Güncel Dağıtık Dosya Sistemlerinin Karşılaştırmalı Analizi

Uğurcan Ergün, Süleyman Eken ve Ahmet Sayar
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Umuttepe Kampüsü, 41380, Kocaeli-Türkiye
ugurcanergn@gmail.com, {suleyman.eken & ahmet.sayar}@kocaeli.edu.tr

***Öz*– Dağıtık Dosya Sistemleri (DDS) günümüzde disklerin ve depolama kaynaklarının ortaklaşa kullanımını sağlayıp dağıtık sistemler aracılığıyla büyük ölçekte hesaplamalar ve işlemler yapılmasına olanak sağlamaktadırlar. DDS'nin tasarımında değişik mimariler ve sınıflandırmalar mevcuttur. Bu çalışmada DDS'nin tasarımına dair bir inceleme yapılmış ve ardından günümüzde yaygın olarak kullanılan DDS'lerden Sun Ağ Dosya Sistem (NFS), Andrew Dosya Sistemi (AFS) ve Google Dosya Sistemi (GFS) tasarım hedefleri, isimlendirme ve lokasyon mekanizmaları, replikasyon ve önbellekleme kullanımı, güvenlik ve sistem yönetim destekleri bakımından benzer ve farklı yönleri ele alınmıştır.**

Anahtar Kelimeler – Dağıtık dosya sistemleri, Bilgi erişimi, Dağıtık sistem, Hataya dayanıklılık, İşletim sistemleri.

I. GİRİŞ

Bilgisayarlarda bütün uygulamaların veri depolaması ve gerektiğinde bu veriye tekrar erişebilmeleri gerekmektedir. Süreçler çalıştıkları müddetçe kısıtlı miktarda veriyi kendi adres alanlarında saklayabilirler. Ancak bu alan bazı uygulamalar için yetmeyecektir. Ayrıca veri süreç sonlanınca kaybolacaktır. Bazı durumlarda verilerin günlerce, aylarca, yıllarca saklanması gerekir. Verilerin kalıcı bir şekilde sürekli depolanması bilgisayarlarda kullanılan en temel soyutlamalardan biridir. Kalıcı depolama ortamlarındaki temel problemleri (verinin bulunması, kullanıcıların birbirlerinin verilerini değiştirmesinin önlenmesi, boş alanların tespiti) çözmek için dosya adı verilen yapı kullanılır.

Bir dosya süreçler tarafında üretilen verilerin saklandığı mantıksal birimlerdir. Dosyalar hem verinin kendisini hem de veriyle ilişkili diğer bilgileri (meta-veri) içerir. Bu meta veriler genelde dosya ismi, dosya boyutu, sahiplik bilgisi, zaman damgası, erişim kontrol listesi gibi kayıtlardır. Dosyalar genelde disklerde veya diğer uçucu olmayan ortamlarda depolanırlar.

Dosya sistemleri; dosyaların bulunması, depolanması, isimlendirilmesi, korunması, organizasyonu gibi işlerden sorumlu yazılımlardır. Ayrıca dosya sistemleri programcıları dosyalar için yer tahsisi, depolama düzeni gibi sorunlarla uğraşmaktan kurtarmak için bir programlama arayüzü sunarlar.

Dosya sistemleri, işletim sistemleri tasarımı açısından her biri bir alttakini kapsayacak şekilde dört seviyeye ayrılabilir: (i) Tek kullanıcılı, tek süreç işleten sistemler (IBM PC-DOS [1] ve Apple Macintosh [2]), (ii) Tek kullanıcılı, çoklu süreç işleten sistemler (OS/2 [3]), (iii) Çok kullanıcılı, çoklu süreç işleten klasik zaman paylaşımlı sistemler (UNIX [4]), (iv) Çoklu bilgisayarlarda çok kullanıcılı ve çoklu süreç işleten sistemler (dağıtık dosya sistemleri). Bu seviyelere ait dosya sistemleri gereksinimleri yukarıya çıktıkça artacaktır. İlk seviyede oldukça basit bir dosya sistemi yeterliyken ikinci seviyede tutarlılık kontrolüne ve üçüncü seviyede kullanıcılar arası güvenliğe, dördüncü seviyede zaman paylaşımlı dosya sistemi soyutlamasının verimli, güvenli ve sağlam bir şekilde yerine getirilmesine ihtiyaç olacaktır [5].

Sınıflandırmanın en üst seviyesinde yer alan DDS için temel amaç farklı bilgisayarların dosyaları ve depolama kaynaklarını ortak kullanabilmelerini sağlamaktır. Bu amaç klasik zaman paylaşımlı sistemlerin dosya sistemlerine benzer bir şekilde gerçeklenmeye çalışılmakta ve genelde UNIX dosya sistemi temel alınmaktadır.

DDS'lerde dosya sistemi istemcilere dosya servisleri sunar. İstemciler istemci arayüzlerini kullanarak belli işlemleri (dosya oluşturma, silme, okuma, yazma vs.) gerçekleştirebilirler. Bu dosyalar sunucunun diskinde bulundurulur. Bir dağıtık dosya sistemi bir dağıtık işletim sisteminin parçası olabileceği gibi bir klasik işletim sisteminde işletim sistemi ve dosya sistemi içinde ara katman görevi görebilir [6].

Bu çalışmada dağıtık sistemler için veri paylaşılması işini yapan dosya sistemleri ele alınmıştır. Giriş bölümünde, dosya sistemlerine dair temel kavramlar incelenmiş; ikinci bölümde, DDS servis kalitesi gereksinimlerine, üçüncü bölümde DDS mimarilerine göz atılmış ve temel bir çerçeve çizilmeye çalışılmıştır. Dördüncü bölümde, günümüzde kullanılan modern dosya sistemleri belli açılardan incelenmiş ve değerlendirilmişlerdir. Son bölümde ise karşılaştırma ve sonuçlar açıklanmıştır.

II. DDS SERVİS KALİTESİ GEREKSİNİMLERİ

A. Şeffaflık

Kendini kullanıcılara ve uygulamalara tek bilgisayar gibi sunabilen dağıtık sistemler şeffaf dağıtık sistemler olarak bilinirler. Şeffaflık dağıtık sistem tasarımında önemli amaçlardan biri olmasına rağmen sistemin tasarımında şeffaflığın mal olacağı performansı ve arttıracağı karmaşıklığı göz önüne almak gerekir [7]. Yedi çeşit şeffaflık türü tanımlamıştır. Ancak bunlardan sadece beşi DDS'nin alanına girmektedir:

Erişim şeffaflığı: Yerel ve uzak dosyalara erişim için tek bir komut seti bulunmalı ve yerel dosyalarla çalışmak için

yazılmış programlar uzak dosyalarla değişiklik yapmadan çalışabilmelidirler.

Konum şeffaflığı: Kullanıcı tekil bir isim alanı görmeli ve dosyalar yolları değiştirilmeden tekrar konumlandırılabilmelidir.

Mobilite şeffaflığı: Dosyalar kullanıcıları etkilemeden taşınabilmedir.

Performansta şeffaflık: Kullanıcı programları sistemde yük altındayken bile performansla çalışmalıdır.

Ölçeklenebilirlikte şeffaflık: Servis farklı ağ boyutları, yük dağılımları ile başa çıkabilmek için genişletilebilir olmalıdır [8].

B. Eşzamanlı Dosya Güncelleme

Bir dosyada bir istemci tarafından yapılan değişiklikler diğer istemcilerin çalışmalarını ve dosyada değişiklik yapmalarını engellememelidir. Paylaşılan veriler eşzamanlı dosya güncelleme çözümleri maliyetli olduğu için dosya servisleri genelde UNIX tarzı kilitleme sistemleri kullanır.

Dosya sistemlerinde bir dosyayı birden fazla süreç tarafından işlenebileceği için bu gibi durumlarda neler olacağını yani senkronizasyon semantiği tanımlamak gerekir. Temelde dosya sistemlerinde dört farklı semantik kullanılır:

UNIX semantiği: Önce yazma, sonra okuma. Bütün değişiklikler bütün süreçlerde anlık olarak görülebilir

Oturum semantiği: Yapılan değişiklik dosya kapatılana kadar sadece değişikliği yapan sürece gözükür.

Salt okunur dosya semantiği: Dosyalara yazma işlemi yapılamaz.

Atomik işlem semantiği: Yazma isteği birleştirilir ve tek bir işlem halinde yollanır bu şekilde işlemin bir bütün olarak uygulandığından emin olunur.

C. Dosya Replikasyonu

Dosyaların birden fazla kopya halinde DDS'lerde saklanmasının iki temel avantajı vardır: güvenilirlik ve performans. Eğer veri sunucular arasında kopyalanmış olursa bir kopyaya erişilememesi veya bir kopyanın bozulması durumlarında o veriler üzerinde çalışmaya devam edilebilir. Ayrıca verinin replikasyon yoluyla dağıtılması sunucular arasında yük dengelenmesini sağlayacak ve performansı arttıracaktır. İlk DDS'nin pek çoğunda sunucu tarafında replikasyon desteği bulunmazken modern DDS'nin çoğunluğu bu desteği sağlamaktadır.

D. Tutarlılık

Replikasyon güvenilirlik ve performans gibi belirli avantajlara sahip olmasına rağmen ortaya yeni bir sorun çıkarmaktadır: tutarlılık. Eğer kopyalardan biri değiştirilirse diğer kopyalardan farklı bir hale gelecektir ve bu durumda kopyaların tutarlılığının sağlanması gerekecektir. Standart UNIX sistemlerde bu sorun one-copy-update yöntemi ile çözülmeye çalışılır. Bir dosyada yapılan değişiklik anlık olarak diğer süreçlere yansıyacaktır. Ancak bu yöntem dağıtık sistemlere pek uygun olmadığından bu ilkeden sapılması gerekir.

DDS'i genelde performansı arttırmak için önbellekleme yaparlar. Önbellekleme performansı arttırırken önbelleğe alınmış dosyaların sunuculardaki dosyalarla aynı dosya olduğundan emin olunması gerekir ve bu ortaya bir tutarlılık sorunu çıkartır. Eğer bir dosya bir istemci de önbelleklendikten sonra sunucuda değişirse önbelleklenmiş kopya eskimiştir ve kullanımı sağlıklı olmayacaktır. Bu duruma karşı bazı önlemler alınmalıdır.

E. Platform Bağımsızlık

Dosya servis arayüzleri, sunucu ve istemci yazılımlarının farklı işletim sistemlerinde ve farklı donanımlarda çalışacak bir şekilde tasarlanmalıdır.

F. Hata Toleransı

Dağıtık sistemleri, merkezi sistemlerden ayıran en önemli kavramlardan biri kısmi hata kavramıdır. Merkezi sistemlerin aksine sistemin bir parçasında bir hata oluştuğunda sistemin geri kalanının etkilenmeme olasılığı vardır. Dağıtık sistemlerdeki önemli amaçlardan bir tanesi de sistemin hatalardan performansı baltalamadan kurtarılmasıdır. Özellikle bir hatayla karşılaşıldığında sistem gerekli onarımlar yapılana kadar çalışabilmeli yani sistem hataları tolere edebilmeli ve varlıklarında dahi çalışabilmelidir [9]. Dosya sunucu semantiği dosya sistemlerinde iki türlüdür: durumsuz servisler ve durumlu servisler.

Durumsuz servisler: Dosyaların durumu hakkında herhangi bir bilgi tutulmaz. Bütün istekler müstakil olarak yapılır. Dosya açma veya kapama gibi işlemler kullanılmaz. İstemci her istekte bulunduğunda dosya ile ilgili gerekli bilgileri ve dosya içindeki konumu bildirmelidir. Hata toleransı yüksektir. Kilitleme ve senkronizasyon desteği zayıftır ve ayrıca performansı iyi değildir. Çünkü her bağlantıda daha fazla veri gönderilmekte ve işlenmektedir.

Durumlu servisler: Dosyaların durumlarıyla ilgili bilgiler tutulur. Sistem üzerinde açılan dosyalara dair kayıtlar tutulur. Sunucu biten işlemlerle ilgili bilgileri temizler. Olası bir istemci hatalarından sunucu haberdar olmalıdır. Gönderilen ve işlenen veri sayısı düştüğü için performans daha iyidir.

G. Güvenlik

Bütün dosya sistemlerinde erişim kontrolü erişim kontrol listeleri yoluyla yapılır. Dağıtık dosya sistemlerinde istemcinin kimliğinin doğrulanması sistemin korunması için önemli olduğundan yetkilendirme dijital imza ve bazen kriptolama kullanılarak yapılır.

H. Verimlilik

Dağıtık dosya sistemlerinin en az klasik dosya sistemleri kadar performanslı ve güvenilir olmaları gerekmektedir. Aynı zamanda yönetimlerinin pratik olmaları ve yöneticilere işlemlerde kolaylık sağlamalı gerekli yönetim araçlarını sağlamalıdırlar [8].

III. DDS MİMARİ YAKLAŞIMLARI

İstemci-sunucu mimarisinde sistemden sorumlu bir veya birden fazla sunucu vardır. Her sunucu kendi yerel dosya sistemine dair standartlaşmış bir görüntü sunar. İstemci sunucu iletişimi genelde Uzak Metot Çağrımı (UMÇ) ile sağlanır. Dosyalara erişmek ve dosyaları aktarmak için iki tane model vardır: uzaktan erişim modeli ve yükleme/indirme modeli.

Uzaktan erişim modelinde sunucunun kullanıcıya dosya işlemleri için yerel dosya sistemine benzer bir arayüz sağlaması gerekir. Yükleme/indirme modelinde istemci sunucudan dosyaları indirir kendi yerelinde işlemlerini, değişikliklerini yapar. Eğer bir değişiklik yapıldıysa sunucuya dosyayı tekrardan geri yükler.

Klasik sunucu istemci mimarisi biraz geliştirilerek sunucu kümelerine de uygulanabilir (küme-tabanlı istemci-sunucu mimarisi). Sunucu kümeleri genelde paralel uygulamalar için kullanıldıkları için dosya sistemleri de buna göre tasarlanır. En çok bilinen tekniklerden biri de dosya ayırma tekniğidir. Bu teknikte dosyaların kendileri de parçalara bölünür ve sistem boyunca dağıtılır.

Küme-tabanlı mimari sadece paralel dosya erişiminin makul olduğu uygulamalarda verimli çalışabilir ki genelde verinin matrisler gibi çok düzenli yapılarda saklanmasını gerektirir. Aksi takdirde dosyalar klasik olarak bütün halinde saklanabilir.

Simetrik mimaride ise diğer mimarilerde olduğu gibi merkezi kontrol yapan sunucular yoktur. Ayrıca bütün düğümler fonksiyonel olarak simetriktirler. Bu mimari aynı zamanda Eşten Eşe Bağlantı (P2P) dosya sistemi olarak da anılır. Bu tarz sistemler genelde Dağıtık Hash Tablosu (DHT) kullanılarak gerçeklenirler. DHT'ler normal hash tabloları gibi anahtar değer çiftleri içerir ve düğümler anahtar alanı dağıtılır, bütün düğümler komşu düğümlerinin listesini bir yönlendirme tablosunda tutar. DHT'nin gerçeklendiği dört temel protokolü vardır: Chord, CAN, Tapestry, Pastry. Bunlardan en popülerlerinden olan Chord DDS gerçeklemelerinde kullanılmaktadır [10].

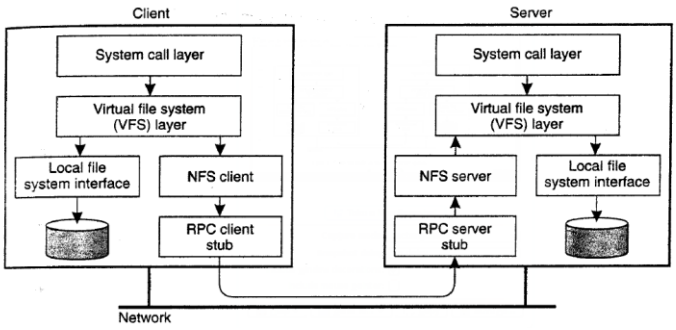
IV. Örnek Dosya Sistemleri

Bu bölümde yaygın olarak kullanılan üç dağıtık dosya sisteminin; ikinci bölümde anlatılan çeşitli servis kalitelerine ve üçüncü bölümde anlatılan mimari yaklaşımlara göre tasarım hedefleri, isimlendirme ve lokasyon mekanizmaları, replikasyon ve önbellekleme kullanımı, güvenlik ve sistem yönetim destekleri bakımından benzer ve farklı yönleri ele alındı.

A. Sun Ağ Dosya Sistemi (NFS)

NFS, Sun Microsystems tarafından ilk olarak 1984 yılında [11] geliştirilmeye başlanmış olup hem bir dağıtık dosya sistemi protokolünün hem de dosya sisteminin adıdır. 1989 yılında ikinci sürümü (RFC 1094), 1995 üçüncü sürümü (RFC 1813) ve 2000 yılında dördüncü sürümü çıkmıştır (RFC 3010, 3530). Açık bir standart olarak tasarlanmıştır. UNIX sistemlerde en çok kullanılan ve gerçeklenmiş DDS'lerden biridir. UMÇ iletişim modelini kullanılır. Bu çalışmada daha çok NFSv3 ve NFSv4 sürümlerinden bahsedilecektir.

Şekil. 1 NFS genel mimarisi [9]



NFS'te genel tasarım amacı klasik UNIX dosya sisteminin dağıtık bir şekilde gerçeklenebilmesidir. Standart UNIX dosya arayüzü Sanal Dosya Sistemi (SDS) ile değiştirilmiştir. SDS programcıların farklı dosya sistemleriyle zahmetsizce çalışabilmesi için işletim sistemi çağrıları ve dosya sistemleri arasında bir ara katman görevi görür. İletişim için UMÇ kullanılır. NFSv3'te her komut için ayrı UMÇ yapılırken NFSv4'te bir UMÇ'de birden fazla komut gönderilebilir (Şekil 1'e bakınız).

İsimlendirme ve Şeffaflık

NFS isimlendirme modelinin temellerinden biri kullanıcıların dosyalara şeffaf bir şekilde erişmesini sağlamaktadır. Bunu istemcilere uzak bir dosya sistemini kendi dosya sistemlerine bağlama imkânı vererek sağlanmaya çalışılır. NFS'te dosya sistemlerinin tamamı değil sadece belli kısımları da bağlanabilir [8]. Bağlama işleminin kendisi bağlanacak sunucunun konumunun bilinmesini gerektirdiği için şeffaflık ilkesini sağlamamaktadır; ancak bağlama işlemi tamamlandıksan sonraki erişimler şeffaf bir şekilde yapılmaktadır. Bağlama işlemi aşağıdaki komut ile yapılır:

"mount -t nfs4 -o proto=tcp,port=2049 nfs-server:/users /home/users"

Bir dizinin istemcilerin erişimine açılabilmesi için sunucu bu dizini erişime açtığına dair bildirimde bulunmalıdır. Bu bildirim genelde UNIX türevi sistemlerde /etc/exports dosyasına bu dizin ve gerekli parametreler eklenerek yapılır.

Dosya ismi çözümlemeleri NFSv3 ve önceki versiyonlarda iteratif olarak yapılırdı. Bu demektir ki /home/ugurcan/proje.pdf dosyasına ulaşılacaksa üç farklı arama (lookup) çağrısı yani üç tane UMÇ isteği gönderilmesi gerekmektedir. Bunun sebebi global bir isim alanı bulunmaması ve dosya adı doğrudan çözülmüş olsaydı her bir istemcinin kendi isim alanına göre göndereceği dosya yollarının geçersiz olabilmesidir. NFSv4 ise NFSv3'den farklı olarak özyinelemeli çağrı yapabilmektedir.

Dosya belirteçleri dosya sistemlerinde dosyalara karşılık gelen tekil referanslardır. Dosyayla beraber oluşturulurlar, dosyaların isimlerinden bağımsızlardır ve dosya silindiğinde tekrar kullanılamazlar. İstemciler belirteçlerin içeriğine ulaşamazlar. Dosya belirteçleri NFSv2'de 32 bit, v3'te 64 bit ve v4'te 128 bit ile temsil edilirler [9].

İstemciler bir dosya sistemini veya dizini kendilerine bağladıklarında onlara bağlandıkları kök dizinin dosya belirteci gönderilir. Dosya işlemlerinin çoğu belirteçlerle yapılır. İstemci bir dosya işlemi gerçekleştirmeden önce dosya isminden belirteci sorgulamalıdır.

Dosya sistemlerinin elle bağlanması çok pratik olmadığından otomatik bağlama yöntemi geliştirilmiştir. Erişilmeye çalışan ve yerel dosya sisteminde bulunmayan bir dizin eğer NFS lookup çağrısı ile bulunabiliyorsa bağlanacaktır. NFS otomatik bağlayıcısı modern Linux sistemlerde "autofs" adı ile kullanılır.

Senkronizasyon

DDS'lerde daha çok oturum semantiği metodu kullanılmasına rağmen NFS'te herhangi bir senkronizasyon metodu bulunmamaktadır. Bu sorunu çözebilmek için NFS'te ayrı bir dosya kilitleme sistemi kullanılır. NFSv3 kilit sisteminde üç farklı kilit değişkeni bulunur: (i) Lock: Belirli bir bayt dizisini kilitler. (ii) Lockt: Belirli bir bayt dizisinin üzerinde kilit olup olmadığını kontrol eder. (iii) Locku: Belirli bir bayt dizisi üzerindeki kilidi kaldırır.

Tutarlılık ve Replikasyon

İstemci önbellekleme için NFSv3 protokolünde bir tanımlama yapılmamış ve bu gerçeklemeye bırakılmıştır. Sun firmasının gerçeklemesinde bir dosyanın önbelleklenmiş kopyasının zaman mührü ile ve sunucudaki kopyanın zaman mührü karşılaştırılır. Eğer mühürler farklıysa dosya değişmiş demektir ve önbellekteki kopya çöpe atılır. Ayrıca önbellekteki dosyalar 3-30 saniye aralığında, dizinler 30-60 saniye aralığında çöpe atılır [8].

NFS'te sunucu tarafında önbellekleme yazma işlemlerinde yapılır. Önbellekteki değişiklik diske yazılmadan kullanıcıya yazma işlemi onayı gönderilmez. NFS sunucularında dosya replikasyonu yapılmamaktadır.

Hata Toleransı

NFSv3'te dosyaların durumu hakkında herhangi bir bilgi tutulmaz. Bütün istekler müstakil olarak yapılır. Dosya açma veya kapama gibi işlemler kullanılmaz. Hata toleransı yüksektir. NFSv4'te ise dosyaların durumlarıyla ilgili bilgiler tutulur ve olası bir istemci hatasına karşın sunucu haberdardır.

Güvenlik

NFS'te yetkilendirme sunucu tarafında yapılır. Güvenlik için güvenli UMÇ kullanılmasının yanı sıra standart UNIX erişim kontrol modeli uygulanır. İstemci her bağlantıda UNIX kullanıcı ve grup id'sini gönderir.

Bu çok güvenli bir yöntem değildir. Ancak NFS istendiğinde Kerberos ağ yetkilendirme sistemini kullanacak şekilde ayarlanabilir.

B. Andrew Dosya Sistemi (AFS)

Andrew Dosya Sistemi 1983 yılında Carnegie Mellon Üniversitesi (CMU) ve IBM işbirliği ile geliştirilmeye başlanmıştır. Tasarımın temel önceliği ölçeklenebilir olmasıdır. AFS ekibi hedeflerini dosya sisteminin 5000 istemciye kadar desteklenmesi olarak koymuşlardır. AFS'nin üç sürümü mevcuttur. Tasarım esasları şöyle açıklanmıştır [12]: (i) Bir iş hem istemci hem sunucuda yapılabiliyorsa istemci de yapılmalıdır. Sunucuya sadece kritik işlemler yaptırılmalıdır. (ii) Mümkün oldukça önbellekleme yapılmalıdır. (iii) Tasarımda genel kullanım alışkanlıklarından faydalanılmalıdır. (iv) Sistem bazında veri ve değişim minimumda tutulmalıdır. (v) Mümkün oldukça az nesneye güvenilmelidir. (vi) İşlemler mümkünse toplu halde yapılmalıdır.

AFS bazı kullanım alışkanlıkları gözetilerek tasarlanmıştır. Sistem komutları, kütüphaneleri ve kullanıcı home dizini gibi dosyaların yerel önbellekteki kopyaları uzun süreler boyunca geçerli olacaktır. Dosya erişimlerinin birçoğu dosyalara yapılmaktadır. Sistem dosyaları nadiren değişecektir ve kullanıcı home dizininde yapılan değişiklik önbellekte olmaktadır. AFS dizaynı dosya boyutları ve kullanımları üzerine bazı varsayımlardan etkilenmiştir. Bu varsayımlar UNIX sistemler üzerine yapılan bazı gözlemler ve çalışmalara dayanır [13]-[15]. Bunlardan en önemlileri: dosyalar genelde 10 KB'tan küçüktür, yazma işlemlerinden fazla okuma işlemi yapılır, genelde sıralı erişim yapılır/rastgele erişim nadirdir, çoğu dosya sadece tek kullanıcı tarafından kullanılır. AFS'de çalışan istemci yazılımları Venus, sunucu yazılımlarına ve bazen sunucuların tamamına Vice ismi verilir.

İsimlendirme ve Şeffaflık

AFS dosya isim alanı ikiye ayrılır: yerel ve paylaşılan. Paylaşılan isim alanı şeffaf bir şekilde bütün kullanıcılarda aynıdır. İsim alanının parçalara ayrılması şeffaflığın biraz azalmasına sebep olsa da bu durumun sıradan kullanıcılar tarafından fark edilme olasılığı düşüktür. AFS-'de adres dönüşümleri çok fazla performans kaybına yol açtığı için [16] AFS-2'de hacim denilen mantıksal yapılar oluşturulmuştur. AFS aynı zamanda dosya belirteci yapısını kullanmaktadır. Hacim bilgileri her sunucuda kopyalanmış bir durumda bulunan hacim konum veritabanında bulunur [6].

Senkronizasyon

AFS senkronizasyon için oturum semantiğini kullanmaktadır. Dosyalar kapatılınca değişiklikler sunucuya gönderilir. Eğer iki istemci aynı dosyaya yazma işlemi yaptılarsa değişiklikleri sonra gönderen istemcinin değişiklikleri geçerli olacaktır.

Tutarlılık ve Replikasyon

NFS'in aksine AFSv1 ve AFSv2 de dosyalar ve dizinler parçalar halinde değil bütünler halinde yollanır ve bu şekilde önbelleklenir. AFSv3 için 64 KB'lık parçalar kullanılır [8]. Burada amaç okuma işlemleri için sunucuyu devreden çıkarmak ve sunucuya sadece değişiklikleri göndermek veya almak için başvurmaktır. AFSv1 de önbelleğin kontrolü çok

fazla kaynak tükettiği için [16] AFSv2 de "callback" adı verilen bir mekanizma getirilmiştir. Buna göre istemci aksi söylenene kadar bütün önbellekleri geçerli kabul edecektir. Bu sistem şöyle işlemektedir. Bir istemci bir önbelleklediği zaman o dosya için sunucudan bir callback jetonu alır ve sunucu o istemcinin o dosyayı önbelleklediğine ve o dosya üzerinde yapılacak değişikliklerden haberdar edilmesi gerektiğine dair "callback promise" adlı bir kayıt tutar.

Callback jetonlarının geçerliliği sürekli değil sadece ilgili dosya açılırken sorgulanır. İstemci bir dosya da değişiklik yaptığı zaman bunu sunucuya bildirir. Sunucu o dosya için callback promise kaydı tuttuğu bütün istemcilere o callback jetonlarını geçersiz kılar. Böyle o dosyanın kopyasını tutan bütün istemciler dosyayı açmak istediklerinde callback jetonlarının geçersiz olduğunu görüp sunucudan dosyayı tekrar indireceklerdir.

Hata Toleransı

AFS durumlu bir servis olduğu için hata toleransı NFS'e göre biraz daha düşüktür. Bir istemci çöktüğünde callback jetonlarını çöpe atar ve sunucuya önbellekteki kopyalarının kontrol edilmesi için başvurur. Bir sunucu çökerse istemciler bunu fark edip jetonlarını çöpe atmalı ve önbellekteki kopyalarının kontrol edilmesi için başvurmalıdırlar.

Güvenlik

AFS'de güvenlik işlevi Vice sunucularına devredilmiştir. Bu sunucular fiziken güvenli kabul edilir ve üzerilerinde sadece güvenilen sistem yazılımları çalıştırılır. AFS güvenli UMÇ ve erişim kontrol mekanizmalarını içermektedir. Ayrıca AFSv3'ten itibaren yetkilendirme için Kerberos kullanılmaya başlanmıştır.

Türevleri

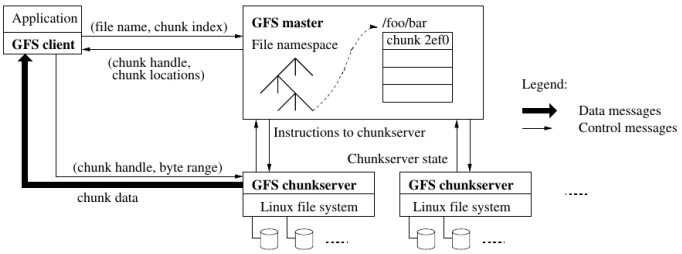
OpenAFS [17]: 1989 yılında CMU'da çalışan bazı araştırmacılar AFS için kurumsal çözümler üretmek üzere Transarc isminde bir firma kurdular. Bu firma 1994 yılında IBM tarafından satın alındı. Transarc AFS 2000 yılına kadar IBM tarafından geliştirildikten sonra OpenAFS adıyla açık kaynaklı olarak yayınlandı. OpenAFS bir AFSv3 gerçeklemesidir.

Coda [9]: CMU'da AFS'i geliştiren Prof. Dr. Mahadev Satyanarayanan ve ekibinin AFS-2 tabanlı bir araştırma projesidir. AFS-2'den temel farkı sunucu tarafında replikasyon, çevrimdışı işlem gibi erişilebilirliği arttıran özellikler getirmesidir. Hacim Depolama Grubu (VSG) çökmelerinde tutarlılığı sağlamak için versiyon vektörleri kullanılır.

Arla [18]: İsveç'te KTH Kraliyet Teknoloji Enstitüsü tarafından geliştirilen bir AFS istemcisidir. Arla'nın OpenAFS'ten temel farkı temel işlevlerin çekirdeğe yerleştirilmeyip kullanıcı seviyesinde daemonlar kullanılarak halledilmeleridir.

C. Google Dosya Sistemi (GFS)

Google yaptığı iş gereği çok büyük verilerle çalışmaktadır. Verileri yüksek kapasiteli az sayıda sunucuda depolamak ve işlemek oldukça maliyetli olmakta ve ölçeklenebilir olmamaktadır. Çok miktarlarda ucuz donanım kullanıp sistemi hatalara karşı daha dayanıklı olması için tasarlamak daha uygun bir çözümdür. GFS'de mantıksal dosya gösterimine yığın denir her bir yığın 64 MB veriden oluşur.



Şekil. 2 GFS mimarisi

Şekil 2'de görüldüğü gibi GFS mimarisi bir adet mastır sunucu ve çokça yığın sunucuları bulunur. Verilerin kendileri yığın sunucularda bulunurken mastır sunucuda sadece meta-veri bulunur. Mastır sunucu yığın sunucularla sürekli kalp atışı adı verilen mesajlarla haberleşir. İsim alanının yönetilmesi, yığınların yaratılması, kopyalanması, dengelenmesi, eski yığınların silinmesi mastırın görevidir.

Mastır aynı zaman çöp toplama işini de yapar. GFS'de dosyalar direk silinmez, silinecek olarak işaretlenir. Silme işlemi çöp toplama sırasında yapılır.

İsimlendirme ve Şeffaflık

GFS'de dizinler için ayrı bir veri yapısı bulunmaz. Ayrıca kısa yol, bağ gibi yapılar da mevcut değildir. GFS isim alanını mantıksal olarak her yol isimlerini meta-verilere eşleyen lookup tablolarıyla düzenler. Bu etkin bir bellek kullanımı sağlar.

GFS mastır içerisinde yapılan bazı işlemler uzun sürebileceği için bunu diğer işlemleri etkilememesi için kilitleme mekanizmaları kullanılır. Kilitler dosya yolu ağacındaki her bir düğüm için alınır.

Senkronizasyon

GFS'de dosyalar iki türlü işlem ile değiştirilebilir: yazma ve kayıt ekleme. Yazma işlemleri için mastır yığınlar arasından bir adet birincil yığın seçer. Yazma işlemi bu yığına yapılır ve buradan yığının diğer kopyalarına dağıtılır. Birincil yığın 60 saniye sonra bu yetkiyi mastıra iade eder. Eğer işi bitmemişse yetkinin uzatılmasını isteyebilir. Ağın kapasitesinin gereksiz yere harcanmaması için her sunucu kendine ağ topolojisinde en yakın sunucuya gönderimde bulunur.

Kayıt ekleme işleminde yazılacak verinin konumu belirtilmez sadece veri belirtilir. GFS bu veriyi kendi seçeceği bir yere yazar ve yazdığı yeri istemciye söyler. Bu sistem birden fazla istemci bir dosyaya yazmaya çalıştığında veri kaybını önlemektedir.

Tutarlılık ve Replikasyon

GFS'te istemciler için önbellekler tutulmaz. Ancak her yığının kopyaları (ortalama üç kopya) çeşitli sunucularda bulundurulur. Mastır'ın operasyon kayıtları çeşitli sunucularda bulundurulur. Mastır'ın kayıtlarına sahip gölge mastırlar salt-okunur hizmet verebilir.

Hata Toleransı

Mastır ve yığın sunucular saniyeler içerisinde tekrar başlatılabilirler. Mastır tekrar başlatıldığında öncelikle diskten operasyon kaydını okur. Sonra bütün yığın sunucularına ellerindeki yığınları sorar. Yığın versiyonlarına göre kendi bilgisini günceller veya bir yığını eski olarak belirler. Eğer bir yığın sunucu çökerse bu kalp atışı mesajları gelmediği için fark edilir ve yığınların başka kopyaları değişik sunucularda tekrar oluşturulur.

Türevleri

Hadoop Dosya Sistemi (HDFS): HDFS Apache Vakfı tarafından geliştirilen Hadoop projesinin bir parçasıdır. HDFS, GFS'nin açık kaynaklı bir gerçeklemesidir. Orijinal GFS, C++ ile yazılmışken HDFS Java ile yazılmıştır. HDFS, GFS gibi kayıt ekleme, çöp toplama gibi mekanizmaları sağlayamamaktadır [19].

V. KARŞILAŞTIRMA VE SONUÇLAR

Bu çalışmada belirli ölçütler açısından dosya sistemleri incelenmiş bazı benzerliklere ve karşıtlıklara rastlanmıştır (Tablo 1'e bakınız.).

Tablo. 1 Dağıtık dosya sistemlerinin karşılaştırılması

Kriter	NFS	AFS		GFS
		CODA	OpenAFS	
Tasarım amacı	Erişim saydamlığı	Yüksek elde edilebilirlik	Yüksek elde edilebilirlik	Veri depolama ve kümeleme
Erişim modeli	Uzak	Upload/ Download	Upload/ Download	Seri numara ile kiralama
İletişim	UMÇ	UMÇ	UMÇ	Kalp atışı mesajı
Bağlanma büyüklüğü	Dizin	Dosya sistemi	Dosya sistemi	Chunk
İsim uzayı	İstemci başına	Global	Global	Yerleşim bağımsız
Replikasyon	Minimal	ROWA*	ROWA	Master veya chunk replikasyonu
Hataya dayanıklılık	Güvenli iletişim	Replikasyon ve önbellekleme	Replikasyon ve önbellekleme	Hızlı kurtarma ve replikasyon
Önbellekleme birimi	Dosya (NFS v4'te)	Dosya	Dosya	Chunk
Erişim kontrolü	İşlemler ile	Dizin işlemleri	Dizin işlemleri	UNIX tabanlı

*ROWA: Read-one, Write-all yaklaşımı.

NFS ve AFS dosya sistemleri birbirlerinin muadilleri gibi gözükmektedirler. Ancak dizaynları arasında büyük farklar mevcuttur. NFS daha ziyade UNIX dosya sistemini dağıtık olarak sağlamaya çalışırken AFS performans ve ölçeklenebilirlik odaklıdır. AFS bazı konulara NFS'den daha iyi çözümler getirmiş ve daha iyi performans sağlamış olsa bile ticari başarı yakalayamamış ve NFS'nin gerisinde kalmıştır.

GFS şu anda mevcut en güçlü DFS gibi gözükmektedir. Belirli bir amaç için özelleşmiştir ve yenilikçi çözümler getirmektedir. Muadili olan HDFS bu mimariyi açık kaynaklı olarak sağlamaya çalışmasına rağmen GFS'in bir kaç sene gerisinde gözükmektedir.

KAYNAKLAR

[1] *IBM PC DOS and Microsoft Windows User's Guide*, IBM Corporation, Indianapolis, IN: Que Corporation, 1995.

[2] *Inside Macintosh*, Volume II, Apple Computer, Inc., Addison Wesley, 1985.

[3] G. Letwin, *Inside OS/2*, Microsoft Press, 1988.

[4] D. M. Ritchie, K. Thompson, "The Unix Time Sharing System", *Communications of the ACM*, vol. 17, no. 7, July, 1974.

[5] M. Satyanarayanan, "A Survey of Distributed File Systems", *Annual Review of Computer Science*, vol. 4, pp. 73-104, 1989.

[6] E. Levy, A. Silberschatz, "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*, vol. 22, no.4, pp. 321-374, 1990.

[7] ISO/IEC 10746, "Open Distributed Processing Reference Model", 1995.

[8] G. Coulourois, J. Dollimore, T. Kindberg, G. Blair, *Distributed Systems: Concepts and Design*", Addison-Wesley, 5. Edition, ABD, 2011.

[9] A. S. Tannenbaum, M. Van Steen, *Distributed Systems: Principle and Paradigms*, Pearson Education, 2. Edition, ABD, 2006.

[10] http://en.wikipedia.org/wiki/Distributed_hash_table (accessed 19 Feb. 2013)

[11] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, "Design and Implementation of the Sun Network File System", USENIX Association Conference Proceedings, pp. 119-130, June 1985.

[12] M. Satyanarayanan, "Scalable, Secure, and Highly-Available Distributed File Access", *Computer*, vol. 23, no. 5, pp. 9-18, 1990.

[13] M. Satyanarayanan, A study of file sizes and functional lifetimes. Proceedings of the 8th ACM Symposium on Operating System Principles, Asilomar, CA, pp. 96-108, 1981.

[14] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, J. Thompson, A Trace-driven analysis of the UNIX 4.2 BSD file system. Proc. of the 10th ACM Symposium Operating System Principles, pp. 15-24, 1985.

[15] R. Floyd, Short term file reference patterns in a UNIX environment. Technical Rep. TR 177, Rochester, NY: Dept of Computer Science, University of Rochester, 1986.

[16] R. Arpacı-Dusseau, A. Arpacı-Dusseau, *Operating Systems: Three Easy Pieces*, 2012.

[17] http://www.openafs.org/ (accessed 19 Feb. 2013)

[18] http://www.stacken.kth.se/project/arla/ (accessed 19 Feb. 2013)

[19] N. Shankaran, R. Sharma, "Cloud Storage Systems - A Survey", 2011.