor iş yükü de mevcuttur.

Hangi tür iş yüklerinin yaygın olduğu hakkında bilgi edinebileceğiniz bir yer, gerçekleştirilen çeşitli araştırma çalışmalarıdır. AFS retrospektifi [H+88] dahil olmak üzere iş yükü analizinin [B+91, H+11, R+00, V99] iyi örnekleri için bu çalışmalardan herhangi birine bakın.

şekil etkisi. Blok tabanlı bir protokol olarak NFS, okuma veya yazma boyutuyla orantılı G/Ç gerçekleştirir.

Genel olarak, NFS ve AFS'nin farklı varsayımlarda bulunduğunu ve sonuç olarak şaşırtıcı bir şekilde farklı performans sonuçları elde ettiğini görüyoruz. Bu farklılıkların önemli olup olmadığı, her zaman olduğu gibi, bir iş yükü sorunudur.

- AFS: Diğer İyileştirmeler

Berkeley FFS'nin (sembolik bağlantılar ve bir dizi başka özellik ekleyen) piyasaya sürülmesiyle gördüğümüz gibi, AFS tasarımcıları sistemlerini oluştururken sistemin kullanımını ve yönetimini kolaylaştıran bir dizi özellik ekleme fırsatını yakaladılar. . Örneğin AFS, istemcilere gerçek bir genel ad alanı

sağlayarak, tüm dosyaların tüm istemci makinelerde aynı şekilde adlandırılmasını sağlar. NFS, aksine, her istemcinin NFS sunucularını istedikleri şekilde bağlamasına izin verir ve bu nedenle, dosyalar istemciler arasında yalnızca geleneksel (ve büyük bir yönetimsel çabayla) benzer şekilde adlandırılabilir.

AFS ayrıca güvenliği ciddiye alır ve kullanıcıların kimliğini doğrulamak ve bir kullanıcı isterse bir dizi dosyanın gizli tutulabilmesini sağlamak için mekanizmalar içerir. NFS, aksine, yıllarca güvenlik için oldukça ilkel bir desteğe sahipti.

AFS, esnek kullanıcı tarafından yönetilen erişim kontrolü için olanaklar da içerir. Bu nedenle, AFS kullanırken, bir kullanıcı tam olarak kimin üzerinde büyük bir kontrole sahiptir.

hangi dosyalara erişebilir. NFS, çoğu UNIX dosya sistemi gibi, bu tür paylaşım için çok daha az desteğe sahiptir.

Son olarak, daha önce bahsedildiği gibi, AFS sistem yöneticileri için sunucuların daha basit yönetimini sağlayan araçlar ekler. Sistem yönetimi hakkında düşünürken AFS, alanın ışık yılı ötesindeydi.

- Özet

AFS, dağıtılmış dosya sistemlerinin NFS'de gördüğümüzden oldukça farklı bir şekilde nasıl oluşturulabileceğini bize gösteriyor. AFS'nin protokol tasarımı özellikle önemlidir; sunucu etkileşimlerini en aza indirerek (tüm dosya önbelleğe alma ve geri aramalar yoluyla), her sunucu birçok istemciyi destekleyebilir ve böylece belirli bir siteyi yönetmek için gereken sunucu sayısını azaltabilir. Tek ad alanı, güvenlik ve erişim kontrol listeleri dahil olmak üzere diğer birçok özellik, AFS'nin kullanımını oldukça keyifli hale getirir. AFS tarafından sağlanan tutarlılık modelinin anlaşılması ve akıl yürütmesi basittir ve NFS'de bazen gözlemlenen gibi ara sıra garip davranışlara yol açmaz.

Belki de ne yazık ki, AFS muhtemelen düşüşte. NFS açık bir standart haline geldiğinden, birçok farklı satıcı onu destekledi ve CIFS (Windows tabanlı dağıtılmış dosya sistemi protokolü) ile birlikte NFS pazara hakim durumda. Zaman zaman AFS kurulumları görülse de (Wisconsin dahil çeşitli eğitim kurumlarında olduğu gibi), tek kalıcı etki muhtemelen gerçek sistemin kendisinden ziyade AFS'nin fikirlerinden olacaktır. Aslında, NFSv4 artık sunucu durumunu (ör. "açık" bir protokol mesajı) ekler ve bu nedenle temel AFS protokolüne giderek artan bir benzerlik taşır.

Referanslar

[B+91] "Measurements of a Distributed File System" by Mary Baker, John Hartman, Martin Kupfer, Ken Shirriff, John Ousterhout. SOSP '91, Pacific Grove, California, October 1991. *An early paper measuring how people use distributed file systems. Matches much of the intuition found in AFS.*

[H+11] "A File is Not a File: Understanding the I/O Behavior of Apple Desktop Applications" by Tyler Harter, Chris Dragga, Michael Vaughn, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci- Dusseau. SOSP '11, New York, New York, October 2011. *Our own paper studying the behavior of Apple Desktop workloads; turns out they are a bit different than many of the server-based workloads the systems research community usually focuses upon. Also a good recent reference which points to a lot of related work.*

[H+88] "Scale and Performance in a Distributed File System" by John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, Michael J. West. ACM Transactions on Computing Systems (ACM TOCS), Volume 6:1, Febru- ary 1988. *The long journal version of the famous AFS system, still in use in a number of places throughout the world, and also probably the earliest clear thinking on how to build distributed file systems. A wonderful combination of the science of measurement and principled engineering.*

[R+00] "A Comparison of File System Workloads" by Drew Roselli, Jacob R. Lorch, Thomas E. Anderson. USENIX '00, San Diego, California, June 2000. *A more recent set of traces as comparedto the Baker paper [B+91], with some interesting twists.*

[S+85] "The ITC Distributed File System: Principles and Design" by M. Satyanarayanan, J.H. Howard, D.A. Nichols, R.N. Sidebotham, A. Spector, M.J. West. SOSP '85, Orcas Island, Wash- ington, December 1985. *The older paper about a distributed file system. Much of the basic design of AFS is in place in this older system, but not the improvements for scale. The name change to "Andrew" is an homage to two people both named Andrew, Andrew Carnegie and Andrew Mellon. These two rich dudes started the Carnegie Institute of Technology and the Mellon Institute of Industrial Research, respectively, which eventually merged to become what is now known as Carnegie Mellon University.*

[V99] "File system usage in Windows NT 4.0" by Werner Vogels. SOSP '99, Kiawah Island Resort, South Carolina, December 1999. *A cool study of Windows workloads, which are inherently different than many of the* UNIX*-based studies that had previously been done.*

Ödev (Simülasyon)

Bu bölüm, Andrew Dosya Sisteminin nasıl çalıştığına dair bilginizi pekiştirmek için kullanabileceğiniz basit bir AFS simülatörü olan afs.py'yi tanıtmaktadır. Daha fazla ayrıntı için BENİOKU dosyasını okuyun.

Sorular

- İstemciler tarafından hangi değerlerin okunacağını tahmin edebildiğinizden emin olmak için birkaç basit durum çalıştırın. Rastgele çekirdek bayrağını (-s) değiştirin ve hem ara değerleri hem de dosyalarda saklanan nihai değerleri takip edip tahmin edip edemeyeceğinize bakın. Ayrıca dosya sayısını (-f), istemci sayısını (-C) ve okuma oranını (-r, 0 ile 1 arasında) biraz daha zorlaştırmak için değiştirin. Daha ilginç etkileşimler için biraz daha uzun izler de oluşturmak isteyebilirsiniz, örneğin (-n 2 veya daha yüksek).
- Bu sorunu çözmek için, istemciler tarafından okunacak değerleri doğru bir şekilde tahmin edebileceğinizi doğrulamak için birkaç test senaryosu çalıştırmalısınız. Görevi daha zor hale getirmek için rastgele çekirdek bayrağını (-s) değiştirmeli ve dosyalarda saklanan ara değerler ile nihai değerleri izlemeye çalışmalısınız. Ayrıca dosya sayısını (-f), istemci sayısını (-C) ve okuma oranını (-r) 0 ile 1 arasında değiştirmelisiniz. Ek olarak, (-n) bayrağını ayarlayarak daha uzun izlemeler oluşturabilirsiniz. 2 veya daha yüksek. Bu, farklı değişkenler arasındaki daha ilginç etkileşimleri görmenizi sağlayacaktır.
- Şimdi aynı şeyi yapın ve AFS sunucusunun başlattığı her geri aramayı tahmin edip edemeyeceğinize bakın. Farklı rastgele çekirdekleri deneyin ve programın yanıtları sizin için hesaplamasını sağladığınızda (-c ile) geri aramaların ne zaman gerçekleştiğini görmek için yüksek düzeyde ayrıntılı geri bildirim (örneğin, -d 3) kullandığınızdan emin olun. Her geri aramanın tam olarak ne zaman gerçekleştiğini tahmin edebilir misiniz? Birinin gerçekleşmesi için kesin koşul nedir?
- AFS sunucusu tarafından başlatılan geri aramaları tahmin etmek için, programı farklı rasgele tohumlarla çalıştırmalı ve geri aramaların ne zaman gerçekleştiğini görmek için yüksek düzeyde ayrıntılı geri bildirim (örn. -d 3) kullanmalısınız. Programın yanıtları sizin için hesaplamasını sağlamak için (-c) bayrağını da kullanabilirsiniz; bu, her geri aramanın tam olarak ne zaman gerçekleştiğini görmenizi sağlar.Bir geri aramanın kesin koşullarını belirlemek için, AFS sunucusunu kontrol eden kodu inceleyebilir ve bir geri aramayı tetikleyecek koşulları arayabilirsiniz.
- Yukarıdakine benzer şekilde, bazı farklı rasgele tohumlarla çalıştırın ve her adımda tam önbellek durumunu tahmin edip edemeyeceğinize bakın. Önbellek durumu -c ve -d 7 ile çalıştırılarak gözlemlenebilir.
- Her adımda tam önbellek durumunu tahmin etmek için, programı farklı rasgele tohumlarla çalıştırabilir ve önbellek durumunu gözlemlemek için (-c) ve (-d 7) işaretlerini kullanabilirsiniz. (-c) bayrağı, programın yanıtları sizin için hesaplamasını sağlar ve (-d 7) bayrağı, her adımda önbellek durumu hakkında bilgiler de dahil olmak üzere yüksek düzeyde ayrıntılı geri bildirim sağlar.Daha sonra, her adımda tam önbellek durumunu görmek için çıktıyı inceleyebilir ve girişe ve kodda belirtilen koşullara dayalı olarak sonraki adımlarda durumu tahmin etmeye çalışabilirsiniz.
- Şimdi bazı özel iş yükleri oluşturalım. Simülasyonu -A oa1:w1:c1,oa1:r1:c1 bayrağıyla çalıştırın. İstemci 1 tarafından rastgele programlayıcı ile çalışırken a dosyasını okuduğunda gözlemlenen farklı olası değerler nelerdir? (farklı sonuçları görmek için farklı rastgele tohumları deneyin)? İki müşterinin işlemlerinin tüm olası program serpiştirmelerinden kaç tanesi müşteri 1'in 1 değerini okumasına ve kaç tanesinin 0 değerini okumasına yol açar?
- Simülasyonu -A oa1:w1:c1,oa1:r1:c1 bayrağıyla çalıştırdığınızda, istemci 1, "a" dosyasına 1 değeriyle bir yazma işlemi gerçekleştirecek ve ardından hemen "a" dosyasına bir okuma işlemi gerçekleştirecektir. . İstemci 1'in "a" dosyasını okuduğunda gözlemleyebileceği olası değerler, iki istemcinin işlemlerinin belirli çizelge serpiştirmelerine bağlı olacaktır.Farklı sonuçlar görmek için simülasyonu farklı rastgele tohumlarla çalıştırmayı deneyebilirsiniz. Bu, farklı program serpiştirmeleriyle sonuçlanacak ve müşteri 1'in "a" dosyasından farklı değerler okumasına neden olabilir.Müşteri 1'in 1 değerini okumasına yol açan program serpiştirmelerinin sayısını ve müşteri 1'in 0 değerini okumasına yol açan sayıyı belirlemek için, simülasyonu farklı rasgele tohumlarla çalıştırabilir ve her değerin müşteri tarafından gözlemlenme sayısını sayabilirsiniz. 1. Ayrıca, olası program serpiştirmeleri hakkında akıl yürütmeye çalışabilir ve bunlardan kaçının her bir değerin müşteri 1 tarafından okunmasıyla sonuçlanacağını belirleyebilirsiniz.
- Şimdi bazı özel programlar oluşturalım. -A oa1:w1:c1,oa1:r1:c1 bayrağıyla çalışırken, ayrıca şu programlarla çalıştırın: -S 01, -S 100011, -S 011100 ve aklınıza gelen diğerleri. İstemci 1 hangi değeri okuyacak?
- Simülasyonu -A oa1:w1:c1,oa1:r1:c1 bayrağı ve -S 01 programı ile çalıştırdığınızda, müşteri 1 "a" dosyasına 1 değeriyle bir yazma işlemi yapacak ve ardından müşteri 2 gerçekleştirecek "a" dosyasında bir okuma işlemi. İstemci 1 sonraki okuma işlemini gerçekleştirdiğinde, istemci 2

tarafından yazılan 0 olan değeri gözlemleyecektir.Benzer şekilde, simülasyonu -S 100011 programıyla çalıştırdığınızda, istemci 1 "a" dosyasına 1 değeriyle bir yazma işlemi yapacak ve ardından istemci 2 "a" dosyasına bir okuma işlemi gerçekleştirecektir. İstemci 1 daha sonra "a" dosyasına 0 değeriyle ikinci bir yazma işlemi gerçekleştirecek ve ardından istemci 2 tarafından başka bir okuma işlemi gerçekleştirilecektir. İstemci 1 son okuma işlemini gerçekleştirdiğinde, istemci 1 tarafından yazılan değer olan 0'ı gözlemleyecektir.-S 011100 programı için, istemci 1, "a" dosyasına 1 değeriyle bir yazma işlemi gerçekleştirecek ve ardından istemci 2 tarafından bir okuma işlemi gerçekleştirecektir. Daha sonra İstemci 1, "a" dosyasına 0 değeriyle ikinci bir yazma işlemi gerçekleştirecektir, ardından müşteri 2 tarafından başka bir okuma işlemi yapılır. İstemci 1 son okuma işlemini gerçekleştirdiğinde, müşteri 1 tarafından yazılan 0 olan değeri gözlemleyecektir.Genel olarak, istemci 1'in okuyacağı değer, -S bayrağı tarafından belirtilen özel işlem planına bağlı olacaktır. Müşteri 1 tarafından okunan değerin nasıl değiştiğini görmek için simülasyonu farklı çizelgelerle çalıştırmayı deneyebilirsiniz.

- Şimdi şu iş yüküyle çalıştırın: -A oa1:w1:c1,oa1:w1:c1 ve programları yukarıdaki gibi değiştirin. -S 011100 ile çalıştırdığınızda ne olur? Peki ya -S 010011 ile çalıştırdığınızda? Dosyanın nihai değerini belirlemede önemli olan nedir?
- Simülasyonu -A oa1:w1:c1,oa1:w1:c1 iş yükü ve -S 011100 programıyla çalıştırdığınızda, istemci 1, "a" dosyasında 1 değeriyle bir yazma işlemi ve ardından bir okuma işlemi gerçekleştirir: istemci 2. İstemci 1 daha sonra "a" dosyasına 0 değeriyle ikinci bir yazma işlemi gerçekleştirecek ve ardından istemci 2 tarafından başka bir okuma işlemi gerçekleştirilecektir.Simülasyonu -S 010011 programıyla çalıştırdığınızda, istemci 1, "a" dosyasına 1 değeriyle bir yazma işlemi gerçekleştirecek ve ardından istemci 2 tarafından bir okuma işlemi gerçekleştirecektir. Ardından, İstemci 1, "a" dosyasına ikinci bir yazma işlemi gerçekleştirecektir. " 0 değeriyle, ardından müşteri 1 tarafından başka bir okuma işlemi yapılır.Her iki durumda da, dosyanın nihai değeri, müşteri 1 tarafından gerçekleştirilen son yazma işlemi tarafından belirlenir. -S bayrağıyla belirtilen özel işlem programı, istemci işlemlerinin yürütülme sırasını belirler, ancak son değer dosyanın değeri her zaman son yazma işlemi tarafından yazılan değer olacaktır.Dosyanın nihai değerinin, diğer istemciler tarafından yazılan değerler veya işlemler gerçekleştirilirken önbelleğin durumu gibi diğer faktörlerden de etkilenebileceğini unutmamak önemlidir.