Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 1

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 1: Dersin tanıtımı, Dağıtık sistemlere giriş,

Dağıtık sistemin amaçları neler

olduğuna kısa bir bakış?

Bu ders neyi anlatıyor?

• Dersin amacı, öğrencilere dağıtık sistemlerle ilgili temel

kavramlar olan donanım ve yazılım kavramları hakkında bilgi

vermektir.

3

• Dağıtık sistemler, bir ağ ortamı yoluyla birbirine bağlanmış

olan ve mesajlaşma yolu ile birbirlerini koordine edebilen

sistemlerdir. Bu tarz sistemlerin geliştirilmesinde ana amaç

kaynakların paylaşımıdır.

• Genel karakteristikler olarak; dağıtık sistemin elemanlarının

eşzamanlı olarak çalışması, tüm sistemde kullanılan tek bir

saatin var olmayışı, elemanların birbirinden bağımsız olarak

bazı hatalar sonucu iş göremez hale gelebilmeleri sayılabilir.

Bu ders neyi anlatıyor?

• Ders kapsamında, iletişim ve katmanlı protokoller, Uzak

Yordam Çağrısı (RPC), Uzak Nesne Çağrısı (ROI),

Mesajlaşmaya

-dayalı iletişim, Akışa

-dayalı iletişim, süreçler,

4

Mesajlaşmaya

-dayalı iletişim, Akışa

-dayalı iletişim, süreçler,

kod taşıma (göçü) ve saat senkronizasyonu detayları ile

açıklanmaktadır.

Ders yıllık planı

• Hafta 1: Dağıtık sistemlere giriş,

• Hafta 2: Dağıtık sistemlerin amaçları, Donanım kavramları.

• Hafta 3: Yazılım Kavramları, İstemci-Sunucu modeli.

• Hafta 4: İletişim, Katmanlı protokoller, Uzak Yordam Çağrısı (RPC),

5

• Hafta 4: İletişim, Katmanlı protokoller, Uzak Yordam Çağrısı (RPC),

• Hafta 5: Uzak Nesne Çağrısı (ROI), Mesajlaşmaya-dayalı iletişim.

• Hafta 6: Akışa-dayalı iletişim, Süreçler, Altsüreçler,

• Hafta 7: İstemciler, Sunucular.

• Hafta 8: Ara sınav

• Hafta 9: Kod taşıma (göçü), Yazılım etmenleri.

• Hafta 10: İsimlendirme, Varlıkların isimlendirilmesi.

• Hafta 11: Hareketli varlıkların yerleştirilmesi, Referanssız varlıkların yok edilmesi.

• Hafta 12: Senkronizasyon, Saat senkronizasyonu, Mantıksal saatler, Genel durum.

• Hafta 13: Seçim algoritmaları, Karşılıklı dışlama, Dağıtık işler.

• Hafta 14: Dağıtık işler- Kısım 2.

• Değerlendirme Metotları:

6

%40 Vize,

%60 Final,

Derse katılım.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

7

8

Đşlemci

A) Dağıtık sistem örneği.

B) Dağıtık sistem örneği.

Dağıtık Sistemler

Đşlemci

Đşlemci

Đşlemci

Bellek

Bellek

Bellek

Đşlemci Đşlemci Đşlemci

Bellek

Bellek

Bellek

C) Paralel sistem örneği.

Dağıtık sistemin tanımı (1/4)

TANIM

1

:

• Dağıtık bir sistem:

Kullanıcılarına tutarlı tek bir sistem olarak görünen

9

bağımsız bilgisayarlar topluluğuna denilmektedir.

TANIM

2

:

• Dağıtık bir sistem:

Bağımsız, otonom ve heterojen bilgisayarlardan oluşur

ve kullanıcı bu sistemi tek bir uyumlu sistem gibi algılar,

çünkü farklılıkları kullanıcıdan gizler.

Bu tanımın iki cephesi vardır:

1-) İlk cephe donanımla ilgilidir, buna göre makineler özerk’tir

(otonom’dur).

2

-) İkinci cephe yazılım ile ilgilidir, kullanıcılar kendilerini tek bir

Dağıtık sistemin tanımı (2/4)

10

2

-) İkinci cephe yazılım ile ilgilidir, kullanıcılar kendilerini tek bir

sistem ile u

ğra

şıyor zannederler.

Da

ğıtık sistemlerin en önemli karakteristiklerinden bir tanesi, çe

şitli

bilgisayarlar arasındaki farklılıklar ve bunların arasındaki ileti

şimin

kullanıcıdan gizlenmesidir.

Aynı

şey, da

ğıtık sistemlerin iç organizasyonlarında da bulunur.

Bir di

ğer önemli karakteristik ise, kullanıcıların ve uygulamaların da

ğıtık bir

sistem ile tutarlı ve bir biçimli (uniform) bir biçimde etkile

şebilmesidir.

Burada etkile

şmenin nerede ve ne zaman oldu

ğu önemsizdir.

Da

ğıtık sisteme kullanıcı, uygulama ve bilgisayar ekleyip-çıkarmak kolaydır.

Dağıtık sistemler kolayca genişletilebilir veya ölçeklenebilirler.

Bu karakteristik doğrudan bağımsız bilgisayarlara sahip olunması nedeniyledir.

Fakat aynı zamanda bu durum, bu bilgisayarların gerçekte sistemde bir bütün

Dağıtık sistemin tanımı (3/4)

11

olarak nasıl yer aldıklarının (kullanıcıdan) gizlenmesi anlamına da gelmektedir.

Bir dağıtık sistem sürekli olarak kullanıma hazır durumdadır, buna rağmen bazı

parçalar/bileşenler geçici olarak hizmet dışı olabilir, bu sistemin tümünü

etkilemez. Kullanıcı ve uygulamalar, bu parçaların (bozuk veya hizmet dışı)

olduğundan, değiştirildiklerinden, tamir edildiklerinden veya sisteme yeni bir

parçanın eklendiğinden haberdar olmamaktadırlar.

Dağıtık sistemler, heterojen bilgisayarları ve ağları desteklemek için (tek bir

sistem görüşü) mantıksal olarak yüksek-seviyeli katmanlarla (yani kullanıcı ve

uygulamaları içeren katman) daha düşük katmanlar arasında (yani işletim

sistemi) bir yazılımsal katmanı önermektedirler. Buna Middleware (Ara katman

yazılımı veya aracı yazılım) denilmektedir.

Dağıtık sistemin tanımı (4/4)

12

1.1

Da

ğıtık bir sistem ara katman yazılımı (middleware) olarak organize olmu

ştur.

Middleware katmanı birden çok makinenin üzerinde bulunmaktadır.

Fig. 1.1

Bazı dağıtık sistem ile ilgili örneklere bakalım. Bir üniversite veya

kampus içerisindeki bilgisayar ağını düşünelim.

Her bir kullanıcının kişisel bilgisayarının bulunmasının yanı sıra, bir

Dağıtık sistem örneği (1)

13

makine dairesi gibi ana merkezde tüm kullanıcıların erişebildiği bir

işlemciler havuzu da oluşturulmuş ve bunun belirli kullanıcılara

ataması yapılmamış, fakat istenildiğinde erişebilmelerine izin verilmiş

olabilir.

Böyle bir sistem tek bir dosya sistemine sahip olabilir. Bu sistem tüm

dosyaların aynı yolla tüm makinelerden erişilebilir olduğu ve aynı yol

ismini kullandığı anlamına gelir.

Dağıtık sistem örneği (2)

• İkinci bir örnek olarak, bir iş akışı bilgi sistemi’ni göz önüne alalım. Tipik olarak, böyle bir sistem insanlar tarafından farklı departmanlarda, olası olarak farklı yerleşimlerde kullanılabilir.

• Örneğin, satış departmanındaki insanlar büyük bir bölgeye yayılmış veya bir

14

bütün ülkeye yayılmış olabilirler. Telefon ağı veya cep telefonu ağı ile sisteme bağlanmış dizüstü bilgisayarlar aracılığıyla siparişler alınıyor olabilir.

• Gelen siparişler, otomatik olarak planlama departmanına iletilir, bundan sonra gelen iç taşıma emirleri de stok departmanına iletilir, bu yolla ödemeler muhasebe departmanına kadar gelir. Sistem otomatik olarak uygun ve boşta bulunan bir kişiye sipariş emirlerini iletir. Kullanıcılar toplamda fiziksel bir siparişin bu sistem üzerinde nasıl olup da iletildiğinden (yani siparişin hangi aşamalardan geçtiğinden) haberdar olmamaktadır.

Dağıtık sistem örneği (3)

• Üçüncü bir örnek olarak, Dünya çapında Veb

(World Wide Web,

WWW) verilebilir. Veb dağıtık dökümanlar için basit, tutarlı ve bir biçimli (uniform) modeli önermektedir.

• Bir dökümanı görmek için bir kullanıcının bir referansı aktif etmesi gerekir.

15

• Bir dökümanı görmek için bir kullanıcının bir referansı aktif etmesi gerekir. Böylece doküman ekranda görünür hale gelir.

• Bir doküman’ın yayınlanması ise oldukça basittir, Bir biçimli veya Tekdüze Kaynak Yerleştirici

(Uniform Resource Locator, URL)

biçiminde bir adres verilmesi yeterli olmaktadır.

• Eğer WWW, kullanıcılarına devasa bir merkezi doküman sistemi olarak gözüküyor ise, bu aynı zaman da bir dağıtık sistemdir denilebilir. Fakat tüm Veb hala o aşamaya gelememiştir.

• Örneğin, kullanıcılar, dokümanların nerede yerleştiğinden/tutulduğundan ve hangi sunucular tarafından servis edilip/barındırıldığından haberdardır.

Dağıtık sistemin amaçları

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources)

16

2-) Saydamlık (Transparency)

3-) Açıklık (Openness)

4

-) Ölçeklenebilirlik (Scalability)

Dağıtık sistemin amaçları

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources)

Farklı kullanıcıların farklı uzak kaynaklara erişimi ve birbirleriyle bu

kaynakları paylaşabilmelerinin sağlanmasıdır.

17

2-) Saydamlık (Transparency)

Bir dağıtık sistemdeki kaynakların ve süreçlerin fiziksel olarak bir çok

bilgisayar arasında dağıtılmış olduğunun kullanıcıdan gizlenmesidir.

3-) Açıklık (Openness)

Bir açık dağıtık sistem, standart kuralları olan servislerden oluşan sistemdir.

Bu kuralları, bu servislerin syntax ve semantiklerini (sözdizim ve

anlamsallıklarını) tanımlamaktadır.

4-) Ölçeklenebilirlik (Scalability)

Dağıtık sisteme yeni bileşen ekleme-çıkarılması ile ilgili konudur.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

18

Ders’in sonu…

19

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 2

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 2: Dağıtık sistemlerin amaçları,

Donanım kavramları.

DAĞITIK SĐSTEMLERĐN AMAÇLARI:

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources)

Farklı kullanıcıların farklı uzak kaynaklara erişimi ve birbirleriyle bu

kaynakları paylaşabilmelerinin sağlanmasıdır.

3

2-) Saydamlık (Transparency)

Bir dağıtık sistemdeki kaynakların ve süreçlerin fiziksel olarak bir çok

bilgisayar arasında dağıtılmış olduğunun kullanıcıdan gizlenmesidir.

3-) Açıklık (Openness)

Bir açık dağıtık sistem, standart kuralları olan servislerden oluşan sistemdir.

Bu kuralları, bu servislerin syntax ve semantiklerini (sözdizim ve

anlamsallıklarını) tanımlamaktadır.

4-) Ölçeklenebilirlik (Scalability)

Dağıtık sisteme yeni bileşen ekleme-çıkarılması ile ilgili konudur.

Dağıtık sistemin amaçları

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources)

Farklı kullanıcıların farklı uzak kaynaklara erişimi ve birbirleriyle bu

4

kaynakları paylaşabilmelerinin sağlanmasına “kaynakları ve kullanıcıları

bağlamak/bir araya getirmek” denilmektedir. Bu işlem kontrollü bir yoldan

yapılır. Kaynaklar sanal olarak her şey olabilir, fakat tipik örnekler arasında

yazıcılar, bilgisayarlar, depolama birimleri, veriler ve dosyalar, Veb sayfaları

ve ağlar bulunabilmektedir.

Kaynakların paylaşılması için bir çok neden vardır. En belirgin neden ise

ekonomiktir. Örneğin bir yazıcıyı birden çok kullanıcının kullanması, her

bir kullanıcıya ayrı ayrı birer yazıcı alınmasından daha ucuz bir maliyete

sahiptir. Buna benzer olarak yüksek hesaplama kapasitelerine sahip süper

bilgisayarlar ve yüksek depolama sistemleri de paylaşılan kaynaklar ile daha

ucuz ve etkin kullanılabilir hale getirilmektedirler.

Dağıtık sistemin amaçları

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources

)

Kaynakların ve kullanıcıların birbirlerine bağlanılması ile, bunlar arasında

5

ortak iş yapmak ve bilgi alışverişinin yapılması daha kolay hale gelmiştir.

Buna en iyi örnek Internet ve onun basit protokolleridir, bu protokoller ile

dosya alışverişi, posta, dokümanlar, ses ve video alışverişi rahatlıkla

yapılabilmektedir.

Internet’in bağlanabilirliği (connectivity) sayesinde çeşitli sanal

organizasyonların da oluşturulması mümkün hale gelmiştir. Bunlar grup

yazılımı

(groupware) denilen, coğrafik olarak dağıtılmış fakat ortak bir iş

için çalışan insanlar için geliştirilen; düzenleme, telekonferans ve benzeri

işlemlerin yapılmasını sağlayan bir tür yazılımdır.

Dağıtık sistemin amaçları

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources)

Bununla birlikte, bağlanabilirlik ve paylaşım arttıkça, güvenlik daha ve

6

daha önemli bir hal almaktadır. Şu anki pratikte, sistemler küçük bir

koruma karşısında iletişimde gizlice dinleme’ye (eavesdropping) veya sızma

(intrusion) maruz kalabilmektedir.

Şifreler ve diğer hassas bilgiler sıklıkla açık metin (cleartext) olarak (yani

şifrelenmemiş) ağ üzerinden gönderilir, veya sunucularda depolanır, bu

durumlarda bu yolun güvenli olduğunu sadece umut ederiz.

Bu bakış açısıyla, geliştirmeye oldukça fazla ihtiyaç duyulduğu bellidir.

Örneğin, kredi kartı ile bir ürünün alımında, müşterinin bu kredi kartına

gerçekten sahip olup olmadığının kanıtlanması gerekmektedir.

Dağıtık sistemin amaçları

1-) Kaynakları ve kullanıcıları bağlamak/bir araya getirmek

(Connecting Users and Resources)

Başka bir güvenlik problemi ise belirli bir kullanıcıya ait özellik profilinin

7

oluşturularak iletişimin takip edilmesidir. Böyle bir izleme, kişisel gizliliğin

(privacy) ihlali anlamına gelmektedir, özellikle bu kullanıcıyı

bilgilendirmeden yapılırsa daha da kötü bir durum oluşur.

Bununla ilişkili bir problem, bağlanabilirliğin (connectivity) arttığı sırada

istenmeyen iletişimin de ortaya çıkması olarak verilebilir, böylece spam adı

verilen bilgi kirliliğine neden olan e-postalar oluşmaktadır. Bu durumlarda,

özel bilgi süzgeçleri

(information filters) ile kendimizi korumamız

önerilebilir.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Dağıtık bir sistemdeki kaynakların ve süreçlerin fiziksel olarak bir çok

bilgisayar arasında dağıtılmış olduğunun kullanıcıdan gizlenmesidir.

8

Böylece bir dağıtık sistem kendini kullanıcısına ve uygulamalarına karşı

sanki tek bir bilgisayarmış gibi gösterebilmektedir. Bu duruma saydamlık

(transparency) denilmektedir.

Dağıtık sistemlerdeki saydamlık çeşitlerine bakılarak, saydamlığın daima

ne zaman gerektiği konusuna bakalım.

Dağıtık bir sistemdeki saydamlık (transparency)

Transparency

(saydamlık) Description (tanım)

Access (Erişim) Veri gösterimi ve kaynağa nasıl erişildiğinin gizlenmesidir.

9

Location (Yerleşim) Kaynağın nerede bulunduğunun gizlenmesidir.

Migration (Göç) Kaynağın başka yere taşındığının gizlenmesidir.

Relocation (Tekrar

yerleşim)

Kaynak kullanımdayken başka yere taşındığının

gizlenmesidir.

Replication

(Kopyalama) Kaynağın kopyalandığının gizlenmesidir.

Concurrency

(Eşzamanlılık)

Kaynağın bir kaç farklı kullanıcı tarafından

paylaşıldığının gizlenmesidir.

Da

ğıtık bir sistemdeki saydamlık çe

şitlerine baktı

ğımızda, tabloda verilenler

kullanıcıdan neyin saklanıldı

ğını göstermektedir. (Kullanıcının fark etmemesi için)

(Eşzamanlılık) paylaşıldığının gizlenmesidir.

Failure (Hata) Kaynaktaki hata ve kurtarmayı gizlemektedir.

Persistence

(Kalıcılık) Kaynağın bellek veya diskte olduğunun gizlenmesidir.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Erişim saydamlığı (Access Transparency):

Erişim saydamlığı, kullanıcı tarafından kaynaklara erişim şeklini ve veri

10

gösterimindeki farklılıkları gizlemek ile ilgilenmektedir. Örneğin, bir

tamsayıyı Intel-tabanlı iş istasyonundan bir Sun SPARC makineye

gönderme durumunu ele alalım.

Bilindiği gibi, Intel işlemciler byte’ları little endian formatında

dizmektedir (yüksek-dereceli byte’lar ilk önce iletilmektedirler) ve

SPARC işlemciler ise big endian formatında (yani düşük-mertebeden

byte’lar ilk önce iletilir). Veri gösteriminde diğer farklılıklar da

bulunabilir.

Örneğin, dağıtık bir sistem, farklı işletim sistemlerini çalıştıran

bilgisayarlara sahip olabilir. Her bir bilgisayar kendi dosyaisimlendirme şartlarına (file-naming conventions) sahip olabilir.

İsimlendirme şartlarındaki farklılıklar, dosyaların nasıl yönetilebildiği,

gibi konuların hepsi kullanıcılardan ve uygulamalardan gizlenmesi

gerekmektedir.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Yerleşim saydamlığı (Location Transparency):

Diğer bir önemli saydamlık ise kaynakların yerleşimi ile ilgilenmektedir.

11

Bu tarz saydamlık, kaynağın fiziksel olarak nerede bulunduğunun

kullanıcıdan gizlenmesi ile ilgilenir. Yerleşim saydamlığı için,

isimlendirme

(naming) önemli bir yer tutmaktadır.

Özellikle, yerleşim saydamlığı sadece mantıksal isimlerin kaynaklara

atanması ile başarılabilmektedir. Bir kaynağın yerleşimindeki isimler gizli

olarak kodlanmaz. Böyle bir isim için örnek olarak URL verilebilir

(http://www.isim.com/anasayfa.html).

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Göç saydamlığı (Migration transparency) :

Dağıtık sistemlerdeki herhangi bir kaynağın, bu kaynağa nasıl

12

erişildiğinden etkilenmeden etkilenmeden taşınabilmesine göç

saydamlığı

(migration transparency) denilmektedir.

Buna özetle; Kaynak pasif iken taşıma diyebiliriz.

Tekrar yerleşim (Relocation transparency) :

Daha güçlü bir durum ise, herhangi bir kaynağa erişim yapılırken

kullanıcı veya uygulama’nın hiçbir şekilde uyarılmadığı durumda

kaynakların yer değiştirilebilmesi (tekrar yerleştirilebilmesi) ‘dir. Bu

durumlarda, sistem için tekrar yerleşim saydamlığı (relocation

transparency) desteklemektedir denilir. Bu tarz saydamlığa örnek

olarak, kablosuz bağlantıya sahip dizüstü bilgisayarların bölgeden

bölgeye taşınırken (geçici olarak) bağlantısının koparılmasının

gerekmemesi verilebilir.

Buna özetle; Kaynak aktif iken taşıma diyebiliriz.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Kopyalama saydamlığı (Replication transparency):

Kopyalama dağıtık sistemlerde önemli rol oynamaktadır. Örneğin,

13

kaynaklar kullanırlığı artırmak veya başarımı (performance) iyileştirmek

için kopyalanabilir (replicated). Bu iş için erişimin yapıldığı yere daha

yakın bir kopya yerleştirilerek başarım artırılabilir.

Kopyalama saydamlığı bir kaynağın birkaç kopyasının var olduğunu

kullanıcıdan gizlenmesi ile ilgilidir. Kopyalamanın kullanıcıdan

gizlenmesi için, tüm kopyaların aynı isme sahip olması gerekir.

Sonuçta, bir sistem kopyalama saydamlığını destekliyorsa, genel olarak

yerleşim saydamlığını da desteklemesi gerekir, çünkü aksi halde farklı

yerleşimlerdeki kopyaların işaret edilmesi imkansız olmaktadır.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Eşzamanlılık saydamlığı (Concurrency transparency) :

Kaynakların paylaşımı önemli bir konudur. Bir çok durumda, kaynakların

14

paylaşımı ortaklaşa bir biçimde iletişim yoluyla yapılır. Bununla beraber,

bir çok yarışmacı kaynak paylaşımı örnekleri de mevcuttur. Örneğin, iki

bağımsız kullanıcının her biri kendi dosyalarını aynı dosya sunucusunda

depolamış veya paylaşımlı bir veritabanındaki aynı tablolara erişiyor

olabilirler. Bu durumlarda, her bir kullanıcının diğer kullanıcının aynı

kaynağı kullandığından haberi olmaması önemlidir. Bu fenomene

eşzamanlılık saydamlığı

(concurrency transparency)

denilmektedir.

Buradaki en önemli konu ise, eşzamanlı bir erişimin paylaşımlı bir

kaynağı tutarlı

(consistent) durumda bırakabilmesidir. Tutarlılık

(consistency) kilitleme mekanizmaları (locking mechanisms) ile

başarılabilmektedir. Bu yolla kullanıcılar, kendi sıraları geldiğinde,

istenilen kaynağa dışlayan bir erişim (exclusive access) ile erişirler.

Daha rafine bir yaklaşım olarak, işlemler

(transactions) de

kullanılabilmektedir.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Hata saydamlığı (failure transparency) :

Dağıtık sistemler için popüler alternatif bir yöntem ise, Leslie

Lamport’un ifadesiyle “Bir bilgisayar çökse bile, başladığınız herhangi

bir işin durduğunu (yarıda kaldığını) görmeyeceğiniz yapıdaki bir

15

sistem, dağıtık bir sistemdir” şeklinde verilebilir. Bu tanım, dağıtık

sistemler için önemli diğer bir konuyu ortaya koymaktadır: Hatalar ile

uğraşmak.

Dağıtık bir sistemi hataya karşı saydam yapmak için bir kullanıcı’nın bir

kaynağın düzgün çalışmaması hakkında hiçbir bilgisinin olmaması

gerekmektedir. Ve bu sistem daha sonrasında bu hatadan kurtarılabilir.

Hataları gizlemek (Masking failures), dağıtık sistemler için oldukça zor

bir uğraş konusudur. Gerçek varsayımlar yapıldığında bazı durumlarda

imkansız bile olabilmektedir.

Buna başlıca nedenler arasında, ölü bir kaynak ile (artık çalışmayan bir

kaynak) çok çok yavaş bir kaynak arasındaki farkı ayırt etmenin güçlüğü

verilebilir. Buna örnek olarak, çok yoğun bir Veb sunucusuna

erişilemediği zaman sunucunun ölü mü yoksa çok çok yavaş erişilebilir

mi olduğuna kullanıcının kesin olarak karar verememesi gösterilebilir.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Kalıcı saydamlık (Persistence transparency):

Saydamlık için bir diğer tarz ise, kalıcı saydamlık

(persistence

transparency) ‘tır. Bu tarz saydamlık, bir kaynağın geçici bellekte veya

16

disk üzerinde herhangi bir yerde olduğunun kullanıcından gizlenilmesi

ile uğraşmaktadır. Örnek olarak, bir çok nesneye-yönelik veritabanı

depolanmış nesnelerin doğrudan çağrılmasına olanaklar sunmaktadır.

Arkaplanda peki neler olmaktadır? İlk önce veritabanı nesne durumunu

diskten ana belleğe kopyalamaktadır, yapılacak işlemi gerçekleştirmekte

ve belki ikinci bir depolama bölgesine bu durumu da yazmaktadır.

Kullanıcı bununla beraber, sunucunun ilk ile ikinci bellek arasında bu

durumu taşıdığından haberdar olmamaktadır. Kalıcılık, dağıtık

sistemlerde önemli bir rol oynamaktadır, fakat aynı zamanda dağıtık

olmayan sistemler için de önemlidir.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Saydamlığın derecesi (Degree of Transparency):

Dağılım saydamlığı genellikle herhangi bir dağıtık sistem için önerilebilir

17

olmasında rağmen, tüm dağılım cephelerinin kullanıcılardan tamamen

gizlenilmesinin her zaman iyi bir fikir olmadığı durumlarda mevcuttur.

Buna örnek olarak bir elektronik gazetenin, bizim posta kutumuza yerel

saat ile sabah saat 7:00 dan önce ulaşmasını dünyanın bir diğer ucunda

olsak bile istiyor olalım. Sabah okuyacağınız gazete artık gerçekten sabah

gazetesi olmayabilir (saat farkı nedeniyle).

Buna benzer olarak, geniş-çapta dağıtılmış bir dağıtık sistem, bir süreç

için Amerika’daki San Fransisco ‘ya bağlanırken bir diğeri de Türkiye’deki

Ankara’ya bağlansın. Buradaki iletişim için iki şehir arasında doğal

nedenlerle oluşan 35 milisaniyelik gecikmeyi kesinlik gizleyemezsiniz.

Pratikte bu iletişim 100 mili saniyeler mertebesinde gerçekleşir, burada

iletişim sadece ışık hızı (fiber optik kablolar) ile sınırlı değil, işlemi yapan

ara cihazların hızı ile ve işlem süresi ile sınırlanmıştır.

Dağıtık sistemin amaçları

2-) Saydamlık (Transparency)

Saydamlığın derecesi (Degree of Transparency):

Sistem başarımı (performance) ile saydamlığın derecesi arasında bir

ölçülülük (trade

-off) bulunmaktadır. Örneğin, bir çok Internet

18

uygulaması, bir sunucuya bağlanmaktan vazgeçmeden evvel defalarca

bağlanmayı denemektedir. Sonuçta, bir diğerini denemeden evvel geçici

sunucu hatasını gizlemeyi denemek sistemin tamamını yavaşlatabilir. Bu

durumda, ya daha önceden vazgeçmek uygun olacak ya da en azından

kullanıcının vazgeçebilmesine olanak sunulması yerinde olacaktır.

Bir diğer konu ise, birkaç kopya’nın farklı kıtalarda yerleşmiş olması ve

tüm zamanlarda tutarlı olmaları gerekliliğidir. Diğer bir deyişle, eğer bir

kopya değişirse, bu değişim diğer tüm kopyalara başka herhangi bir işlem

yapılmadan yayılacaktır. Tek bir güncelleme işlemi tamamlanmak için

saniyeleri alacaktır, bazı şeyler kullanıcıdan gizlenememektedir.

Sonuç olarak, saydamlık dağılımı için amaç dağıtık sistemlerin tasarımı ve

gerçekleştirimi için iyi bir amaçtır, fakat başarım göz önünde

bulundurulmalıdır.

Dağıtık sistemin amaçları

3-) Açıklık (Openness)

Dağıtık sistemler için diğer önemli bir konu ise açıklık (openness) olarak

verilir. Bir açık dağıtık sistem, bu sisteme ait servislerin anlamsallığı

(semantic) ve söz dizimi’nin (syntax) tanımlandığı standart kurallara göre

19

bazı servisleri sunmaktadır. Örneğin bilgisayar ağlarında, gönderilip-alınan

mesajların anlamını, içeriğini ve biçimini (format) yöneten böyle standart

kurallar bulunmaktadır. Bunlara protokoller denilmektedir. Dağıtık

sistemlerde, servisler genellikle arayüzler (interfaces) üzerinden

belirlenmektedir. Bunun için Arayüz Tanımlama Dili

(Interface

Definition Language, IDL) oluşturulmuştur.

Arayüz tanımları bir IDL’de neredeyse sadece servislerin söz dizimlerini

yakalamaktadır. Diğer bir deyişle, ortaya çıkan olası istisna (exception), geri

dönüş değerleri (return values), parametre tipleri ile kullanılır olan

fonksiyonların özellikle isimlerinin belirlenmesini arayüz tanımları sağlar.

Bu işin en zor kısmı bu servislerin nasıl yapılacağıdır ve bu arayüzlerin

anlamsallığıdır. Pratikte, böyle belirlenimler (specifications) formal

olmayan bir yoldan doğal dil ile verilmektedir.

Dağıtık sistemin amaçları

3-) Açıklık (Openness)

Eğer doğru bir biçimde belirlenebilirse, bir arayüz tanımı keyfi bir sürece

izin verebilir. Bu keyfi süreç, bu arayüz’ü sağlayan başka bir süreç ile

konuşmak için ana bir arayüz’e ihtiyaç duymaktadır. Bu arayüzlerin

20

tamamen farklı gerçekleştirimlerinin oluşturulması için iki tane bağımsız

parçaya izin verilir., böylece iki ayrı dağıtık sistem tam olarak aynı yoldan

çalışır. Uygun belirlenimler (specifications) tam ve doğaldırlar (neutral).

Buradaki tamlık, gerçekten belirlenmiş olan bir gerçekleştirimin yapılması

için gerekli her şey anlamındadır.

Bir çok arayüz tanımı tam değildir, bu yüzden geliştirici uygulamaya-özgü

detaylar eklemek zorunda kalır. Tamlık

(Completeness) ve Doğallık

(Neutrality) kavramları, birlikte çalışabilirlik

(interoperabilitiy) ve

taşınabilirlik

(portability) için önemlidirler.

Birlikte çalışabilirlik, sistemlerin iki gerçekleştirimini veya farklı üreticilerin

ürettiği bileşenlerin genişletilmesini karakterize ederek, bunların genel bir

standart tarafından belirlenmiş olan her birine ait servislerini bir arada

çalıştırabilmelerini sağlar.

Dağıtık sistemin amaçları

3-) Açıklık (Openness)

Taşınabilirlik

(Portability), dağıtık bir A sistemi için üretilen bir

uygulamanın, hiçbir değişikliğe maruz kalmadan farklı bir dağıtık sistem

olan B sisteminde çalıştırılabilmesini karakterize eder. B sistemi A sistemi

ile aynı arayüzleri gerçekleştirmektedir.

21

Açık dağıtık sistem aynı zamanda esnek (flexible) de olmak zorundadır.

Bunun anlamı, farklı geliştiricilerin olası farklı bileşenlerinden oluşan

sistemin ayarlanmasının ne kadar kolay yapılacağı ile ilgilidir. Böylece yeni

bir bileşen eklenilmesi veya var olan bir tanesinin değiştirilmesi diğerlerini

etkilemeden oldukça kolay olmalıdır.

Dağıtık sistemin amaçları

3-) Açıklık (Openness)

Politikayı Mekanizmadan ayırmak:

Açık dağıtık sistemlerde esnekliği

(flexibility) sağlamak için, sistemin

görece küçük ve kolaylıkla değiştirilebilir ve uyarlanabilir bileşenler’den

22

(adaptable components) oluşan bir topluluk olarak organize edilmesi

önemlidir. Bu gerçekleştirim, sadece yüksek-seviyeli arayüz’lerin

tanımlarını içermemeli, kullanıcı ve uygulamalar tarafından da

görülebilmelidir.

Bu yaklaşım, görece yenidir. Bir çok eski ve çağdaş sistem, bir yekpare

yaklaşım

(monolithic approach) kullanılarak oluşturulmuştur. Bu

yaklaşımda, bileşenler mantıksal olarak ayrıştırılmış fakat tek bir tane

büyük program olarak gerçekleştirilmişlerdir. Bu yaklaşım, bir bileşenin

tüm sistemi etkilemeden değiştirilmesini ve uyarlanabilmesini oldukça

zorlaştırmaktadır. Yekpare sistemler açık sistemlerden çok kapalı sistemlere

yakındırlar.

Dağıtık sistemin amaçları

3-) Açıklık (Openness)

Politikayı Mekanizmadan ayırmak: (devam)

Bir dağıtık sistemi değiştirmeye ihtiyaç duyulması sıklıkla bir bileşenin özel

bir kullanıcı veya uygulama için optimal kuralı/politika’yı (optimal policy)

23

sağlayamaması ile oluşmaktadır. Buna örnek olarak, Dünya-çapında Veb

için önbellekleme

(caching) göz önüne alındığında, bir kullanıcı’nın

önbellekleme politikasını uyarlamasına önbellek büyüklüğünü

ayarlayabilmesi ile izin verilir, böylece önbelleklenmiş bir doküman daima

veya sadece bir oturumda tutarlılık için kontrol edilebilir.

Bununla beraber, kullanıcı diğer önbellekleme parametrelerine etki

edemez, örneğin bir dokümanın önbellekten kadar uzun süre kalacağı veya

hangi dokümanın önbellekten temizleneceği gibi.

Ayrıca, bir dokümanın içeriğini temel alarak önbellekleme kararlarını

vermek imkansızdır. Örneğin, bir kullanıcı demiryolu sefer tablosunu

önbelleklemek istiyor olsun, bu tablo pek sık değişmeyeceği için bu işlem

yapılabilecektir, fakat otoyollardaki şu anki trafik durumunu önbelleklemek

ise imkansız olabilmektedir.

Dağıtık sistemin amaçları

3-) Açıklık (Openness)

Politikayı Mekanizmadan ayırmak: (devam)

Politikayı mekanizmadan ayırmak adına şunlar söylenebilir:

Politika, bir yerden bir yere gitmek, yani amaç olursa; Mekanizma ise,

24

Politika, bir yerden bir yere gitmek, yani amaç olursa; Mekanizma ise,

bir yerden bir yere nasıl gidildiğidir, yani araç olmaktadır, diye söylenebilir.

Veb önbellekleme örneğine tekrar bakacak olursak; bir Internet tarayıcısı

(browser) ideal olarak sadece dokümanları depolamak için olanaklar

sunmalıdır ve aynı zamanda kullanıcılara hangi dokümanın ne kadar süre

depolanacağına karar verebilmelerine de izin vermelidir. Pratikte, bu işlem

kullanıcının (dinamik olarak) ayarlayabildiği zengin bir parametre kümesi

önerilerek gerçekleştirilebilir.

Daha iyi bir durum ise kullanıcı’nın kendi kurallarını/politikasını Internet

tarayıcısına eklenebilecek şekilde bir bileşen biçiminde gerçekleştirebilmesidir. Tabii ki, bu bileşen bir arayüz’e sahip olmalıdır, böylece Internet

gezgini de bu arayüz’ün prosedürlerinin çağrıldığını anlayabilmelidir.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Dünya-çağında Internet üzerinden bağlanırlık (connectivity), dünyanın

diğer ucundaki bir insana bir posta kartı göndermek kadar genel hale

gelmiştir. Bunu aklımızda tutarak, ölçeklenebilirlik (scalability) konusuna

25

bakacak olursak, geliştiriciler için dağıtık bir sistemin tasarımındaki en

önemli konu olduğu görülür.

Bir sistemin ölçeklenebilirliği, en az üç farklı boyut ile ölçülebilir.

1

-) Boyut (Büyüklük) olarak: Sisteme kolayca daha fazla kullanıcı ve

kaynak eklenebilmesi anlamındadır.

2

-) Coğrafik olarak: Kullanıcı ve kaynakların birbirlerinde oldukça uzak

mesafelerde konumlandırılmış olmasıdır.

3

-) Yönetimsel olarak: Sistem bir çok bağımsız yönetimsel

(administrative) organizasyona bölünmüş bile olsa kolayca yönetilebilir

olmalıdır.

Bunlara göre ölçeklenebilirlik için bu

3 boyuttan birisi ortaya çıktığında

sistemin başarımı düşmektedir.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri:

Bir sistemin ölçeklenmesi gerekiyor ise, farklı tipteki problemlerin

çözülmesi de gerekmektedir. İlk önce, ölçeklenebilirlik için büyüklük

26

boyutunu ele alırsak; eğer daha fazla kullanıcı veya kaynak’ın

desteklenmesi gerekiyor ise, sıklıkla merkezi servisler, merkezi veriler ve

merkezi algoritmaların sınırlamaları ile karşı karşıya kalırız.

Örneğin, bir çok servis merkezidir, buna göre dağıtık sistem içerisindeki

belirli bir makine üzerinde çalışan tek bir sunucu tarafından

gerçekleştirilmektedir. Bu şema ile ilgili problem oldukça barizdir:

sunucu kullanıcı sayısı artıkça bir darboğaza girecektir. Sanal olarak

sınırsız işlem gücü ve depolama kapasitemiz olsa dahi iletişim yüzünden

sunucu daha fazla büyümeye izin vermeyecektir.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Ne yazık ki, sadece tek bir sunucu kullanmak bazen kaçınılmaz

olmaktadır. Şöyle bir şeyi göz önüne alalım, tıbbi kayıtlar, banka

27

hesapları, bireysel krediler gibi yüksek derecede gizli bir bilginin

yönetilmesi için bir servisimiz olsun. Bu durumda, bu servisi verecek tek

bir sunucuyu, ayrı bir güvenlikli odada tutarak ve özel ağ bileşenleri

üzerinden dağıtık sistemin diğer parçalarından koruyarak bu işlem

gerçekleştirilebilmektedir.

Sunucunun bir çok yerleşime kopyalanması ile başarım artırılması

sağlanabilmekte, fakat bu seferde güvenlik ataklarına karşı daha zayıf

hale (vulnerable) gelmektedir.

Ölçeklenebilirlik (Scalability) Problemleri

Concept (Kavram) Example (Örnek)

28

Centralized services

(Merkezileştirilmiş

hizmetler)

A single server for all users

(Tüm kullanıcılar için tek bir sunucu)

Centralized data

(Merkezileştirilmiş veri)

A single on-line telephone book

(Tek bir çevrim içi telefon defteri)

Centralized algorithms

(Merkezileştirilmiş

Doing routing based on complete information

(Yönlendirmenin tam bilgiyi temel alarak

Ölçeklenebilirlik sınırlandırmaları ile ilgili örnekler.

algoritmalar) yapmak)

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Merkezi servisler kadar kötü bir durumda merkezi veriler’dir. 50 milyon

insanın adreslerini ve telefon numaralarını nasıl takip etmeliyiz? Her bir

29

veri kaydının 50 karakter ile sınırlı olduğunu varsayalım. Tek bir 2.5

gigabyte’lık disk, depolama için yeterli olacaktır.

İletişim konusunu tekrar aklımıza getirirsek, tek bir veritabanına sahip

olunduğunda şüphe yok ki buna doğru gelen ve bundan çıkan tüm iletişim

hatlarını bu iş tüketecektir. Benzer olarak, Alan İsmi Sistemi

(Domain

Name System, DNS) milyonlarca bilgisayarın bilgisinin bakımını

yapmaktadır, bu sayede Veb sunucularının konumlandırılması için temel

servisleri biçimlendirmektedir.

Eğer bir URL ’nin çözülmesi istendiğinde, sadece ve sadece tek bir yere

yani tek bir DNS sunucusuna iletilseydi, bu işlem yüzünden Internet’in son

kullanıcı kullanımı imkansız olurdu.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Bunların yanı sıra, merkezi algoritmalar da oldukça kötü bir fikirdir.

30

Büyük bir dağıtık sistemde, muazzam sayıda mesaj bir çok hat boyunca

dolaşmaktadır.

Kuramsal bakış açısıyla baktığımızda, bu iletişimi gerçekleştirmek için

optimal yol, tüm makineler ve hatlar hakkında tam bir bilginin toplanması

ile oluşturulur ve tüm optimal rotaların hesaplanılması için bir çizge

kuramı

(graph theory) algoritması çalıştırılır. Bu bilgi daha sonra

sistem boyunca dağıtılarak yönlendirmenin iyileştirilmesine çalışılır.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Tüm girdi ve çıktı verilerinin, toplanılması ve taşınması, ağı mesajlar

yüzünden doldurmak anlamına geldiği için kötü bir fikirdir. Gerçekte,

31

herhangi bir algoritmanın tüm alanlardan bilgi toplayarak çalışmasından,

bu bilgiyi tek bir makineye işlenmek üzere göndermesinden ve daha

sonra sonuçların dağıtılmasından kaçınmak gerekir.

Sadece merkezi-olmayan

(decentralized) algoritmalar kullanmak

gerekir. Merkezi-olmayan (ademi-merkezi olan) algoritmaların, merkezi

algoritmalardan ayrıldığı bazı karakteristikleri mevcuttur:

1

-) Hiç bir makine tüm sistemin o anki durumunu bilmek zorunda değildir.

2-) Her bir makine kararlarını yerel bilgilerine göre verir.

3-) Bir makine bozulacak olursa algoritmanın çalışmasını etkilememelidir.

4-) Global bir saatin olma zorunluluğu yoktur.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Yukarıdaki, merkezi-olmayan algoritmaların karakteristiklerin

32

baktığımızda; sonuncusu daha az bariz durmaktadır fakat oldukça

önemlidir. Herhangi bir algoritma şöyle başlamış olabilir: “ Saat 12:00:00

‘da tüm makineler çıktı kuyruklarının büyüklüğünü not etmektedir ”.

Fakat bu durum hata verebilir, çünkü tüm saatler tam olarak senkronize

edilmemiştir. Daha geniş sistemler daha çok belirsizlik demektir.

Tek bir LAN üzerinde, akla yatkın bir çaba gösterilerek tüm saatler

birkaç mili saniyeye seviyesine kadar senkronize edilebilir, fakat bunun

ulusal veya uluslar arası olarak yapılması oldukça güçtür.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Coğrafik olarak ölçeklenebilirlik kendine özgü problemlere sahiptir.

Bunun ana nedenlerinden birisi, var olan dağıtık sistemleri ölçekleme’nin

33

oldukça zor olmasıdır. Bunun nedeni, yerel-alan ağları için tasarlanmış

olmaları ve senkronize iletişim

(synchronous communication)

temelli olmalarıdır. İletişimin bu biçimi, kısmi bir istek

(request)

servisidir, genellikle bir istemci

(client) olarak adlandırılır ve bir cevap

geri dönene kadar bloklanmış olarak bekler.

Bu yaklaşım genellikle LAN ‘lar üzerinde iyi çalışır, keza LAN üzerindeki

iki makine arasındaki iletişim genellikle en kötü olarak bir kaç yüz

mikrosaniye seviyesindedir. Bununla beraber, bir geniş

-alandaki

sistemde, süreçler arası iletişimi de göz önüne almamız gerekebilir, bu

iletişim yüz milisaniyeler seviyesinde olabilir. Geniş-alanlardaki

sistemlerde senkronize iletişimi kullanan etkileşimli uygulamalar büyük

bir dikkat gerektirmektedir.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Başka bir problem ise coğrafik olarak ölçeklenebilirliği engellemektedir.

Bu problem, geniş-alanda ağlarındaki iletişimin niteliği gereği

34

güvenilmez olmasıdır ve sanal olarak daima noktadan-noktaya

olmaktadır.

Bunun zıttı olarak, yerel-alan ağları genel olarak yayınlama

(broadcasting) temelli yüksek güvenirlikli iletişim olanakları

sağlamaktadırlar. Böylece dağıtık sistemin geliştirilmesi daha kolay hale

gelir. Örneğin, bir servisin yerleştirilmesi problemini göz önüne alalım.

Bir yerel-alan sisteminde, bir süreç çok basitçe her makine için bir mesaj

yayınlayabilir (broadcast), bu yolla istediği servisin çalışıp çalışmadığını

sorabilir. Sadece bu servisten sorumlu olan makinelerin her biri, cevap

mesajında (reply message) kendi ağ adreslerini de sağlarlar.

Böyle bir yerleşim şeması geniş-alandaki sistemlerde düşünülemez.

Bunun yerine, özel yerleşim servisleri tasarlanması gerekebilir. Bunlar,

milyarlarca kullanıcıya servis verme kapasitesinde ve dünya çapında

ölçeklenebilir olabilmelidirler.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Coğrafik olarak ölçeklenebilirlik, büyüklük ölçeklenebilirliğini engelleyen

merkezi çözümlere ait problemlerle güçlü bir biçimde ilişkilidir. Eğer bir

35

çok merkezi bileşeni bulunan bir sistemimiz varsa, coğrafik

ölçeklenebilirlik, geniş alan iletişiminin bir sonucu olarak başarım ve

güvenirlik problemleri nedeniyle sınırlandırılmış olabilmektedir.

Ayrıca, merkezi bileşenler ağ kaynaklarının boşa kullanılmasına neden

olmaktadır. Tek bir e-posta sunucusu’nun tüm ülke için kullanıldığı

durumu düşünelim. Bunun anlamı, bir e-postayı komşunuza yollamak

istediğinizde bunun ilk önce, oldukça uzaktaki merkezi bir posta

sunucusuna gideceğidir. Bu oldukça kötü bir durumdur.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Örneğin, bir dağıtık sistem için bir çok bileşen, aynı alan (domain)

içerisinde çalışan kullanıcılar tarafından (çoğunlukla) güvenilen tek bir

36

alan içerisinde bulunmaktadır. Bu durumlarda, sistem yönetimi, test

edilmiş ve onaylanmış uygulamalara sahip olabilmektedir ve böyle

bileşenlerin kurcalanmadığından emin olunan özel ölçüler alabilecektir.

Aslen, kullanıcılar kendi sistem yöneticilerine güvenmektedirler.

Bununla beraber, alan sınırları boyunca doğal olarak güven sürekli

değildir, yani bir noktada güvensizlik mutlaka vardır.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçeklenebilirlik problemleri: (devam)

Eğer bir dağıtık sistem diğer alanlar’a (domains) genişletilecekse, iki

tip güvenlik ölçüsü alınması gerekir. Bunlardan ilki, dağıtık sistemin yeni

37

alandan gelen istenmeyen saldırılara karşı kendini korumasıdır. Yeni

alandaki kullanıcıların, orijinal alanındaki sistemin dosya servisine

sadece okuma erişimine sahip olmaları buna örnek olarak verilebilir.

Bunlardan ikincisi, yeni alan’ın (domain) dağıtık sistemden gelen

istenmeyen saldırılardan kendi korumasıdır. Buna tipik bir örnek ise, Veb

gezgini’ne ait applet’ler gibi programların indirilmesidir. Temelde, yeni

alan böyle yabancı bir kodun çalışması ile neler olacağını beklemesi

gerektiğini bilemez ve bu kod için erişim haklarını kısıtlama yoluna gider.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri:

Bazı ölçeklenebilirlik problemleri, aklımıza bunların nasıl çözülebileceği

sorusunu da beraberinde getirmektedir. Dağıtık sistemlerdeki

38

ölçeklenebilirlik problemleri, sunucuların ve ağların sınırlı kapasitesi

nedeniyle oluşmuş başarım problemleri olarak görünmektedirler.

Buna göre, ölçeklenebilirlik için sadece üç tane bulunmaktadır. Bunlar:

iletişim gecikmelerinin gizlenilmesi, dağıtım, kopyalama’dır.

Bunlara bakmadan evvel kısaca senkron ve asenkron iletişimi

hatırlayacak olursak; Senkronize iletişimde, isteği yapan tarafın cevabı

beklediğini; Asenkron iletişimde ise isteği yapan tarafın cevap gelene

kadar daha başka faydalı bir ile iş ile uğraştığını ve cevap gelince elindeki

işe kaldığı yerden devam ettiğini söyleyebiliriz.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri:

İletişim gecikmelerinin gizlenilmesi, coğrafik ölçeklenebilirlik

durumunda karşılaşılan bir sorundur. Temel fikir oldukça basittir.

Uzaktaki servislerin istekleri için cevaplarını mümkün olduğunca çok

39

beklemekten kaçınmak. Örneğin, bir servis uzaktaki bir makineden

istenilirse, sunucudan gelecek bir cevap için bekleme süresine alternatif

olarak isteği yapan taraf daha faydalı bir işle meşgul olur. Bir cevap

geldiğinde ise, uygulama kesintiye uğratılarak, özel bir işleyici (handler)

daha önceki istekte istenilen işi tamamlamak için çağırılır.

Asenkron iletişim, genellikle yığın-işlem sistemleri (batch-processing

systems) ve paralel uygulamalarda (parallel applications)

kullanılmaktadır. Bu tarz uygulama ve sistemlerde, diğer bir görev

iletişimini tamamlarken daha az veya daha çok olmak üzere bağımsız

görevler (tasks) çalıştırılmak için tarifelendirilir (scheduled). Alternatif

olarak, kontrole ait yeni bir iş parçacığı

(thread) [veya buna bazı

kaynak kitaplarda “kanal” da denilmektedir] isteği yerine getirmek üzere

çalıştırılabilir. Bununla beraber, bu yeni iş parçacığı cevap için

bloklanarak beklerken, süreç içerisindeki diğer iş parçacıkları ise

çalışmalarına devam ederler.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri: (devam)

Bununla birlikte, bir çok uygulama asenkron iletişimi etkin olarak

kullanamamaktadır. Örneğin, etkileşimli uygulamalarda bir kullanıcı bir

40

isteği gönderdiğinde, cevabı beklemekten başka yapacak daha iyi bir şeyi

yoktur. Böyle durumlarda, daha iyi bir çözüm toplam iletişimi

düşürmektir, örneğin normal olarak sunucuda yapılan hesaplamanın bir

kısmını servisi isteyen istemci sürece taşımak gibi. Bu yaklaşımın çalıştığı

tipik bir durum formları kullanan erişim veritabanları durumudur.

Normal olarak, formun doldurulması, her bir form alanı (form field) için

ayrı bir mesajın gönderilmesi ile yapılmakta ve sunucudan bir onay

(acknowledgement) beklenilmektedir.

Örneğin, sunucu sözdizimsel (syntactic) hataları bir girdiyi (entry) kabul

etmeden evvel kontrol edebilir. Daha iyi bir çözüm ise formda doldurmak

için kodu da istemciye göndermektir, burada girdilerin kontrol edilmesi

olasıdır, istemci sonuçta tamamlanmış bir formu geri döndürür. Bu

“kodun gönderilmesi” (shipping code) yaklaşımı, Java applet’leri

biçiminde Veb tarafından desteklenmektedir.

Ölçekleme teknikleri (1)

41

1.4

Teslimat farklılıkları:

a) Bir sunucu veya

b) Bir istemcinin Internet üzersindeki var olan formların doldurulup

doldurulmadı

ğını kontrol etmesi.

Fig. 1.4

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri: (devam)

Diğer bir ölçekleme tekniği ise dağıtım

(distribution) ‘dır. Dağıtım bir

bileşenin alınmasını, daha küçük parçalara bölünmesi ve sonrasında bu

42

parçaların sistem boyunca yayılmasını içermektedir. Dağıtım için güzel

bir örnek olarak, Internet Alan İsmi Sistemi

(Domain Name

System, DNS) verilebilir.

DNS isim uzayı

(name space), alanlar’ın (domains) bir ağacı

şeklinde hiyerarşik yapıda organize edilmiştir, bunlar birbirleriyle

örtüşmeyen bölgeler’e (zones) bölünmüşlerdir. Her bir bölgedeki

isimler tek bir isim sunucusu tarafından idare edilmektedir. Olayı çok

detaylandırmadan bahsedecek olursak; her bir yol (path) ismi,

Internet’teki bir konakçı’nın (host) ismi olmaktadır. Buna örnek olarak,

“ce.comu.edu.tr” adres örneğine bakacak olursak, karşımızda bir

sunucu makineyi görürüz.

Scaling Techniques (2)

Fig. 1.5

43

1.5

Domain Name System (DNS) isim uzayı’nın bölgelere bölünmesi örne

ği.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri: (devam)

Ölçekleme problemlerinin sıklıkla başarım düşüşü şeklinde oluştuğuna

dikkat etmek gerekir. Genellikle, bir dağıtık sistem boyunca bileşenleri

kopyalama

(replicate) oldukça iyi bir fikirdir. Kopyalama, sadece

44

kullanılırlığı

(availability) artırmaz, fakat daha iyi bir başarım elde

edilmesi için bileşenler arasındaki yükün dengelenmesine yardımcı olur.

Böylece, coğrafik olarak geniş çapta dağıtık sistemlerde, daha önce

bahsettiğimiz iletişim gecikmesinden daha fazla gizleme (yani buna maruz

kalmama şansı) yakın bir kopyaya sahip olunması ile mümkün

olabilmektedir.

Önbellekleme

(caching), kopyalamanın özel bir biçimidir. İkisi arasında

ayrım yapmak çoğu zaman oldukça zordur. Kopyalama durumunda,

genelde bu kaynağa erişen istemcinin yakınlığında, önbellekleme sonuçları

bir kaynağın bir kopyasının oluşturulması ile sonuçlanır. Bununla beraber,

kopyalamanın zıttı olarak, önbellekleme bir kaynağın istemcisi için

istemciye ait bir karardır, fakat bu karar kaynağın sahibi tarafından

verilmemektedir.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri: (devam)

Önbellekleme ve kopyalama, ölçeklenebilirlik üzerinde oldukça sorun

oluşturabilmektedir. Bir kaynağın birden çok kopyasına sahipsek, bir

kopyanın değiştirilmesi bu kopyayı diğerlerinden farklı bir kopya yapar.

45

Sonuçta, önbellekleme ve kopyalama, tutarlılık

(consistency)

problemlerine neden olmaktadır.

Bir kaynağın kullanımına fazlaca bağımlı olarak tutarsızlıklar’a

(incosistencies) neden olan sebepler katlanılabilir hale getirilebilir.

Örneğin, bir çok Veb kullanıcısı Internet gezginleri’nin (browser) son

birkaç dakika için geçerliliğinin kontrol edilmediği önbelleklenmiş bir

dokümanı geri döndürmesin (onlara göstermesini) kabul edebilirler.

Bununla beraber, elektronik borsa alışverişleri gibi, güçlü bir tutarlılığın

garanti edilmesi gerektiği bir çok durumda mevcuttur. Güçlü

tutarlılığın

(strong consistency) için problem ise, bir güncellemenin

tüm diğer kopyalara hemen yayılması (propagated) gerekmesidir. Ayrıca,

eğer iki güncelleme eşzamanlı

(concurrent) olursa, her bir kopyanın

aynı sırada güncellenmesi sıklıkla gerekmektedir.

Dağıtık sistemin amaçları

4-) Ölçeklenebilirlik (Scalability)

Ölçekleme teknikleri: (devam)

Böyle durumlarda, genel senkronizasyon mekanizması

(global

synchronization mechanism) kullanmak gerekebilir. Ne yazık ki,

böyle mekanizmalar, ölçeklendirme için son derece zor ve neredeyse

46

imkansızdırlar. Sonuçta, kopyalama yoluyla ölçeklendirme diğer

ölçeklendirilemez çözümleri ortaya çıkartır.

Kavramlar

Donanım kavramları

47

Donanım kavramları

• Tüm dağıtık sistemler, birden çok CPU ‘ya sahiptirler. Böyle durumlarda

donanımın organize edilebilmesi için birkaç farklı yol mevcuttur, özellikle

nasıl birbirleri ile ara bağlantı ve iletişim kurdukları konuları ortaya çıkar.

Birden çok CPU’ya sahip bilgisayar sistemleri için çeşitli sınıflandırma

şemaları yıllar boyunca önerilmiştir, fakat bunlardan hiç biri gerçekten

48

genel topluluk tarafından kullanılır hale gelmemiş ve geniş çapta adapte

edilmemişlerdir.

Amacımız dahilinde, bu ders kapsamında sadece bağımsız bilgisayarlar

topluluğu şeklinde oluşturulmuş dağıtık sistemleri göz önüne alalım.

Şekil 1.6 ‘da tüm bilgisayarlar iki ana gruba bölünmüştür: Çoklu işlemcili

bilgisayarlar (multiprocessors) ve çoklu bilgisayarlar

(multicomputers). Buna göre, eğer paylaşımlı bellek kullanılıyor ise,

buna çoklu işlemcili bilgisayarlar, eğer paylaşımlı bellek genellikle

kullanılmıyor ise buna çoklu bilgisayarlar denilmektedir. Temel fark

aslında bu durumdur. Çoklu işlemciye sahip bilgisayarlar, tüm CPU ‘lar

tarafından paylaşılan tek bir fiziksel adres uzayı’nı (address space)

paylaşmaktadırlar. Tüm makineler aynı belleği paylaşmış olurlar.

Fig. 1.6

49

Dağıtık bilgisayar sistemlerinde farklı temel organizasyonlar ve bellekler.

Donanım kavramları

• Çoklu işlemcili bilgisayarların aksine, bir çoklu bilgisayar (multicomputer)

için her bir makine kendi özel belleğine sahiptir. Bu duruma örnek olarak,

kişisel bilgisayarların bir ağ üzerinden birbirlerine bağlanması ile oluşan

çoklu bilgisayar verilebilir. Bu kategorilerin her biri, daha sonra ara

bağlantılı ağ mimarisi temelli olarak bölümlenirler. Şekil 1.6 ‘da iki kategori

50

gösterilmiştir. Bunlar: Veriyolu

(Bus) ve Anahtarlamalı

(Switched)

olarak verilmişlerdir.

Veriyolu ile tek bir ağın olduğu, arka yüz’ün (backplane), yeriyolu ve

kabloların veya tüm makineleri biribirine bağlayan diğer ortamların

bulunduğunu ifade etmiş oluruz. Kablolu televizyon buna benzer bir şema

kullanmaktadır. Kablolu TV şirketi bütün sokağa bir hat döşer ve tüm

aboneler kendilerine ait priz (kutu) aracılığı ile televizyon hizmetini alırlar.

Anahtarlamalı sistemler, kablolu televizyonlar gibi tek bir omurga’ya

(backbone) sahip değillerdir. Bunun yerine, makineden makineye bir çok

farklı örüntü

(pattern) kullanımı ile bireysel kablolama yapılmıştır.

Mesajlar, kablolardan rota boyunca her adımda anahtarlama kararı

verilerek akmaktadır. Buna benzer olarak dünya çapında telefon

sistemlerinin organizasyonu verilebilir.

Donanım kavramları

• Dağıtık bilgisayar sistemleri arasında bir ayrım yapmak istendiğinde,

bunlar homojen ve heterojen olarak ayrıştırılabilir. Bu ayrım, çoklu

bilgisayarlar için oldukça kullanışlıdır. Homojen bir çoklu bilgisayarda,

sadece tek bir ara bağlantı ağı bulunmaktadır ve bu ağ her yerde aynı

teknolojiyi kullanmaktadır. Benzer olarak, tüm işlemciler aynıdır ve genelde

51

aynı miktar özel belleğe erişirler. Homojen çoklu bilgisayarlar, çoklu

işlemcili olanlara benzer olarak daha çok paralel sistemler olarak (aynı

problem üzerinde çalışırlar) kullanılırlar.

Heterojen bir çoklu bilgisayar sistemi, farklı çeşitte ve bağımsız

bilgisayarlar içerebilmektedir. Örneğin, dağıtık bir bilgisayar sistemi, farklı

yerel-alan bilgisayar ağlarının bir topluluğu olarak oluşturulmuş olabilir. Bu

ağlar birbirlerine FDDI (Fiber Distributed Data Interface) veya ATM

(Asynchronous Transfer Mode) anahtarlamalı bir omurga ile ara

bağlantılandırılmışlardır.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

• Çoklu bilgisayar sistemlerinin hepsi tek bir anahtar özelliği paylaşmaktadır:

Tüm CPU ’lar paylaşımlı belleğe doğrudan erişime sahiptirler. Veriyolutemelli çoklu bilgisayarlar , belirli sayıda CPU ‘yu içerecek şekilde

oluşmaktadır, tümü genel bir veriyolu’na ve bir bellek modülüne

bağlanmıştır. Basit bir düzenleşim (veya konfigürasyon, “configuration”),

52

yüksek-hızlı bir arka yüz’e ve ana karta sahiptir. Bu arkayüz veya anakarta

CPU ve bellek kartları yerleştirilmiştir.

Sadece bir tane bellek olduğu için, eğer A isimli CPU belleğe bir kelime

(word) yazarsa ve daha sonra B isimli CPU bu kelimeyi birkaç milisaniye

sonra okursa, B sadece yazılmış olan veriyi alır. Bu özelliğe sahip bir bellek

için tutarlı

(coherent) denilir. Bu şemanın problemi,

4 veya

5 tane CPU

gibi az işlemcinin bulunduğu durumlarda, veriyolu’nun genellikle aşırı

yüklendiği

(overloaded) ve başarımın büyük ölçüde düştüğü durumdur.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

• Buna çözüm olarak, CPU ile veriyolu arasına yüksek-hızlı bir önbellek

(cache memory) eklemek verilebilir, bunu Şekil 1.7 ‘den görebilirsiniz.

Önbellek en güncel erişilen kelimeleri (words) tutmaktadır. Tüm bellek,

(gelen veriler için) önbellekten geçilmesini istemektedir. Eğer istenilen

kelime (word) önbellekte bulunuyor ise, önbellek CPU ‘ya kendisi cevap

53

verir ve hiçbir veriyolu isteği yapılmaz.

Eğer önbellek yeterince büyükse, hit rate

(başarı oranı) denilen başarı

olasılığı, sistemde daha çok CPU olmasına izin verildiğinde, bu oran yüksek

olabilir ve CPU başına veriyolu trafiği miktarı büyük oranda düşebilir. Hit

rate ‘ler %90 veya daha fazla olabilmektedir. Önbellekler ise 512 KB ile

1 MB

arasında (veya daha fazla) değişebilmektedir.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

Önbelleklerin ortaya çıkması ciddi bir problemi de beraberinde getirmiştir.

A ve B isimli iki CPU ‘nun bulunduğu durumu ele alalım. Her biri kendisine

ait önbelleleğinden aynı kelimeyi okumaktadır. Daha sonra A isimli CPU

kelimenin (kendi önbelleğinde) üzerine yazsın. B daha sonra kendi

önbelleğindeki kelimeyi okuduğunda, A isimli CPU ‘nun yazdığını değil,

54

kendi önbelleğindeki eski değeri elde etmiş olur. Bellek şimdi tutarlı

değildir (incoherent) ve sistemin programlanması oldukça zordur.

Önbellekleme, dağıtık sistemlerde yaygın bir biçimde kullanılmaktadır.

Veriyolu-temelli çoklu bilgisayarların problemi, önbellekleri kullanırken

onlara ait sınırlı ölçeklenebilirlik’leridir. 256 işlemciden daha fazla çoklu

işlemcili bir bilgisayar oluşturmak için, bellek ile CPU’ları birbirine

bağlamak için farklı bir yöntem gerekmektedir.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

Şekil 1.8 ‘de görüldüğü gibi, bir olasılık, belleği modüllere bölerek, bunları

bir çapraz anahtar (crossbar switch) ile CPU‘lara bağlamaktır. Her bir CPU

ve her bir bellek kendi ile birlikte gelen bir bağlantıya sahiptir. Her ara

kesit, küçük bir elektronik üstgeçit anahtarı

(crosspoint switch)

55

kesit, küçük bir elektronik üstgeçit anahtarı

(crosspoint switch)

şeklindedir. Bu anahtar donanımda açılabilir ve kapanabilir.

Bir CPU belirli bir belleğe erişmek istediğinde, ona bağlı üstgeçit anahtarı

kısa bir süre için kapanarak erişimin olmasına izin verir. Çapraz anahtarın

etkisinde dolayı, bir çok CPU aynı zamanda belleğe erişebilir. Bununla

beraber, eğer iki CPU aynı belleğe eş anlı

(simultaneously) erişmeyi

denerse, bunlardan bir tanesinin beklemesi gerekir.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

• Bir çapraz anahtarın altında,

n adet CPU ve

n adet bellek bulunmaktadır, bu

nedenle adet üstgeçit anahtarı gerekmektedir. Büyük bir

n değeri için,

bu sayı engelleyici

(prohibitive) olabilmektedir. Bunun bir sonucu

olarak, insanlar daha az anahtarın kullanılacağı alternatif anahtarlama

ağları bulmaya ve oluşturmaya çalışmaktadırlar.

56

2

n

Omega anahtarlama ağı, Şekil 1.8 ‘de gösterilmektedir. Bu ağ dört tane 2 x 2

anahtar içerir. Bunların her biri, iki girdi ve iki çıktıya sahiptir. Her bir

anahtar ya girdi yönünü yada çıktı yönünü yönetebilir. Şekil 1.8 ‘e dikkatlice

bakıldığında, her bir CPU ‘nun her bir belleğe erişiminin bulunduğunu

görebiliriz.

Anahtarlama ağlarının olumsuzluğu, CPU ve bellek arasında var olan birkaç

anahtarlama adımıdır. Sonuçta, CPU ile bellek arasında düşük gecikmenin

olabilmesinden emin olmak için, anahtarlamanın ucuz olmayacak bir

şekilde oldukça hızlı olması gerekmektedir.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

Fig. 1.7.

57

Veriyolu (Bus) temelli çoklu işlemci.

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

Fig. 1.8

58

a) Bir Çapraz anahtar (Crossbar switch)

b) Bir Omega anahtarlama ağı (omega switching network)

Donanım kavramları – Çoklu Đşlemciler (Multiprocessors)

• İnsanlar, hiyerarşik sistemlere geçerek anahtarlama maliyetini azaltmayı

denemişlerdir. Bazı bellekler her bir CPU ile ilişkilendirilmiştir. Her bir

CPU kendi yerel belleğine hızlı bir biçimde erişebilir, fakat başkasının

belleğine erişimi oldukça yavaştır. Bu tasarım, Tekdüze olmayan Bellek

Erişimi ‘ni (NonUniform Memory Access, NUMA) ortaya çıkartır.

59

NUMA makineleri, omega ağları temelli makinelerden daha iyi bir ortalama

erişim süresine sahiptirler. Bunlar yeni bir olumsuzluğa sahiptir, buna göre

veri ve programların yerleştirilmesi düzgünce yerel belleğe giden bir çok

erişimin yapılmasında kritik bir hale gelmektedir.

Donanım kavramları – Homojen Çoklu Bilgisayar

Sistemleri (Homogeneous Multicomputer Systems)

• Çoklu işlemcilerin aksine, bir çoklu bilgisayar’ın oluşturulması görece

kolaydır. Her bir CPU kendi yerel belleğine doğrudan bağlıdır. Burada kalan

tek sorun, CPU ‘ların birbirleri ile nasıl iletişime geçeceğidir. Açıkça, bazı

60

ara bağlantılar için şema gerekmektedir, fakat bu şema sadece CPU-ile-CPU

arasındaki iletişim içindir. Trafik hacmi, CPU-ile-bellek arasındaki trafik

için kullanılan ara bağlantı ağı’nınkinden birkaç mertebe daha düşük

boyutlardadır.

Homojen çoklu bilgisayarlara göz atalım. Sistem Alan Ağları

(System

Area Networks, SANs) diye adlandırılan bu sistemlerde, düğümler

(nodes) büyük bir dolaba (big rack) monte edilmişlerdir ve sıklıkla yüksekbaşarımlı, tek bir ara bağlantı ağına bağlanmışlardır. Önceki gibi, veriyolu

(bus) ve anahtar (switch) temelli sistemler arasında bir ayrım yapmamız

geremektedir.

Donanım kavramları – Homojen Çoklu Bilgisayar

Sistemleri (Homogeneous Multicomputer Systems)

• Veriyolu temelli çoklu bilgisayarda, süreçler paylaşımlı bir çoklu erişim ağı

(multiaccess network) boyunca bağlanmışlardır. Ağ bantgenişliği, tipik

olarak 100 Mbps olmaktadır. Veriyolu temelli çoklu işlemcilerde olduğu

gibi, veriyolu temelli çoklu bilgisayarlar da sınırlı ölçeklenebilirliğe sahiptir.

61

Düğümlerin iletişime gerçekten ihtiyaçlarının ne kadar olduğuna bağlı

olarak, 25 ilâ 100 düğümden daha fazla düğümü bulunan sistemlerden çok

fazla başarım beklememek gerekmektedir.

Anahtar temelli çoklu bilgisayar da, mesajlar işlemciler arasında, veriyolu

temelli sistemlerdeki gibi yayınlamanın yerine ara bağlantılı bir ağ

üzerinden yönlendirilmişler

(routed) olurlar.

İki popüler topoloji olarak örgü

(mesh) ve hiperküp

(hypercube)

verilebilir. Bunlar Şekil 1.9 ‘dan görülebilirler.

Donanım kavramları – Homojen Çoklu Bilgisayar

Sistemleri (Homogeneous Multicomputer Systems)

Fig. 1.9

62

a) Grid

b) Hypercube

Donanım kavramları – Homojen Çoklu Bilgisayar

Sistemleri (Homogeneous Multicomputer Systems)

• Bir hiperküp,

n-boyutlu bir küptür. Şekil 1.9 (b) ‘deki hiperküp ise dörtboyutludur. Bunlar iki tane sıradan küp olarak düşünülebilir, her birinin 8

adet köşesi

(vertices) ve 12 tane de kenarı

(edges) bulunmaktadır. Her

bir köşe CPU ‘yu göstermektedir.

63

Her bir kenar ise iki CPU arasındaki bir bağlantıyı göstermektedir. İki

küpteki ilgili köşeler bağlantılanmışlardır. Hiperküpü beş boyuta

genişletmek için, iki tane daha ara bağlantılı küpü eklemek gerekecektir. Bu

böylece boyut arttıkça devam edecektir.

Ayrıca, Kitlesel Paralel İşlemciler

(Massively Parallel Processors,

MPPs) de bulunmaktadır. Bunlar oldukça büyük ve çok çok pahalı süper

bilgisayarlardır ve binlerce CPU içermektedirler. Bir çok durumda, CPU ‘lar

iş istasyonu veya PC ‘ler de kullanılanlardan farklı olmamaktadır.

Donanım kavramları – Homojen Çoklu Bilgisayar

Sistemleri (Homogeneous Multicomputer Systems)

• Böyle bir bilgisayar teknolojisi skalasında, anahtarlanmış çoklu

bilgisayarlar’ın popüler bir biöi de İş istasyonları Kümesi

(Clusters of

Workstations, COWs) olmaktadır. Bunlar temelde, kullanıma-hazır (offthe-shelf) iletişim bileşenleri (örneğin Myrinet kartları) üzerinden birbirine

bağlanmış standart PC ‘ler veya iş istasyonlarının bir topluluğudur. Aradaki

64

bağlantı ağı COW ‘ları MPP ‘ler den ayırmaktadır. Böylece, yüksek

Girdi/Çıktı bantgenişliğinden emin olmak veya sistem hatalarına karşı

korunmak için genelde özel ölçüler alınmaz. Büyük çoğunlukla, bu yaklaşım

COW ’ları basit ve ucuz yapar.

Bir çok dağıtık sistem, günümüzde heterojen çoklu bilgisayarlar

kullanılarak oluşturulmaktadır. Bunun anlamı, sistemin parçalarını

oluşturan bilgisayarların geniş çapta çeşitlilik göstermesidir. Bu çeşitlilik

işlemci tipleri, bellek büyüklükleri ve Girdi/Çıktı bantgenişlikleri gibi

olgulara bağlıdır.

Bilgisayarlar, çoklu işlemcili veya homojen çoklu bilgisayarlar gibi yüksek

başarımlı paralel sistemler olabilmektedirler.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

65

Ders’in sonu…

66

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 3

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 3: Yazılım kavramları,

İstemci-Sunucu modeli.

Hatırlatma

• Simultaneous = Eşanlı

• Concurrent = Eşzamanlı

3

• Read-only = Salt-okunur.

Yazılım kavramları (Software concepts)

• Dağıtık sistemler için donanım önemlidir, fakat yazılım dağıtık bir sistemin

gerçekte nasıl göründüğünü belirlemektedir. Dağıtık sistemler geleneksel

işletim sistemlerine oldukça benzemektedirler. İlk olarak, bunlar altta yatan

donanım

(underlying hardware) için kaynak yöneticileri

(resource

4

donanım

(underlying hardware) için kaynak yöneticileri

(resource

managers) gibi davranmaktadırlar.

• Bu sayede, birden çok kullanıcı ve uygulamalara CPU’lar, bellekler, çevre

cihazları, ağ ve her çeşit veri şeklindeki kaynakların paylaştırılmasına imkan

tanımış olmaktadırlar. İkinci olarak, belki de en önemlisi olarak, bir dağıtık

sistemin uygulamaların kolaylıkla çalıştırılabilmesi için bir sanal makine

(virtual machine) sağlayarak altta yatan donanımın heterojen doğasını ve

karmaşıklıkları

(intricacies) gizlemeye çalışmasıdır

Yazılım kavramları

• Dağıtık sistemlerin doğasını anlamak için, dağıtık bilgisayarlar ile işletim

sistemlerinin ilişkisine yakından bakalım. Dağıtık sistemler için işletim

sistemleri iki kategoriye bölünmüştür:

Sıkı

-bağlı sistemler

(tightly

-coupled systems) ve gevşek

-bağlı

5

Sıkı

-bağlı sistemler

(tightly

-coupled systems) ve gevşek

-bağlı

sistemler

(loosely-coupled systems).

• Sıkı-bağlı sistemlerde, işletim sistemi temelde yönettiği kaynakların genel ve

tek bir görünümünün bakımını yapar. Gevşek-bağlı sistemler, her biri kendi

işletim sistemini çalıştıran bilgisayarların bir topluluğu olarak düşünülebilir.

Bununla beraber, bu işletim sistemleri bir arada çalışarak kendi servislerini ve

kaynaklarını diğerleri için kullanılır hale getirirler.

Yazılım kavramları

• Sıkı-bağlı ve gevşek-bağlı sistemler arasındaki bu ayrışım, donanım

sınıflandırması ile ilişkilidir. Sıkı-bağlı bir işletim sistemi genellikle bir dağıtık

işletim sistemi

(distributed operating system, DOS) olarak bilinir ve

çoklu süreçlerin ve homojen çoklu bilgisayarların yönetilmesinde

kullanılmaktadır. Geleneksel tek işlemcili işletim sistemleri gibi, bir dağıtık

6

sistemin ana hedefi, birden çok süreç tarafından paylaşılabilen altta yatan

donanımın karmaşıklıklarının (intricacies) gizlenilmesidir.

• Gevşek-bağlı bir ağ işletim sistemi

(network operating system, NOS)

heterojen çoklu bilgisayar sistemleri için kullanılır. Bununla beraber, altta

yatan donanım, bir ağ işletim sistemi için önemli bir konudur. Geleneksel

işletim sistemlerinin doğal bir sonucu olarak gelen yerel servisler uzak

istemciler için kullanılır hale getirilmişlerdir.

• Ağ işletim sistemlerinin servislerinin geliştirilmesi, dağıtım saydamlığının

sağlanması için daha iyi bir destek olarak gerekmiştir. Bu gelişmeler, ara

katman yazılımı

(middleware) olarak bilinen ve modern dağıtık sistemlerin

kalbi niteliğindeki yapıyı oluşturmuştur. Ara katman yazılımı, DOS ve NOS

karşılaştırması aşağıdaki tabloda verilemektedir.

Yazılım kavramları – Dağıtık Đşletim Sistemleri

(Distributed Operating Systems)

System Description Main Goal

Çoklu işlmeciler ve homojen çoklu bilgisayarlar Donanım

7

DOS

Çoklu işlmeciler ve homojen çoklu bilgisayarlar

için Sıkı-bağlı

(Tightly-coupled) işletim

sistemidir

Donanım

kaynaklarının

gizlenilmesi ve

yönetilmesi

NOS

Heterojen çoklu bilgisayarlar (LAN ve WAN

üzerinden) Gevşek-bağlı

(Loosely-coupled)

işletim sistemidir

Uzak istemciler için

yerel servislerin

önerilmesi

Middleware NOS ‘un üzerine ek olarak gerçekleştirilen genel amaçlı servislerdir. Dağıtım saydamlığının sağlanması

• Bu tabo ile şu üç sistem arasında karşılaştırma yapılmaktadır:

• Dağıtık İşletim Sistemi, DOS (Distributed Operating Systems)

• Ağ İşletim Sistemi, NOS (Network Operating Systems)

• Ara katman yazılımı, Middleware

sağlanması

Yazılım kavramları – Dağıtık Đşletim Sistemleri

(Distributed Operating Systems)

• İki tip dağıtık işletim sistemi vardır. Bir çoklu işlemci işletim sistemi

(multiprocessor operating system) bir çoklu işlemcili bilgisayarın

kaynaklarını yönetmektedir. Bir çoklu bilgisayar işletim sistemi

(multicomputer operating system) homojen çoklu bilgisayarlar için

8

üretilmiş bir işletim sistemidir.

• Dağıtık işletim sistemlerinin fonksiyonelliği temelde tek işlemcili

(uniprocessor) geleneksel işletim sistemleri ile aynıdır. Tek fark dağıtık işletim

sistemleri birden çok CPU ile uğraşmaktadır.

• İşletim sistemleri, geleneksel olarak sadece tek bir CPU’yu yönetmek üzere

oluşturulmuşlardır. Bu sistemlerin ana hedefi, kullanıcı ve uygulamalara, CPU,

ana bellek, diskler ve çevre cihazları gibi kaynakların kolay yoldan paylaşımını

sağlamaktır.

• Kaynakların paylaşımı, farklı uygulamaların, izole edilmiş bir biçimde bulunan

aynı donanımı kullanabilmeleridir. Bir uygulama için, kendi kaynaklarına sahip

olduğunda ve aynı sistem üzerinde aynı zamanda birkaç uygulama

çalıştırılabildiğinde, her birinin kendi kaynak kümesine sahip olması gibi

görülebilir. Bu açıdan, işletim sistemi bir sanal makine gerçekleştirmiş ve

uygulamalara çoklu iş imkanı (multitasking facilities) sağlamış olmaktadır.

Yazılım kavramları – Tek işlemcili Đşletim

Sistemleri (Uniprocessor Operating Systems)

• Böyle sanal makinelerde kaynakların paylaşımı’nın önemli bir cephesi,

uygulamaların birbirlerinden korunmasıdır. Örneğin, iki tane bağımsız

uygulama olan A ve B ‘nin aynı zamanda çalıştırılması kabul edilemez olduğu

durumda, A uygulaması verinin şu an depolandığı ana bellekteki parçaya

9

basitçe erişerek B uygulamasına ait veriyi değiştirebilir. Benzer olarak,

uygulamaların sadece işletim sistemi tarafından önerilen imkanları

kullandığından emin olmamız gerekir.

• Örneğin, bir uygulamanın mesajlarını doğrudan bir ağ kartına

kopyalayabilmesini engellemek gerekmektedir.Bunun yerine işletim sistemi,

iletişim için bazı ilkelleri (primitives) sağlamakta ve böylece farklı makineler

üzerindeki uygulamalar arasında mesajların gönderilip

-alınması mümkün

olabilmektedir.

Yazılım kavramları – Tek işlemcili Đşletim

Sistemleri (Uniprocessor Operating Systems)

• Sonuçta, işletim sistemi donanım kaynaklarının nasıl kullanıldığı ve

paylaşıldığını tam kontrol edebilmelidir. Böylece, bir çok CPU en azından iki

işletim modu’nu desteklemektedir. Çekirdek modu’nda (kernel mod), tüm

komutlar’a (instructions) çalıştırılmak üzere izin verilmiştir ve tüm bellek

10

komutlar’a (instructions) çalıştırılmak üzere izin verilmiştir ve tüm bellek

ve tüm kayıtçıların (registers) topluluğu, çalıştırılma boyunca erişilebilirdir.

• Aksine, kullanıcı modu’nda (user mode), bellek ve kayıtçı erişimi

kısıtlanmıştır. Örneğin, bir uygulamanın (işletim sistemi tarafından

ayarlanmış) bir adres aralığının dışında kalan bellek yerleşimlerine erişimine

izin verilmemiş veya cihaz kayıtçıları’na (device registers) doğrudan

erişimine izin verilmemiş olabilir.

• İşletim sisteminin kodu çalıştırılırken, CPU’nun kendisi çekirdek modu’na

(kernel mode) anahtarlanır (değiştirilerek çevrilir). Bununla birlikte, kullanıcı

mod’undan çekirdek modu’na değiştirimin tek yolu işletim sistemleri

tarafından gerçekleştirilen sistem çağrıları’dır (system calls). Çünkü sistem

çağrıları, bir işletim sisteminin önerdiği sadece temel servislerdir. Donanımın

belleği ve kayıtçı erişimini kısıtlaması nedeniyle, bir işletim sistemi tam

kontrolü ele alabilmektedir.

Yazılım kavramları – Tek işlemcili Đşletim

Sistemleri (Uniprocessor Operating Systems)

• İşletimin iki modu’nun bulunması, tüm işletim sistemi kodu’nun çekirdek

modu’nda çalıştırıldığı işletim sistemleri organizasyonuna neden olmuştur.

Sonuçta, sıklıkla büyük, yekpare

(monolithic) bir program elde edilir ve bu

program tek bir adres uzayı üzerinde çalışmaktadır.

11

• Bu yaklaşımın sorunu, sistemin uyarlanabilmesinin sıklıkla çok zor olmasıdır.

Diğer bir deyişle, işletim sistemi bileşenlerinin tam bir kapatma

(shutdown)

olmadan ve olası bir tam tekrar derleme

(recompilation) ve tekrar

kurma

(reinstallation) olmadan, değiştirilmesi veya uyarlanabilmesinin

oldukça zor olmasıdır.

• Yekpare işletim sistemleri, açıklık bakış açısı, yazılım mühendisliği, güvenirlik

ve bakımı yapılabilirlik

(maintainability) için kötü bir fikir olarak

durmaktadır.

Yazılım kavramları – Tek işlemcili Đşletim

Sistemleri (Uniprocessor Operating Systems)

• İşletim sistemini organize etmek için daha fazla esnek bir yaklaşım ise onu iki

parça şeklinde organize etmektir. İlk parça, donanım yönetimi için modüllerin

bir topluluğundan oluşmaktadır. Fakat bu modüller, kullanıcı modu’nda eşit

olarak çalıştırılırlar. Örneğin, bellek yönetimi temelde belleğin hangi

12

parçalarının süreçler için tahsis edildiğinin ve hangi parçalarının boş

olduğunun takip edilmesini içermektedir. Çekirdek modu’nda çalıştırmanın

gerektiği zaman, sadece bellek yönetim birimi’ne (memory management

unit, MMU) ait kayıtçıların ayarlandığı zamandır.

• İşletim sisteminin ikinci parçası, çekirdek modu’nda çalıştırılması gereken

kodun sadece küçük bir mikroçekirdek tarafından içerildiği durumdur. Pratikte,

mikroçekirdek; cihaz kayıtçılarının ayarlanması, süreçler arasında CPU ‘nun

değiştirilmesi, MMU ‘nun idare edilmesi ve donanım kesmeleri’nin

(hardware interrupts) yakalanması için gerekli kodun içerilmesine sadece

ihtiyaç duyar.

• Ayrıca, sistem çağrılarının uygun kullanıcı-seviyesi işletim sistemi modüllerinin

çağrılarına geçirilmesi için gerekli kodu ve bunların sonuçlarının

döndürülmesini de içermektedir.

• Bu yaklaşıma ait organizasyon Şekil 1.11 ‘de gösterilmektedir.

Yazılım kavramları – Tek işlemcili Đşletim

Sistemleri (Uniprocessor Operating Systems)

Fig. 1.11

13

Şekil 1.11 ‘de bir mikroçekirdek (microkernel) aracılığıyla uygulamaların

işletim sistemi kodundan ayrılması görülmektedir.

Yazılım kavramları – Tek işlemcili Đşletim

Sistemleri (Uniprocessor Operating Systems)

• Mikro çekirdek kullanımını bir çok yararı vardır. Bunlardan en önemlilerinden

birisi de esnekliktir. Bunun nedeni işletim sisteminin büyük bir bölümü’nün

kullanıcı modu’nda çalıştırılmasıdır. Bu durum, bir modülün tekrar derlenmesi

veya tüm sisteme tekrar kurulmasına gerek kalmadan değiştirilebilmesi

14

(replace) için görece kolaylık sağlamaktadır. Başka bir önemli konu ise

kullanıcı-seviyesi modüllerinin prensipte farklı makinelerde yerleştirilmiş

olabileceğidir.

• Örneğin, bir dosya yönetim modülünü farklı bir makineye kolaylıkla bir dizin

servisi’ni (directory service) yöneten modül yerine yerleştirebiliriz. Diğer

bir deyişle, mikro çekirdek yaklaşımı bir tek işlemcili (uniprocessor) işletim

sistemini dağıtık bilgisayarlara kendiliğinden genişletmiş olmaktadır.

• Mikro çekirdekler iki önemli dezavantaja sahiptirler. İlki, şu anki işletim

sistemlerinin çalıştığı şekilden farklı olmalarıdır ve kitlesel direnişle (massive

resistance) ile daima karşılaşan kolay kolay değişmeyen herhangi bir statüko’yu

(status quo) değiştirmeye çalışmalarıdır. (buna örnek olarak şu cümle

söylenmektedir: “Bu işletim sistemi dedem için yeterince iyiyse benim için de

yeterince iyidir”).

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• İkinci olarak, mikro çekirdekler ek bir işletime sahiptirler ve böylece belirli bir

başarım kaybı’na (performance loss) neden olurlar. Bununla beraber ne

kadar modern ve hızlı CPU ‘ya sahip olursak olalım, %20 ‘lik başarım kaybı

ölümcül olabilmektedir.

• Tek işlemcili işletim sistemleri için, başka bir önemli fakat sıklıkla tam olarak

15

açıkça ortada olmayan bir uzantı

(extension) ise, çoklu işlemcilerin

paylaşımlı bir belleğe erişebilmelerine destek vermektir. Kavramsal olarak,

uzantı oldukça basittir, bunda donanımı yönetmek için işletim sistemi

tarafından tüm veri yapılarına ihtiyaç duyulur. Bu donanıma CPU da dahildir

ve veriler paylaşımlı bellekte tutulur. Bu verilerin ana farkı, şimdi birden çok

işlemci tarafından erişilebiliyor olmalarıdır. Ayrıca tutarlılığın garanti edilmesi

için eşzamanlı

(concurrent) erişimden korunmaktadırlar.

• Bununla beraber, bir çok işletim sistemi, özellikle PC’ler ve iş istasyonları için

olanlar, kolaylıkla birden çok CPU ‘yu idare edemezler. Bunun ana nedeni,

bunların yekpare

(monolithic) olarak tasarlanmış olmalarıdır. Bu mantıkla,

sadece tek bir kontrol iş parçacığı ile çalıştırılabilmektedirler. Böyle işletim

sistemlerinin çoklu işlemcilere uyarlanması, genellikle tüm çekirdeğin tekrar

tasarlanması ve tekrar gerçekleştirilmesi anlamına gelir.

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• Modern işletim sistemleri, daha tasarımın başında birden çok işlemci ile

çalışacak şekilde oluşturulmaktadır.

• Çok işlemcili işletim sistemleri, birden çok CPU üzerinden yüksek başarım

sağlamayı hedeflemektedir. Uygulama için CPU sayısını saydam yapmak (yani

16

kullanıcı uygulamadan sistemde kaç tane CPU olduğunu gizlemek) önemli bir

amaçtır. Böyle bir saydamlığı başarmak görece kolaydır, çünkü farklı

uygulamalar (ve bunların parçaları) aynı ilkelleri (primitives) iletişim sırasında

birbirleri arasında kullanırlar, bunları çok işli (multitask) tek işlemcili işletim

sistemindeki gibi yaparlar.

• Buradaki ana fikir, iletişimin paylaşımlı bellek yerleşimlerinde bulunan verinin

yönetilmesi ile yapılmasıdır, ve veri sadece eşanlı (simultaneous) erişimden

korunmaktadır.

• Koruma senkronizasyon ilkelleri ile sağlanmaktadır. İki önemli (ve birbirine

denk) ilkel ise, semafor ve monitör’dür.

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• Semafor, yukarı ve aşağı şeklinde iki işlemi olan bir tam sayı olarak

düşünülebilir. Aşağı işlemi, semafor’un değerinin sıfırdan büyük olup

olmadığını kontrol etmek içindir. Eğer öyleyse, değerini artırarak devam eder.

Eğer değeri sıfır ise, çağıran süreç bloklanır.

17

• Yukarı işlemi bunun tam aksini yapar. Bu ilk önce, daha önceki bir aşağı

işlemini tamamlamak için yapılmamış olan şu an-bloklanmış süreçlerin olup

olmadığını kontrol eder. Eğer öyleyse, bunlardan birinin bloklanmasını kaldırır

(unblocking) ve devam eder. Aksi halde, basitçe semafor değerini artırır.

Bloklanmamış bir süreç, aşağı işleminden dönüşte basitçe işine devam eder.

• Semafor işlemlerin en önemli özelliği, atomik olmalarıdır. Bunun anlamı, bir

aşağı veya yukarı işlemi başlatıldığında, başka hiçbir süreç semafor’a işlem

bitene kadar erişemez (veya bir süreç bloklayana kadar erişemez).

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• Süreçleri senkronize etmek için semaforlar ile programlama, basitçe korunumlu

veri ile kullanımının haricinde, hataya-eğilimli (error-prone) bir durum olarak

bilinir. Ana problem, semaforların kullanımı yapılandırılmamış kod’a

(unstructured code) neden olur, benzer olarak gereksiz “goto” ifadesi’nin çok

18

miktarda kullanımı da buna benzer bir sonuç oluşturur. Buna alternatif olarak,

bir çok modern sistem, eşzamanlı programlama’yı (concurrent

programming) destekleyen monitör gerçekleştirimleri için bir kütüphaneyi

sağlamaktadır.

• Resmi olarak, bir monitör bir programlama-dili yapısıdır, nesne-temelli

programlamadaki bir nesne’ye benzemektedir. Bir monitör, değişkenler ve

prosedürlerden oluşan bir modül olarak düşünülebilir. Değişkenler, sadece

monitör prosedürlerinden birinin çağrılması ile erişilebilirler. Bu mantıkla, bir

monitör, bir nesneye oldukça benzerdir. Bir nesne kendi özel verisine sahiptir.

Bu veri, sadece nesne tarafından gerçekleştirilen metotların vasıtasıyla

erişilebilir olur. Monitör ile nesneler arasındaki fark, monitör’ün sadece tek bir

sürecin bir seferde bir prosedür çalıştırmasına izin vermesidir.

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

monitor Counter {

Fig. 1.12.

19

private:

int count = 0;

public:

int value() { return count;}

void incr () { count = count + 1;}

void decr() { count = count

– 1;}

• Şekil 1.12 ‘de, eşzamanlı (concurrent) erişim sırasında bir tamsayıyı (integer)

korumak için oluşturulan bir monitor görülmektedir.

void decr() { count = count

– 1;}

}

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• Diğer bir deyişle, eğer bir A süreci bir monitör içeren bir prosedürü

çalıştırdığında (buna A süreci monitöre “girmiştir” denilir) ve bir B süreci

monitör’ün prosedürlerinden birini çağırırsa, A süreci işini bitirene kadar B

süreci bloklanır (yani A süreci monitörden “çıkana” kadar).

20

• Bir örnek olarak, bir tamsayı değişkeni koruyan basit bir monitörü göz önüne

alalım. Buna ait şematik Şekil 1.12 ‘de verilmiştir. Monitör tek bir (özel)

değişken sayısını/sayacını içerir. Bu sayıya/sayaca, sadece üç (halka açık,

public) prosedür vasıtasıyla erişilebilmektedir. Bunlar sırasıyla; onun şu anki

değerini okuma, değerini

1 ‘er artırma veya azaltma prosedürleridir. Monitör

yapısı, bu prosedürlerin birini çağıran herhangi bir sürecin atomik olarak

monitör tarafından içerilen özel veriye erişeceğini garanti eder.

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• Uzunca bir zaman, monitörler paylaşımlı veri’nin basitçe korunmasında

kullanışlı olmuşlardır. Bununla beraber, bir sürecin koşulsal olarak bloklanması

istenebilir. Örneğin, “azalt” isimli bir işlemi çağıran bir süreci bloklamak istiyor

olalım. Bu işlem, bir sayının/sayacın değerinin sıfıra ne zaman düştüğünü

21

bulmaktadır. Bu amaçla, monitörler koşulsal değişkenler

(conditional

variables) içermektedirler. Bunlar, “bekle” (wait) ve “sinyal” (signal) isimli iki

özel değişkendirler.

• Bir A süreci bir monitör içerisindeyse, bu monitörün içerdiği koşulsal bir

değişkeni üzerinde “bekle” değişkenini çağırır, A süreci bloklanır ve monitöre

olan dışlayan erişiminden vazgeçer. Sonuçta, monitöre girmek için beklemiş bir

B süreci, bundan sonra devam edebilir. Belirli bir noktada, B süreci A sürecinin

bloklamasını kaldırır (unblock). Bunu A süreci’nin beklemekte olduğu “sinyal”

isimli değişkeni koşulsal değişkeni oluşturarak yapar. Monitör içerisinde iki

sürecin aktif olmasını önlemek için, sinyalleşen süreçlerin monitörü terk etmesi

gerektiği bir şema’nın uygulanması gerekir.

• Şekil 1.13 ’de bir monitör şematik olarak gösterilmektedir.

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

monitor Counter {

private: void decr() {

Fig. 1.13.

22

int count = 0;

int blocked\_procs = 0;

condition unblocked;

public:

int value () { return count;}

void incr () {

if (blocked\_procs == 0)

if (count ==0) {

blocked\_procs = blocked\_procs + 1;

wait (unblocked);

blocked\_procs = blocked\_procs – 1;

}

else

count = count – 1;

• Şekil 1.13 ‘de, eşzamanlı (concurrent) erişim sırasında bir tamsayıyı (integer)

korumak için oluşturulan bir monitör görülmektedir, fakat bir süreç

tarafından bloklama mevcuttur.

count = count + 1;

else

signal (unblocked);

}

}

}

Yazılım kavramları – Çoklu işlemcili Đşletim

Sistemleri (Multiprocessor Operating Systems)

• Monitörlerin bir eksiği ise, programlama-dili yapısı olmalarıdır. Örneğin, Java

monitörler için bir kavram sağlamaktadır. Buna göre, temelde senkronize

olmuş ifadeler sayesinde eşzamanlı erişimlerden her bir nesne’nin kendisini

korumasına izin verilmektedir. Java, nesneler için

“bekle” (wait) ve “bildir”

23

korumasına izin verilmektedir. Java, nesneler için

“bekle” (wait) ve “bildir”

(notify) isimli işlemlerini de sağlamaktadır.

• Monitörler için kütüphane desteği genellikle, sadece sıfır ve bir değerlerini

alabilen basit semaforlar vasıtasıyla verilmektedir, bunlara genel olarak mutex

değişkenleri

(mutex variables) denilmektedir. Bunlar, “kilitleme” (lock) ve

“kilit açma” (unlock) işlemleri ile ilişkilidirler.

• Bir mutex ’i kilitlemek, sadece bu mutex değeri

1 ise yapılabilir. Aksi halde,

çağrıyı yapan süreç bloklanır. Benzer olarak, bazı bekleyen bir sürecin kilidi

açılmadıkça bir mutex ‘in “kilidini açma”, onun değerini

1 ‘e ayarlamaktır.

İlişkili işlemleri ile birlikte koşulsal değişkenler kütüphane rutinleri olarak

ayrıca sağlanmaktadır.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

• Çoklu bilgisayarlar için işletim sistemleri, tamamen farklı yapıdadır ve çoklu

işlemcili işletim sistemlerinden daha karmaşıklardır. Bu fark, sistem-çapında

kaynak yönetimi için veri yapılarının sadece fiziksel olarak paylaşılmış bellekte

bunların kolaylıkla daha fazla süre paylaşılamayacağıdır. Bunun yerine, iletişim

24

mesaj geçme

(message passing) ile yapılmaktadır. Şekil 1.14 ‘te çoklu

bilgisayar işletim sistemlerine ait genel organizasyon gösterilmektedir.

• Her bir düğüm; bellek, yerel CPU, bir yerel disk ve benzeri yerel kaynakların

yönetilmesi için modüller içeren kendi çekirdeğine sahiptir. Böylece, her bir

düğüm işlemciler arası iletişim’in (interprocessor communication)

idaresi için ayrı bir modüle sahiptir, düğümlerden ve düğümlere mesajların

alımı ve gönderimi gerçekleşmektedir.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

Fig. 1.14

25

Şekil 1.14 ‘de, bir çoklu bilgisayar işletim sistemi’nin (multicomputer

operating system) genel yapısı görülmektedir.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

• Her bir yerel çekirdek, genel bir yazılım katmanıdır. Bu katman, çeşitli işlerin

(tasks) eşzamanlı ve paralel olarak çalıştırılmasını destekleyen sana bir makine

olarak işletim sistemini gerçekleştirir. Bu katman, bir çoklu işlemci makinesine

ait soyutlamayı sağlayabilir. Soyutlama; kelime anlamı olarak, belirli bir amaca

26

özgü bilgiler üzerinde odaklanan, bunun dışındakileri göz ardı eden işlemler

olarak bilinir.

• Diğer bir deyişle, paylaşımlı belleğin tam bir yazılım gerçekleştirimi

sağlanmaktadır. Bu katmanda genellikle gerçekleştirilen ek imkanlar, örneğin,

bir işlemciye bir işin atanması, donanım hatalarının gizlenmesi, saydam

depolamanın sağlanması ve genel süreçler arası iletişimin sağlanması gibi

imkanlardır. Diğer bir deyişle, bu imkanlar herhangi bir işletim sisteminden

normal olarak beklenilmelidir.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

• Paylaşımlı bellek için bir kavram sağlamayan çoklu bilgisayar işletim sistemleri,

uygulamalar için sadece mesaj-geçme imkanları önermektedirler. Ne yazık ki,

mesaj-geçme ilkelleri ’ne (message-passing primitives) ait anlamsallık

farklı sistemler arasında geniş çapta değişmektedir.

27

• Mesajların tamponlanıp tamponlanmamasına bakılarak aralarındaki farkı

açıklamak daha kolay olur. Ayrıca, alıcı ve gönderici sürecin bloklanmış

olmasını da göz önüne almak gerekir. Şekil 1.15 ‘te tamponlama ve bloklama

şematik olarak gösterilmektedir.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

Fig. 1.15

28

Şekil 1.15 ‘de, mesaj geçme için bloklama ve tamponlama için alternatifler

görülmektedir.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

• Mesajların tamponlanabildiği iki yer bulunmaktadır. Bunlar; gönderici

tarafta

(sender’s side) ve alıcı tarafta

(receiver’s side) olmaktadır. Bu

durum, dört tane olası senkronizasyon noktası oluşmasına neden olur. Bunun

anlamı gönderici ve alıcı’nın bloklanabilme olasılığının bulunduğu yer

29

olmasıdır. Eğer gönderici tarafta bir tampon varsa, sadece tampon dolduğunda

gönderici’nin bloklanması gerektiği anlamına gelir. Bu durum Şekil 1.15 ‘te S

1

senkronizasyon noktası olarak gösterilmektedir. Alternatif olarak, bir mesaj bir

tampona yerleştirildiğinde, işlemin başarılı olduğunu gösteren bir durum

(status) mesajı geri döndürülür. Bu durum, tampon tamamen dolu olduğunda,

göndericinin bloklanmasını engeller.

• Aksi halde, gönderici tamponu yoksa, göndericiyi bloklamak için üç alternatif

nokta bulunmaktadır: Mesajın gönderildiği yer (Şeki nokta bulunmaktadır: Mesajın gönderildiği yer (Şekil 1.15 ‘te S

2 noktası olarak noktası olarak

görülmektedir), mesajın alıcıya vardığı yer (Şekil 1.15’te S

3 senkronizasyon

noktası olarak görülmektedir) veya mesaj alıcıya teslim edildiğinde

(Şekil 1.15’te S

4 noktası olarak görülmektedir).

• Eğer bloklama ya S

2, S

3 veya S

4 ‘te oluyor ise gönderici tarafta bir tampona

sahip olmanın hiçbir anlamı yoktur.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

• Alıcıyı bloklamak sadece senkronizasyon noktası S

3 ‘te anlamlı olmaktadır. Bu

durum hiçbir alıcı tamponu bulunmadığı ve tampon boş olduğu durumda

meydana gelir. Gelen mesajlar için alıcı havuzunu oluşturmak bu duruma bir

alternatif olabilir. Bununla beraber, böyle yapılması CPU zamanının boşa

30

harcanmasına neden olur, veya gelen mesajlara çok geç cevap verilir. Böylece,

bu durum gelen mesajların atılması (dropped) ile sonuçlanan tampon

taşmalarına neden olabilir.

• Bir diğer konu ise, mesaj-geçme anlamsallıklarının anlaşılabilmesi için

önemlidir. Bu konu iletişimin güvenilir olup-olmamasıdır. Güvenilir iletişimin

ayırt edici özelliği, gönderici’nin mesajlarının alınabileceğine dair verilen

garantidir.

• Şekil 1.15 ‘te, tüm mesajlar senkronizasyon noktası S

3 için tüm mesajlar garanti

edilmiştir. Güvenilir-olmayan (unreliable) iletişimde, hiç bir garanti

bulunmamaktadır. Gönderici taraftaki iletişimde bir tampon bulunuyor ise ya

güvenilirdir ya da değildir. Benzer olarak, gönderici S

2 ‘de bloklandığında

işletim sistemi güvenilir iletişimi garanti etmeye ihtiyaç duymaz.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

• Bununla beraber, eğer işletim sistemi bir göndericiyi mesajlar S

3 veya S

4 ‘e

gelene kadar blokluyor ise, güvenilir bir iletişimi de garanti etmesi gerekir.

Ayrıca, gönderici’nin alım (receipt) veya teslimat (delivery) ile ilgili onayı

beklediği bir durumla da karşılaşılabilinir, bu sırada mesaj iletim sırasında

31

kaybolmuştur. Bloklama (blocking), tamponlama (buffering) ve güvenilir

iletişimin garanti edildiği durumlar Şekil 1.16 ‘da özetlenmiştir.

• Çoklu bilgisayarlı işletim sistemlerinin oluşturulması bir çok konuyu

içermektedir. Çoklu bilgisayarlı işletim sistemleri ve dağıtık sistemler

arasındaki ana fark, çoklu bilgisayar işletim sistemlerinin genel olarak, altta

yatan donanımı homojen ve tamamen yönetilmiş olarak varsaymasıdır. Bir çok

dağıtık sistem, bununla beraber, sıklıkla var olan işletim sistemlerinin üzerine

inşa edilirler.

Yazılım kavramları – Çoklu bilgisayar Đşletim

Sistemleri (Multicomputer Operating Systems)

Fig. 1.16

32

Senkronizasyon noktası (Synchronization

point)

Gönderim

tamponu

(Send buffer)

Güvenilir iletişim

sağlanıyor mu?

(Reliable comm.

guaranteed?)

Tampon dolmayıncaya kadar Göndericiyi blokla

(Block sender until buffer not full) Evet Gerekli değil

Mesaj gönderilene kadar Göndericiyi blokla

(Block sender until message sent) Hayır Gerekli değil

Mesaj alınana kadar Göndericiyi blokla

(Block sender until message received) Hayır Gerekli

Mesaj teslim edilene kadar Göndericiyi blokla

(Block sender until message delivered) Hayır Gerekli

ekil 1.16 ‘da; Bloklama, tamponlama ve güvenli iletişim arasındaki ili

şki

gösterilmektedir.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

• Pratikler göstermiştir ki, çoklu bilgisayarların programlanması çoklu

işlemcilerin programlanmasından oldukça zordur. Aradaki fark, paylaşımlı

veriye erişen süreçlerin temelinde iletişimin ifade edilmesi ve semafor, monitör

gibi senkronizasyon ilkellerinin kullanımının sadece mesaj

-geçme imkanlarının

33

kullanılırlığından daha basit olmasıdır. Tamponlama, bloklama ve güvenilir

iletişim sadece işleri kötüleştirir.

• Bu nedenle, çoklu bilgisayarlardaki paylaşımlı belleğin öykünme ile

oluşturulması (emulating) oldukça yoğun bir araştırma alanıdır. Buradaki

amaç, çoklu bilgisayar üzerinde çalışan sanal bir paylaşımlı bellek

sağlanmasıdır.

• Buna göre, uygulamalar (hali hazırda var olmadığı düşünülen) bir paylaşımlı

bellek modeli kullanılarak yazılacaktır. Burada, çoklu bilgisayar işletim sistemi

önemli bir rol oynamaktadır.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

• Bir yaklaşım olarak, büyük bir sanal adres uzayını desteklemek için her bir

bireysel düğümün sanal bellek kapasitelerini kullanmak ortaya atılmıştır. Bu

yaklaşım, sayfa-temelli bir dağıtık paylaşımlı bellek

(distributed shared

memory, DSM) yapısına neden olmaktadır.

34

• DSM ‘nin prensiplerine bakacak olursak; bir DSM sisteminde, adres uzayı

sayfalara (pages) bölünmüştür (bunlar tipik olarak

4 KB veya

8 KB olmaktadır),

sistemdeki tüm işlemcilere yayılmış sayfalar ile bu işlem yapılmaktadır. Bir

işlemci yerelde bulunmayan bir adresi işaret ettiğinde (reference), bir tuzak

(trap) oluşur ve işletim sistemi adresi içeren sayfayı alıp getirir (fetch) ve hatalı

komutu (işlemi) tekrar başlatır.

• Böylece bu işlem (instruction) başarı ile tamamlanır. Bu kavram Şekil 1.17 (a)

şıkkında, 16 sayfalı bir adres uzayı için ve dört işlemci için gösterilmektedir. Bu

temelde normal sayfalama (paging) olrak bilinir. Burada uzak RAM (bellek)

yerel diskin yerine bir depolama birimi gibi kullanılmaktadır.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

a) Dört makine

arasındaki dağıtık

Fig. 1.17.

35

adres uzayına ait

sayfalar.

b) CPU 1 ‘in sayfa 10 ‘u

referans vermesinden

sonraki durum.

c) Eğer sayfa 10 saltokunur ise ve

kopyalama

(replication) var ise

oluşan durum.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

• Bu örnekte, eğer

1 no’lu işlemci; 0, 2, 5 veya

9 ‘uncu sayfalardaki işlem veya

veriyi işaret ediyor (referans gösteriyor) ise, bu referanslar yerelde

oluşturulacaktır. Diğer saylardaki referanslar tuzak’a (traps) neden olur.

Örneğin, 10 ‘uncu sayfadaki bir adresin referansı işletim sistemi için bir tuzak

36

oluşmasına neden olmaktadır. Böylece, Şekil 1.17 (b) şıkkında görüldüğü gibi,

2

no’lu makine’den

1 no’lu makineye 10 no’lu sayfa taşınır.

• Temel sistemi daha da geliştiren bir olgu ise, sıklıkla başarımı artıran saltokunur olan sayfaların kopyalanması’dır (replicate). Örneğin, program metnini,

salt-okunur sabitleri veya salt-okunur veri yapılarını içeren sayfalar gibi.

• Eğer Şekil 1.17 ‘deki 10 ‘nolu sayfa bir program metni ise,

1 no’lu işlemci

tarafından bunun kullanımı,

1 ‘nolu işlemciye bunun bir kopyasının

gönderilmesi ile sonuçlanır. Bu sırada

2 no’lu işlemci’nin belleğindeki orijinal

kopya dağıtılmamaktadır. Bu durum Şekil 1.17 (c) şıkkında görülmektedir. Bu

yolla,

1 no’lu işlemci ve

2 no’lu işlemci 10 no’lu sayfayı, istenilen sıklıkta kayıp

belleğin alıp getirilmesine ait tuzağa neden olmadan referans gösterir.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

• Diğer bir olasılık ise sadece salt-okunur sayfaları değil tüm sayfaları

kopyalamaktır (replicate). Yeterince uzun süre okuma yapıldığında, saltokunur

(read-only) bir sayfa ile yazılır-okunur

(read-write) bir sayfa

arasında etkin olarak fark yoktur.

37

• Bununla beraber, eğer bir kopyalanmış sayfa aniden değişikliğe uğrarsa, birden

çok, tutarsız kopyanın var olmasını önlemek için özel bir hareket yapılır. Tipik

olarak tüm kopyalar, fakat bir tanesi yazma işlemi gerçekleşmesine izin

verilmeden evvel geçersiz kılınır (invalidated).

• Daha ileri bir başarım iyileştirilmesi, eğer kopyalanmış sayfalar arasında

kesin tutarlılık

(strict consistency) oluşturulursa, sağlanmaktadır. Diğer

bir deyişle, bir kopyanın geçici olarak diğerlerinden farklı olmasına izin verilmiş

olur. Pratikler bu yaklaşımın gerçekten yardımcı olduğunu göstermektedir,

fakat ne yazık ki, programlayıcıların hayatını oldukça güç hale getirmektedir,

bunun başlıca sebebi ise programcının böyle tutarsızlıklardan haberdar

olmasıdır. Bu göz önüne alındığında, programlamanın kolaylaştırılması DSM

sistemlerinin geliştirilmesinde önemli bir neden olarak ilk sırayı almaktadır.

Zayıflatılmış tutarlılık gerçek bir alternatif olmayabilir.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

• Etkin DSM sistemlerinin tasarlanmasında başka bir konu ise, sayfaların ne

kadar büyük olacağıdır. Burada karşılaşılan bir diğer zorluk ise, tek işlemcili

sanal bellek sistemlerinde sayfa büyüklüğüne karar vermektir. Örneğin, bir

network üzerinden bir sayfanın iletilmesinin maliyeti, ilk önce iletim

38

kurulumunun maliyeti ile belirlenmekte, fakat iletilen verinin miktarı ile

belirlenmemektedir.

• Sonuçta, büyük sayfalara sahip olmak, bitişik bir verinin büyük bir bölümüne

erişilmek istenildiğinde toplam iletim sayısını (total number of transfers)

azaltabilecektir. Diğer yandan, eğer bir sayfa farklı işlemciler üzerindeki iki

bağımsız sürece ait verileri içeriyor ise, işletim sistemi bu iki süreç arasında

sayfaları tekrar tekrar iletmeye (transfer) ihtiyaç duyabilir. Bu durum Şekil 1.18

‘de görülmektedir. Aynı sayfada iki bağımsız sürece ait veriye sahip olmaya,

yanlış paylaşım

(false sharing) denilmektedir.

Yazılım kavramları –Dağıtık Paylaşımlı Bellek

Sistemleri (Distributed Shared Memory Systems)

Fig. 1.18.

39

Şekil 1.18 ‘de, iki bağımsız süreç arasında bir sayfanın Yanlış paylaşımı

(False sharing) görülmektedir.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Dağıtık işletim sistemlerinin aksine, ağ işletim sistemleri altta yatan donanımın

homojen olduğunu varsaymaz. Ayrıca, bu ağ işletim sistemleri (NOS) tek bir

sistem gibi yönetilmek zorunda olduğunu da varsaymaz. Bunun yerine, bunlar

genellikle tek işlemcili sistemlerin bir topluluğundan oluşturulurlar.

40

• Tek işlemcili sistemlerin her biri kendi işletim sistemine sahiptir. Bu durum

Şekil 1.19 ‘da görülmektedir. Makineler ve onların işletim sistemleri farklı

olabilir, fakat bir bilgisayar ağı üzerinden her biri birbirine bağlanmıştır.

Böylece, ağ işletim sistemleri belirli bir makine üzerindeki kullanılır servislerin

kullanılması için kullanıcılara izin verecek imkanları sağlamaktadırlar.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

Fig. 1.19

41

Şekil 1.19 ‘da, bir ağ işletim sistemi’nin (Network operating system, NOS)

genel yapısı görülmektedir.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Genel olarak, ağ işletim sistemleri tarafından sağlanan bir servis, bir kullanıcıya

başka bir makineye bir komut kullanarak uzaktan erişebilmesi ve log in olması

iznini vermektedir:

42

rlogin makine

• Bu komut’un etkisi, kullanıcı’nın kendi iş istasyonunu uzak makineye erişip

girebilecek uzak bir uç birim (terminal) haline getirir. Bu esnada, kullanıcı bir

grafiksel iş istasyonunun başında bulunmakta, komutlar klavyeden girilmekte

ve böylece uzak makineye gönderilmektedir.

• Uzak makinenin çıktısı, kullanıcının ekranında gösterilmektedir. Farklı bir uzak

makineye geçmek için, yen bir pencere açarak orada da rlogin komutu

kullanarak başka bir makineye bağlanılabilir. Bu sırada makine seçimi

tamamen elle yapılmaktadır.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Ağ işletim sistemleri sıklıkla bir makineden bir diğerine dosyaları kopyalamak

için bir uzak kopya komutunu kullanmaktadırlar:

rcp makine

1:dosya

1 makine

2:dosya

2

43

rcp makine

1:dosya

1 makine

2:dosya

2

• Böylece dosya1 makine1 ‘den makine2 ‘ye ismi dosya2 olarak de

ğiştirilmi

ş olarak

kopyalanmı

ş olur. Dosyaların ta

şınması i

şlemi kesindir ve tüm dosyaların nerede

bulundu

ğundan ve tüm komutların nerede çalı

ştırıldı

ğından kullanıcı tamamen haberdar

olmaktadır.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Bu çeşit iletişim son derece ilkeldir ve sistem tasarımcılarını, bilgi paylaşımı ve

iletişim için daha elverişli biçimlerini araştırmaya sevk eder. Bir yaklaşım da,

paylaşımlı, global dosya sistemine tüm iş istasyonlarından erişilebilirliğin

44

sağlanmasıdır. Dosya sistemi, dosya sunucuları

(file servers) denilen bir

veya daha fazla makine tarafından desteklenmektedir.

• Dosya sunucuları, diğer (sunucu olmayan) makineler üzerinde çalışan kullanıcı

programlarından gelen istekleri

(requests) kabul ederler. Bu makinelere

istemci

(clients) denilmektedir. Bu istekler üzerine yazma ve okuma işlemi

yapılır. Her bir gelen istek işlenir ve çalıştırılır, sonra cevap

(reply) geri

döndürülür. Bu aşamalar Şekil 1.20 ‘de gösterilmektedir.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

Fig. 1.20

45

Şekil 1.20 ‘de, bir ağ işletim sistemi’ndeki iki istemci ve bir sunucu

görülmektedir.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Dosya sunucuları genelde hiyerarşik dosya sistemlerinin bakımını yapar, her

biri alt dizinler

(subdirectories) ve dosyalar içeren bir kök

(root) dizinine

sahiptir. İşistasyonları (workstations) bu dosya sistemlerini içe aktarabilir

(import) veya bağlayabilirler (mount). Bu sayede kendi yerel dosya sistemlerini,

46

sunucu üzerinde bulunan diğerleri ile genişletebilirler. Örneğin, Şekil 1.21 ‘de

iki dosya sunucusu görülmektedir.

• Bir sunucunun dizin hiyerarşisinde bir istemicinin nereye bağlandığı sıklıkla

sorun oluşturmaz, şunu bilmek önemlidir ki, farklı istemciler dosya sisteminin

farklı görünümlerini sahip olabilmektedirler. Bir dosyanın ismi kendisine

nereden erişildiği ve kendi dosya sistemini bu makinenin nasıl kurduğuna

bağlıdır. Çünkü her bir istemci makine diğerlerinden görece bağımsız olarak

çalışmaktadır, bu sebeple hepsinin kendi programlarında aynı dizin

hiyerarşisini sağlaması için bir garanti bulunmamaktadır.

• Ağ işletim sistemleri, dağıtık sistemlerden daha ilkeldir. Bu iki tip işletim

sistemi için temel ayrışma noktası, dağıtık sistemlerin tam saydamlığı (full

transparency), tek bir sistem görünümü için sağlamaya ciddi anlamda

çalışmasıdır. Bunun anlamı, ağ işletim sistemlerindeki gibi hiyerarşi sorunu

oluşmamasının sağlanmaya çalışılmasıdır.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Ağ işletim sistemlerinde saydamlığın eksikliği bazı belirli kusurlara neden olur.

Örneğin, uzaktaki makinelere kullanıcılar girmek (log in olmak) istediğinde

veya bir makineden bir diğerine dosyaları kopyalamak istediğinde, ağ işletim

sistemlerini kullanmak sıklıkla güç olabilmektedir. Burada bir yönetim

47

problemi olduğu görülür. Çünkü ağ işletim sistemi içerisindeki tüm makineler

bağımsızdır, sıklıkla sadece bağımsız olarak yönetilebilirler. Bunun bir sonucu

olarak, bir kullanıcı uzak bir

X makinesine log in olmaya (girmeye)

çalışabilmesi için

X makinesi üzerinde bir hesabının olması gerekmektedir.

• Benzer olarak, bir kullanıcı sadece tek bir şifre

(password) kullanmak da

isteyebilir. Bu durumda bir şifrenin değiştirilmesi her bir makinede de

değiştirilmesini gerektirir. Genel olarak tüm erişimler için izinler her bir

makine için bakımı yapılması (kontrol edilmesi) gerekli hale gelir. Her yerde

aynı olduğunda izinlerde değişiklik yapmak için basit bir yol bulunmamaktadır.

• Bu güvenlik için ademi-merkezi (decentralized) yaklaşım, istenmeyen zararlı

saldırılara karşı ağ işletim sisteminin korunmasını bazen zor bir hale getirir.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

Fig. 1.21.

48

Şekil 1.21 ‘de farklı yerlerden sunucuya bağlanan farklı istemciler görülmektedir.

Yazılım kavramları –Ağ Đşletim Sistemleri

(Network Operating Systems)

• Dağıtık sistemler ile karşılaştırıldığında ağ işletim sistemlerinin bazı avantajları

da mevcuttur. Ağ işletim sistemindeki düğümler birbirlerinden oldukça fazla

bir biçimde bağımsızdırlar, bu sebeple bir makinenin eklenilmesi veya

çıkartılması oldukça kolaydır.

49

• Bazı durumlarda, bir makineyi böyle bir sisteme eklemek için genel bir ağ’a

bağlamak ve ardından diğer makineler tarafından bilinir (görünür) yapılması

yeterli olmaktadır. Internet’te, örneğin, yeni bir sunucunun eklenilmesi buna

benzer bir yoldan yapılır.

• Tüm Internet üzerinden bir makinenin bilinir olması olması için, O’nun ağ

adresinin sağlanılması veya daha iyi olarak ona bir sembolik isim verilmesi

gerekmektedir. Bu sembolik isim DNS adresine O’nun ağ adresi ile

eklenilmektedir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

• Ne bir dağıtık işletim sistemi ne de bir ağ işletim sistemi gerçekten bir dağıtık

sistemin literatürde verilen tanımını karşılamamaktadır. Bir dağıtık işletim

sistemi, bağımsız bilgisayarların bir topluluğunu idare etmeyi

amaçlamamaktadır. Bir ağ işletim sistemi de tek bir tutarlı sistem görünümünü

50

sağlamamaktadır.

• Akla şöyle bir soru gelebilir; hem ağ işletim sistemlerinin ölçeklenebilirliği

(scalability) ve açıklığı (openness), hem de dağıtık işletim sistemlerinin

saydamlığı

(transparency) ve görece kolay kullanımını kullanarak bunları

bir araya getiren bir dağıtık sistem geliştirilemez mi?

• Çözüm, ek bir yazılım katmanı ile sağlanabilmektedir. Buna göre; ağ işletim

sistemleri için altta yatan platformların heterojenliğini az yada çok gizlemek ve

bu sırada dağıtım saydamlığını iyileştirmek adına bu katman kullanılmaktadır.

Bir çok modern dağıtık sistem bu ek katmana ara katman yazılımı

(middleware) ismini vermektedir.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Ara katman yazılımı (Middleware) yerleştirilmesi (positioninig) :

Bir çok dağıtık uygulama, ağ işletim sistemi tarafından önerilen programlama

ara yüzünü doğrudan kullanmaktadır. Örneğin, iletişim sıklıkla soketler

üzerinden yapılan işlemler ile ifade edilmektedir. Bu sayede, farklı

makinelerdeki süreçlere birbirlerine mesaj geçirebilmeleri (pass) için izin

51

verilmiş olmaktadır. Ayrıca, uygulamalar sıklıkla yerel dosya sistemine ait ara

yüzlerden yararlanmaktadırlar.

Bu yaklaşımla ile ilgili bir problem olarak; dağıtım oldukça fazla saydamdır.

Buna bir çözüm ise; uygulamalar ile ağ işletim sistemi arasına ek bir yazılım

katmanının yerleştirilmesidir. Bu katman daha yüksek seviden bir soyutlama

önermektedir. Bu katmana ara katman yazılımı (middleware) denilir. Şekil 1.22

’de buna ait şematik gösterilmektedir.

Soyutlama kelimesine baktığımızda aşağıdaki anlamlara geldiğini görürüz;

1) Belirli bir amaca özgü bilgiler üzerinde odaklanan, bunun dışındakileri göz

ardı eden işlemler.

2) Uygulama programları ara yüzü (API) gibi özel yazılım araçları ile bir

yazılımı makine tipine bağımlılıktan ve karmaşıklıktan kurtarma. Örneğin, bir

iletişim yazılımını sadece iletişim görevine odaklamak ve iletişim aygıtlarının

farklılıklarıyla uğraşmasını engellemek.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) yerleştirilmesi (positioninig) :

Altta yatan ağ işletim sisteminin her bir yerel sistemi biçimlendiren parçasının

basit iletişime ek olarak yerel kaynak yönetimini de sağladığı varsayılır. Bu

basit iletişimin bilgisayarların birbirine bağlanması olarak göz önüne alabiliriz.

52

Diğer bir deyişle, ara katman yazılımı bireysel bir düğümü yönetmez, bu iş

tamamen yerel işletim sistemine bırakılmıştır.

Uygulamalardan altta yatan donanımın heterojenliğinin gizlenmesi önemli bir

amaçtır. Böylece, bir çok ara katman yazılım sistemi az-veya-çok tam bir servis

topluluğunu önermektedir ve başka herhangi bir şeyin kullanımının önüne

geçerek bu servisler için kendi ara yüzlerinin kullanılmasını önermektedirler.

Diğer bir deyişle, ara katman yazılımının atlanarak hemen altta yatan işletim

sistemine ait bir servisin çağırılması, sıklıkla hoş karşılanmamaktadır.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) yerleştirilmesi (positioninig) :

Fig. 1.22

53

Şekil 1.22 ‘de, ara katman yazılımı olarak bir dağıtık sistemin genel yapısı

görülmektedir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) yerleştirilmesi (positioninig) :

Donanım ve yazılım üreticileri, daha fazla ağ ile bağlantılandırılmış uygulamaya

sahip oldukça, daha yüksek-seviyeli uygulamadan-bağımsız servisleri bu tarz

sistemlerine eklemeye başlamışlardır.

54

Doğal olarak, doğru bir ara katman yazılımı yaklaşımı üzerinde uzlaşma

sağlamaları kolay olmamıştır. Bazı ara katman yazılımı çözümleri için genel bir

standart’ı tanımlayan bir organizasyonun kurulması bir yaklaşım olarak ortaya

atılmıştır.

Günümüzde, böyle belirli sayıdaki standartlar mevcut ve kullanılabilir

durumdadır. Standartlar genel olarak birbirleri ile uyumlu değillerdir ve daha

da kötüsü, ürünü geliştiren farklı üreticilerin bulunduğu durumda aynı

standart’ın gerçekleştiriminde nadiren birlikte çalışırlık sağlanır. Kesinlikle,

birileri “üst-yazılım” önermedikçe bu böyle devam edecektir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) modellemesi:

Dağıtık uygulamaların tümleştirilmesi ve geliştirilmesinin olabildiğince

basitleştirilmesi için, bir çok ara katman yazılımı aynı model’i, veya

paradigma’yı dağıtım ve iletişim için temel almaktadır. Görece basit bir model,

55

her şeye bir dosya gibi davranmaktadır. Bu yaklaşım orjinalinde ilk önce UNIX

sistemlerinde ortaya atılmış, daha sonra Plan

9 dağıtık dosya sisteminde de

takip edilmiştir.

Plan

9 sisteminde, tüm kaynaklar, buna klavye, fare, disk, ağ ara yüzleri gibi

Girdi/Çıktı (I/O) cihazları da dahil olmak üzere dosyalar gibi davranmaktadır.

Temelde, bir dosyanın yerelde veya uzakta olması hiçbir şeyi fark

oluşturmamaktadır.

Bir uygulama önce bir dosyayı açar, byte ’ları okur veya yazar, ve sonuçta

dosyayı kapatır. Çünkü dosyalar bir kaç süreç tarafından paylaşılabilmektedir,

iletişim aynı dosyaya basit bir erişim ile azaltılmaktadır.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Ara katman yazılımı (Middleware) modellemesi:

Basit fakat Plan

9 ‘dan daha az sıkı olan bir yaklaşım ise, dağıtık dosya

sistemleri’nde (distributed file systems) ara katman yazılımı’nın

(middleware) merkezileştiği bir yaklaşımdır. Bir çok durumda, böyle bir ara

katman yazılımı, bir ağ işletim sisteminin sadece bir adım önündedir. Buradaki

56

temel anlam, dağıtım saydamlığı’nın sadece geleneksel dosyalar için

sağlanmasıdır. Buna göre; dosyalar sadece veri depolamak için kullanılırlar.

Örneğin, süreçlerin sıklıkla belirli bir makinede belirgin bir biçimde başlatılmış

olması gerekir. Dağıtık dosya sistemi temelli ara katman yazılımı, ona

popülaritesini sağlayan makul bir ölçeklenebilirliği sağlamaktadır.

Başka bir öncül, ara katman yazılımı modeli ise Uzak Yordam Çağrısı

(Remote Procedure Call, RPC) temellidir. Bu modelde, uzak bir makinede

bulunan bir gerçekleştirimin prosedürü’nü (yordamını) çağırması için bir sürece

izin verilerek ağ iletişiminin gizlenilmesi (kullanıcıya gösterilmemesi) ortaya

atılmıştır.

Böyle bir prosedürün çağrılmasında, parametreler saydam olarak uzaktaki

makineye iletilir, prosedür çalıştırıldıktan sonra sonuçlar arayan tarafa geri

döndürülür.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) modellemesi:

Nesneye yönelim ortaya çıkınca, bu konular daha da görünür hale gelmiştir.

Eğer prosedür çağrılarının makine sınırlarını geçmesi gerekiyor ise, saydam bir

biçimde uzaktaki makinede bulunan nesnelerin çağrılabilmesi de mümkün

57

olabilmelidir. Bu durum, bir çok dağıtık sistemde, dağıtık nesneler

(distributed objects) kavramının ortaya çıkmasına neden olmuştur.

Dağıtık nesnelerin temeli, her bir nesnenin bir ara yüzü gerçekleştirmesidir, bu

ara yüz nesnenin tüm iç detaylarını onun kullanıcılarından gizlemektedir.

Nesnenin gerçekleştirdiği yöntemleri içeren bir ara yüz, bunların ne daha

fazlasını ne de daha azını içerebilir. Bir sürecin gördüğü sadece bir nesnenin ara

yüzü’dür.

Dağıtık nesneler için sıklıkla her bir nesnenin kendisi tek bir makine üzerinde

bulunmakta ve buna ek olarak nesnenin ara yüzü de diğer bir çok makine

üzerinde kullanılabilir olmaktadır. Bir süreç bir yöntemi (method) çağırdığında,

sürecin bulunduğu makinedeki ara yüz gerçekleştirimi (interface

implementation) basitçe yöntem çağrısını bir mesaja dönüştürür. Bu mesaj

nesneye gönderilmiştir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) modellemesi:

Daha sonra nesne istenilen yöntemi çalıştırır ve sonucu geri döndürür. Ara yüz

gerçekleştirimi sonra cevap mesajını bir dönüş değeri’ne (return value)

dönüştürür. Bu değer çağrıyı yapan süreç tarafından ele alınır. RPC

58

durumunda, süreç tamamen ağ iletişiminden habersiz tutulabilmektedir.

Ağ sistemlerinin kullanımı en basite indirgeyen modellerden birisi de Dünya

Çapında Veb

(World Wide Web) olarak verilebilir. Veb ‘in başarısı, son

derece basit ve oldukça etkin bir model olan dağıtık dokümanlar

(distributed documents) modelden kaynaklanmaktadır. Veb modelinde,

bilgi dünyanın herhangi bir yerinde saydam biçimdeki bir makinede bulunan

her bir dokümanı sağlanmaktadır.

Dokümanlar bağlantıları (links) içermekte, bu bağlantılar da başka

dokümanları işaret etmektedir. Bir bağlantının takip edilmesi ile, doküman

kendi yerleşiminden alınarak kullanıcının ekranında gösterilmektedir. Bir

dokümanın ana fikri, sadece metin-tabanlı bilgi ile sınırlandırılamaz. Örneğin,

Veb aynı zamanda ses ve video dokümanlarını, etkileşimli grafik-tabanlı

dokümanları da içermektedir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) servisleri:

Bir çok ara katman yazılımı sistemi için belirli sayıda servis mevcuttur. Kesin

olarak, tüm ara katman yazılımları, bir yoldan veya diğer bir yoldan, erişim

saydamlığı’nı (access transparency) sağlamaya çalışmaktadırlar. Bunu

59

yüksek-seviyeli iletişim imkanları’nı (communication facilities)

önererek yaparlar. Bu imkanlar, bilgisayar ağları üzerinden düşük-seviyeli

mesaj geçilmesini (kullanıcıdan) gizlemektedir.

Ağ işletim sistemi tarafından önerilen iletim katmanı’na (transport layer)

ait programlama ara yüzü, böylece tamamen diğer imkanlar tarafından yer

değiştirilmiş olur. İletişimin nasıl olduğu, kullanıcılara ve uygulamalara

önerilen ara katman yazılımı dağıtım modeline oldukça bağlıdır.

Uzak yordam çağrıları ve dağıtık-nesne çağrılarını göz önüne alacak olursak,

bunların haricinde bir çok ara katman yazılım sistemi uzak veri için saydam

erişim yapılabilmesi adına imkanlar sağlamaktadırlar. Bunlar arasında; dağıtık

dosya sistemleri ve dağıtık veri tabanları sayılabilir. Veb üzerinden saydam

olarak dokümanların ele alınması, yüksek-seviyeli (tek-yönlü) iletişimin başka

bir örneği olarak verilebilir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) servisleri:

Tüm ara katman yazılımları için önemli bir servis de isimlendirme’dir

(naming). İsim servisleri, varlıkların paylaşılması ve (dizinlerin içerisinde

olarak) bakılabilmesi’ne (looked up) izin vermektedir. Bu yapı, telefon rehberi

60

veya gazete ilan sayfaları ’nın (sarı sayfalar) yapısına benzetilebilir.

Bununla beraber, isimlendirme ilk önce oldukça basit gözükebilir,

ölçeklenebilirlik göz önüne alındığında zorluklar ortaya çıkacaktır. Problemler,

büyük-ölçekli sistemlerdeki bir isme etkin olarak bakılması (look up) ile ilgili

olarak oluşmaktadır. Bir varlığın yerleşimi, sabit olduğu varsayımı ile

isimlendirilmiş olmalıdır.

Bu varsayım, bu varsayım Dünya Çapında Veb ‘de yapılmıştır, bir URL anlamı

ile her bir doküman şu an isimlendirilmiş olmaktadır. Bir URL, dokümanın

bulunduğu sunucu ismini depolamaktadır. Böylece, eğer doküman başka bir

sunucuya taşınmış ise, URL adresi çalışmayı bırakacaktır.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Ara katman yazılımı (Middleware) servisleri:

Bir çok ara katman yazılımı, depolama için kalıcı

(persistence) olarak bilinen

özel imkanları önermektedir. Bu basit biçiminde, kalıcılık bir dağıtık dosya

sistemi üzerinden önerilmektedir, fakat daha gelişmiş ara katman yazılımı

sistemlerine veri tabanlarını entegre etmişlerdir, veya diğer bir şekilde veri

61

tabanlarına bağlanma da uygulamalar için imkanlar

(facilities)

sağlanmaktadır.

Veri depolamanın önemli bir rol oynadığı çevrelerde, imkanlar genellikle

dağıtık işlemler

(distributed transactions) için önerilmektedir. İşlem için

önemli bir özellik ise, birden çok okuma ve yazma işleminin atomik

(atomically) yapılmasına izin verilmesidir.

Atomik olma, işlemin ya başarılı olması yada başarısız olması anlamına gelir.

Bunların anlamları, başarılı olması için tüm yazma işlemlerinin güncel olarak

yapılmış olması, başarısız olması için tüm referans edilmiş verilerin

etkilenmemiş olarak bırakılması demektir. Veri üzerinde çalışan dağıtık işler,

birden çok makine arasında yayılmış olabilir.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Ara katman yazılımı (Middleware) servisleri:

Özellikle hataların gizlenilmesi

(masking failures), dağıtık sistemler için

zor olmaktadır, dağıtık işlemler gibi servislerin önerilmesi için önemlidir. Ne

yazık ki, işlemlerin bir çok yerel makine arasında ölçeklenmesi zordur, bu

makineler tek başlarına coğrafik olarak dağıtılmış makinelerdirler.

62

Sonuçta, tüm ara katman yazılımı sistemleri, deneysel olmayan çevrelerde

güvenlik

(security) için imkanları sağlamaktadırlar. Ağ işletim sistemleri için

karşılaştırıldığında, ara katman yazılımında güvenlik ile ilgili problem, ara

katman yazılımının yaygın

(pervasive) olması zorunluluğudur.

Temelde, ara katman yazılımı katmanı altta yatan yerel işletim sistemine tam

bir ağ için güvenliği desteklemekte güvenmemektedir. Bunun sonucunda,

güvenlik ara katman yazılımı katmanının kendisinde kısmen yeniden

gerçekleştirilir. Uzatılabilirlik isteği ile birleştirilerek, güvenlik dağıtık

sistemlerde gerçekleştirilecek en zor servislerden birine dönüşmüş olur.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) ve açıklık (openness):

Modern dağıtık sistemler genellikle, işletim sistemlerinin bir aralığı için ara

katman yazılımı olarak oluşturulmuşlardır. Bu yolla, belirli bir dağıtık sistem

için oluşturulan uygulamalar işletim sisteminden bağımsız hale gelir. Ne yazık

63

ki, bu bağımsızlık sıklıkla belirli bir ara katman yazılımı üzerinde kuvvetli bir

bağımlılık ile değişmektedir. Ara katman yazılımının istenilenden daha az açık

olmasından dolayı problemler oluşmaktadır.

Önceden açıklanıldığı gibi, gerçek bir açık dağıtık sistem tam olan ara yüzler ile

belirlenmektedir. Tam olma, burada sistemi gerçekleştirmek için gerçekten

belirtilmiş gerekli her şeydir. Ara yüz tanımlarının tam olmaması ise, sistem

geliştiricilerinin kendi ara yüzlerini eklemek zorunda bırakıldıkları bir duruma

neden olur. Sonuçta, farklı geliştirme takımlarının ürettiği iki ara katman

yazılımı sisteminin aynı standart’a eklenilmesi, fakat bir sistem için yazılan

uygulamaların kolaylıkla birinden diğerine taşınamadığı (port edilemediği)

ortaya çıkar.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) ve açıklık (openness):

Eşit derecede kötü bir durum ise, tam olmamanın iki farklı gerçekleştirimin asla

bir arada çalışmaması şeklinde olduğu duruma neden olmasıdır. Burada, bu

ikisi tamamı ile (exactly) aynı ara yüz kümesi ile gerçekleştirilmişler fakat altta

64

farklı protokoller bulunmaktadır.

Örneğin, eğer iki farklı gerçekleştirim altta yatan ağ işletim sistemindeki

kullanılabilir olan uyumsuz iletişim protokollerine dayanıyor ise, birlikte

çalışabilirliğin

(interoperability) kolayca başarılabileceği için ortada ufak

bir umut vardır. Ara katman yazılımı protokolleri ve ara katman yazılımı için

ara yüzler aynıdır. Bu durum Şekil 1.23 ‘ten görülmektedir.

Başka bir örnek, farklı gerçekleştirimler arasındaki birlikte çalışabilirlikten

emin olmak için, farklı sistemlerdeki varlıkların aynı yollardan referans edilmiş

olması gerekliliğidir. Eğer diğer sistem ağ adreslerini kullanarak referansları

gerçekleştirirken, bir sistemdeki varlıklar URL’ler olarak referans edilmişler ise,

bu çapraz referans gösterme’nin açıkça probleme neden olacağı aşikardır.

Böyle durumlarda, ara yüz tanımları referansların neye benzediğini detaylı bir

biçimde tarif etmelidir.

Yazılım kavramları – Ara katman yazılımı

(Middleware)

Ara katman yazılımı (Middleware) ve açıklık (openness):

Fig. 1.23

65

Şekil 1.23 ‘den, bir açık ara katman yazılımı-temelli dağıtık sistemde, her bir ara

katman yazılımı katmanı tarafından kullanılan protokollerin aynı olmak zorundadır,

arayüzlerin yanı sıra diğer uygulamalar için de önerildiği anlaşılmaktadır.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Sistemler arasında bir karşılaştırma (A comparison between systems) :

Dağıtık işletim sistemleri ile ağ işletim sistemleri ve (ara katman yazılımıtemelli) dağıtık sistemlerin kısa bir karşılaştırması Şekil 1.24 ‘te verilmektedir.

Saydamlığa bakışla, dağıtık işletim sistemleri ağ işletim sistemlerinden daha iyi

bir iş yaptığı açıkça görülmektedir. Çoklu işlemcili sistemlerde sadece daha fazla

işlemci olduğu gizlenilir, bu görece kolay bir iştir.

66

Asıl zor olan kısım ise, belleğin fiziksel olarak dağıtık olmasını gizlemektir.

Bunun nedeni, çoklu bilgisayar işletim sistemlerini oluştururken tam dağıtım

saydamlığının desteklenmesinin oldukça zor olmasıdır. Dağıtık sistemler sıklıkla

saydamlığın geliştirilmesini iletişim ve dağıtım için belirli bir modelin

uyarlanması ile sağlarlar.

Örneğin, dağıtık dosya sistemleri genellikle dosyalara erişim ve yerleşimlerinin

gizlenilmesinde iyidirler. Bununla beraber, belirli bir modele ait terimlerle her

şeyi ifade etmek için kullanıcıların mecbur tutulduğu bazı genelleştirmeleri

elden kaçırmış olurlar. Belirli bir uygulama için bazen bu model uygunsuz olur.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Sistemler arasında bir karşılaştırma (A Comparison between systems) :

Ö

ğ

e

Da

ğıtık i

şletim sistemi

A

ğ i

şletim

sistemi

Ara katman

yazılımı-temelli

işletim sistemi Çoklu

işlemciler

Çoklu

bilgisayarlar

Saydamlı

ğın derecesi Oldukça

Fig. 1.24

67

Saydamlı

ğın derecesi

(Degree of transparency)

Oldukça

yüksek Yüksek Dü

şük Yüksek

Tüm dü

ğümlerde aynı i

şletim

sistemi

(Same OS on all nodes)

Evet Evet Hayır Hayır

Đşletim sistemi kopya sayısı

(Number of copies of OS) 1 N N N

Đleti

şimin temeli

(Basis for communication)

Payla

şımlı

bellek Mesajlar Dosyalar Modele özgü

Şekil 1.24 ‘de, çoklu işlemcili işletim sistemleri, çoklu bilgisayar işletim sistemleri, ağ işletim

sistemleri ve ara katman yazılımı temelli dağıtık sistemlerin bir karşılaştırması görülmektedir.

Kaynak yönetimi

(Resource management) Genel, merkezi Genel, da

ğıtık Dü

ğüm ba

şına Dü

ğüm ba

şına

Ölçeklenebilirlik (Scalability) Hayır Orta ölçekte Evet De

ğişken

Açıklık (Openness) Kapalı Kapalı Açık Açık

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Sistemler arasında bir karşılaştırma (A comparison between systems) :

Dağıtık işletim sistemleri homojendirler, bu iş her bir düğümün aynı işletim

sistemini çekirdek

(kernel) olarak çalıştırması ile yapılmaktadır. Çoklu

işlemcili sistemler, tabloların hiçbir kopyası gerekmemektedir, hepsi ana bellek

üzerinden paylaşılabilmektedir. Bu durumda, tüm iletişim ana bellek üzerinden

68

olmaktadır, burada çoklu bilgisayar işletim sistemi mesajları kullanılmaktadır.

Ağ işletim sistemlerinde, iletişimin çoğunlukla tamamen dosya tabanlı olduğu

kabul edilmektedir.

Örneğin, Internet’te, bir çok iletişim, dosyaların iletilmesi ile yapılmaktadır.

Bununla beraber, elektronik posta sistemleri ve ilan tahtası biçimindeki yüksekseviyeli mesajlaşmalar, oldukça sık kullanılmaktadır. Ara katman yazılımıtabanlı dağıtık sistemlerdeki iletişim sistem tarafından özellikle uyarlanmış

modele bağımlıdır.

Yazılım kavramları – Ara katman yazılımı (Middleware

)

Sistemler arasında bir karşılaştırma (A comparison between systems) :

Ağ işletim sistemlerindeki ve dağıtık sistemlerdeki kaynaklar, böyle sistemleri

görece daha kolay ölçeklemek için düğüm başına yönetilmektedir. Bununla

beraber, pratikler dağıtık sistemlerdeki ara katman yazılımı katmanının bir

gerçekleştiriminin sıklıkla sınırlı ölçeklenebilirliğe sahip olduğunu

göstermektedir.

69

Dağıtık işletim sistemleri genel kaynak yönetimine sahiptir, bu yüzden

ölçeklenebilmeleri oldukça zor olmaktadır. Çoklu işlemcili sistemlerdeki merkezi

yaklaşım nedeniyle (tüm yönetim verisinin ana bellekte tutulması nedeniyle) bu

sistemlerin sıklıkla ölçeklenmesi zordur.

Sonuçta, ağ işletim sistemi ve dağıtık sistemler açıklık (openness) geldiğinde

kazançlı çıkarlar. Genelde, düğümler TCP/IP gibi bir standart iletişim

protokolünü desteklerler, böylece birlikte çalışabilirlik kolay olmaktadır.

Bununla beraber, işletim sistemlerinin bir çok farklı cinsi kullanıldığında

uygulamaların taşınması sırasında (port edilmesi) bir çok problem ortaya çıkar.

Genelde, dağıtık işletim sistemleri açık (open) olmak üzere tasarlanmamışlardır.

Bunun yerine, başarım için sıklıkla bunlara eniyileme (optimization) yapılır, bu

şekilde açık bir sistemin gittiği yoldaki bir çok çözümlere ulaşılır.

İstemci – Sunucu modeli

(The Client

– Server model)

70

(The Client

– Server model)

Đstemci – Sunucu modeli (The Client – Server Model)

Bu noktada, dağıtık sistemlerin geçerli organizasyonları hakkında herhangi bir

şey söylemek oldukça zordur.

Burada soruların merkezini sistemdeki süreçler’in (processes) nasıl organize

edildiği oluşturmaktadır. Bir çok dağıtık sistemdeki ana fikire ulaşmak bazen

oldukça zor olmaktadır, istemci’ler (clients) ve sunucu’lar (servers) için

71

oldukça zor olmaktadır, istemci’ler (clients) ve sunucu’lar (servers) için

servis istekleri-cevaplarının anlaşılması ve dağıtık sistemlerin karmaşıklığının

yönetilmesi önemli konulardır.

Bu nedenlerle istemci – sunucu modeli’ne (client-server model)

yakından bakmak gerekir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemciler ve Sunucular:

En temel istemci-sunucu modelinde, dağıtık bir sistemdeki süreçler iki

(örtüşmeleri de olası) gruba ayrılırlar.

Bir sunucu

(server), özel bir servisi gerçekleştiren bir süreçtir, örneğin, bir

72

Bir sunucu

(server), özel bir servisi gerçekleştiren bir süreçtir, örneğin, bir

dosya sistemi servisi veya bir veri tabanı servisi.

Bir istemci

(client), bir sunucu’dan ona bir istek

(request) göndererek bir

servisi istemektedir ve sonrasında sunucu’nun cevabı’nı (reply) beklemektedir.

Bu istemci-sunucu etkileşimi, istek-cevap davranışı olarak da bilinmektedir.

Şekil 1.25 ‘te görülmektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemciler ve Sunucular:

Fig. 1.25

73

Şekil 1.25 ‘te, bir sunucu ve bir istemci arasındaki genel etkileşim

gösterilmektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemciler ve Sunucular:

Bir istemci ve bir sunucu arasındaki iletişim bir çok yerel-alan ağında altta yatan

ağ oldukça güvenli olduğunda basit bir bağlantısız protokol ile

gerçekleştirilebilir.

74

Bu durumlarda, bir istemci bir servisi istediğinde, basitçe sunucu için bir mesajı

paketler, istediği servisi tanımlar, bunun yanı sıra gerekli girdi verisini de pakete

ekler.

Mesaj daha sonra sunucuya gönderilir. İkinci taraf (sunucu tarafı), daima

gelebilecek bir isteği beklemektedir, kendi sırası geldiğinde, bu isteği işleme tabi

tutar, elde ettiği sonuçları bir cevap mesajı içerisinde paketleyerek tekrar

istemciye geri yollar.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemciler ve Sunucular:

Bağlantısız (connectionless) bir protokol kullanımı etkin olmak gibi açıkça bir

avantaja sahiptir. Mesajlar kayıp olmadığı veya bozulmadığı sürece, istek/cevap

protokolü oldukça iyi çalışmaktadır. Ne yazık ki, tesadüfi (occasional) iletim

hatalarına karşı protokolü dirençli kılmak önemsiz de değildir. Yapılabilecek tek

75

şey, hiçbir cevap mesajı gelmediğinde isteğini tekrar göndermesini sağlamaktır.

Problem, bununla beraber, istemcinin orijinal istek mesajının kaybolduğunu

veya cevap iletimi’nin hatalı olduğunu tespit edememesidir.

Eğer cevap kaybolduysa, isteğin tekrar gönderilmesi, yapılacak işlemin iki kez

yapılması ile sonuçlanabilir. Eğer işlem şu örnekteki gibi ise: “10 bin TL ‘yi

banka hesabıma transfer et”, burada bir hata oluşması durumunda basitçe

raporlama ile belirtilmesi daha iyi olacaktır. Diğer yandan, eğer işlem şöyle ise:

“hesabımda kaç param kaldığını söyle”, o zaman isteğin tekrar

gönderilmesinde bir sorun yoktur. Bu örneklerden de anlaşılacağı gibi bu

problemlerin tek bir çözümü yoktur.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemciler ve Sunucular:

Bir alternatif olarak, bir çok istemci-sunucu sistemi güvenilir bir bağlantıtemelli

(connection-oriented) protokol kullanır. Her ne kadar bu çözüm

yerel-alan ağında görece düşük başarımı nedeniyle tamamen uygun olmasa da,

iletişimin güvensiz olduğu geniş

-alan sistemlerinde mükemmelen çalışmaktadır.

76

Örneğin, tüm Internet uygulama protokolleri güvenilir TCP/IP bağlantıları

temellidir. Bu durumda, bir istemci bir servisi istediğinde, isteğin

gönderilmesinden evvel bir bağlantıyı sunucuya kurar.

Sunucu genellikle, bağlantı red edildiğinde, aynı bağlantıyı cevap mesajını

göndermek için kullanır. Özellikle istek ve cevap mesajları küçük olduklarında,

bir bağlantının kurulması ve bozulması görece maliyetlidir.

Đstemci – Sunucu modeli (The Client – Server Model)

Uygulama katmanlama (Application layering):

İstemci-sunucu modeli, bir çok görüşme ve çatışmayı içermektedir. Ana

konulardan birisi de bir istemci ile bir sunucu arasında açık bir ayrımın nasıl

yapılacağıdır. Ortada aşikar bir ayrım da bulunmamaktadır.

77

Örneğin, dağıtık bir veri tabanı için bir sunucu sürekli bir istemci gibi

davranabilir, bunun nedeni veri tabanı tablolarının gerçekleştirilmesinden

sorumlu farklı dosya sunucuları için istekleri iletmesi olabilir.

Bu durumlarda, veri tabanı sunucusunun kendisi süreç sorgularından daha fazla

bir şeyi temelde yapmamaktadır.

Đstemci – Sunucu modeli (The Client – Server Model)

Uygulama katmanlama (Application layering):

Bir çok istemci-sunucu uygulaması veri tabanlarına kullanıcı erişimini

desteklemektedir, bir çok insan aşağıda verilen üç farklı seviyenin ayrımını

yapılmasını savunmaktadırlar:

1. Kullanıcı

- ara yüz seviyesi (The user

-interface level

)

78

1. Kullanıcı

- ara yüz seviyesi (The user

-interface level

)

2. İşlem seviyesi (The processing level)

3. Veri seviyesi (The data level)

Kullanıcı - ara yüz seviyesi, görünüm yönetimi gibi kullanıcı ile doğrudan ara

yüz işlevini yapılması için gereklidir. İşlem seviyesi tipik olarak uygulamaları

içermektedir. Veri seviyesi ise üzerinde uğraşılan geçerli veriyi içermektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

Uygulama katmanlama (Application layering) :

Kullanıcı-Arayüz seviyesi (User-Interface level) :

İstemciler tipik olarak kullanıcı-ara yüz seviyesini uygularlar. Bu seviye

programlardan oluşur, bu sayede son kullanıcının uygulamalarla etkileşimi

mümkün olur. En basit kullanıcı

-ara yüzü programı bir karakter

-tabanlı

79

mümkün olur. En basit kullanıcı

-ara yüzü programı bir karakter

-tabanlı

ekrandan başka bir şey değildir. Pull-down menüler, X-window ara yüzü, MSDOS PC ara yüzleri buna örnektir.

Đstemci – Sunucu modeli (The Client – Server Model)

Uygulama katmanlama (Application layering) :

İşlem seviyesi (Processing level) :

Bir çok istemci-sunucu uygulaması tamamen üç farklı parçadan oluşur: bir

parça bir kullanıcı ile etkileşimi idare eder, bir parça bir veri tabanı veya dosya

sistemi üzerinde işlem yapar ve bir ortanca parça bir uygulamanın çekirdek

80

fonksiyonelliğini içerir. Bu ortanca parça, mantıksal olarak işlem seviyesi’nde

(processing level) bulunmaktadır.

Buna örnek olarak, Internet arama motoru verilebilir. Kullanıcı arama

motoruna kelimeyi yazar. Büyük bir veri tabanından aradığı kelimeye uygun

sonuçlar bulmak için, veri tabanı sorguları şeklinde bir çok veri tabanına aradığı

kelime gönderilir. Bulunan sonuçlar sıralanır

(ranking) ve html olarak

kullanıcıya sunulur.

Đstemci – Sunucu modeli (The Client – Server Model)

Uygulama katmanlama (Application layering) :

İşleme seviyesi (Processing level) :

Fig. 1.28

81

Şekil 1.28 ‘de, üç farklı katmanda bir Internet arama motoru’nun (search engine) genel

organizasyonu görülmektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

Uygulama katmanlama (Application layering) :

Veri seviyesi (Data level) :

İstemci-sunucu modelinde, veri seviyesi

(data level) uygulamaların üzerinde

işlem yaptığı geçerli veriyle uğraşan programları içerir. Bu veri seviyesi’nin en

önemli özelliği verilerin kalıcı

(persistent) olmasıdır. Bunun anlamı, hiçbir

82

önemli özelliği verilerin kalıcı

(persistent) olmasıdır. Bunun anlamı, hiçbir

uygulama çalışmasa da, verinin bir yerde bir sonraki kullanım için

depolandığı/saklandığıdır. Bunun en basit biçimi bir dosya sistemidir, fakat en

genel kullanımı tam bir veri tabanıdır. İstemci-sunucu modelde, veri seviyesi

tipik olarak sunucu tarafından gerçekleştirilir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemci-Sunucu Mimarileri (Client-Server Architecture) :

Basit organizasyon sadece iki tip makineye sahiptir:

1. Bir istemci makine, sadece kullanıcı-ara yüz seviyesi (parçası) gerçekleştirimi

yapan programlar içerir.

2. Bir sunucu makine, işlem ve veri seviyesi gerçekleştirimi yapan programlar

83

içerir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemci-Sunucu Mimarileri (Client-Server Architectures) :

Çok taraflı (basamaklı) mimariler (Multitiered Architectures) :

Sunucu ve istemci için sadece fiziksel olarak iki-taraflı

(two-tiered) mimari

vardır. Fakat sunucunun bazen istemci gibi davranması gerekebilir.

84

Böyle durumlarda buna üç-taraflı

(three-tiered) mimari denilir. Bu üç taraflı

mimari’de işlem seviyesi’nin parçası olan programlar başka bir sunucuda

bulunmaktadır, fakat istemci ve sunucu makineler arasında kısmen dağıtılmış

olabilirler. En tipik üç-taraflı mimari örneği işlemi işlemek’tir (transaction

processing).

Şekil 1.29 ‘da istemci-sunucu organizasyonları görülmektedir.

Şekil 1.30 ‘da istemci gibi davranan bir sunucu şematik olarak görülmektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemci-Sunucu Mimarileri (Client-Server Architectures) :

Çok taraflı (basamaklı) mimariler (Multitiered Architectures) :

Fig. 1.29

85

Şekil 1.29 (a) ‘dan (e) şıkkına kadar, alternatif istemci-sunucu organizasyonları

görülmektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemci-Sunucu Mimarileri (Client-Server Architectures) :

Çok taraflı (basamaklı) mimariler (Multitiered Architectures) :

Fig. 1.30

86

Şekil 1.30 ‘da, bir istemci gibi davranan bir sunucu görülmektedir.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemci-Sunucu Mimarileri (Client-Server Architectures) :

Çok taraflı (basamaklı) mimariler (Multitiered Architectures) :

Modern mimariler (Modern architectures):

Bir çok iş alanında dağıtık işleme, istemci-sunucu uygulama organizasyonu olan

çok

-taraflı

(multitiered) mimariye eşittir (denktir). Buna dikey dağıtım

87

çok

-taraflı

(multitiered) mimariye eşittir (denktir). Buna dikey dağıtım

(vertical distribution) denilir. Bunu başarmak için mantıksal olarak farklı

bileşenleri (farklı makinelerdeki) yerleştirmek gerekir. Düşey parçalama

(vertical fragmentation), dağıtık ilişkisel veri tabanlarında kullanılır,

buradaki tablolar kolon-bazlı olarak bölümlenerek bir çok makine arasında

dağıtılır. Dikey dağıtım, istemci-sunucu uygulamalarında sadece tek yoldur.

Bunun haricinde, yatay dağıtım

(horizontal distribution) vardır, buna

göre bir istemci veya sunucu, mantıksal olarak denk iki parçaya ayrılır, fakat her

bir parça kendi bütün veri kümesi üzerinden işlem yapar, bu yükün

dengelenmesini sağlar (load balancing).

Şekil 1.31 ‘de Veb servislerinin yatay dağıtımının bir örneği görülmektedir.

Peer-to-peer

(eşten eşe) terimi; her düğümün hem sunucu hem de istemci

olduğu durumlarda kullanılır.

Đstemci – Sunucu modeli (The Client – Server Model)

İstemci-Sunucu Mimarileri (Client-Server Architectures) :

Çok taraflı (basamaklı) mimariler (Multitiered Architectures) :

Fig. 1.31

88

Şekil 1.31 ‘de Veb servislerinin yatay dağıtımının bir örneği görülmektedir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

89

Ders’in sonu…

90

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 4

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 4: İletişim,

Katmanlı protokoller,

Uzak Yordam Çağrısı

(Remote Procedure Call , RPC).

Đletişim

• İşlemciler arası iletişim tüm dağıtık sistemin kalbidir. Dağıtık sistemlerde

iletişim her zaman düşük-seviyeli mesaj geçme (gönderme) ‘nin altta yatan ağ

tarafından sağlanılması temellidir. Modern dağıtık sistemler, sıklıkla yüzlerce

veya milyonlarca işlemci’nin dağınık şekilde üzerinde bulunduğu güvensiz

Internet ağındadır. İşlemcilerin iletişiminde protokoller ve bu protokollerin

3

yapısını oluşturan katman biçimleri vardır.

• Dağıtık sistemler için bir model RPC (Uzak Yordam Çağrısı, Remote

Procedure Call) olarak adlandırılır. Bir RPC mesaj geçme’nin (gönderme)

karışıklığının gizlenmesini hedefler ve istemci-sunucu uygulamaları için

ideal’dir. RPC modeli’nin daha da gelişmişi olarak RMI (Uzak Yöntem

Çağrısı, Remote Method Invocation) bulunur; RMI dağıtık nesne

tabanlıdır. Düşük-seviyeli iletişimin (bilgisayar ağları’nın) dağıtım saydamlığına

etkisi vardır (kötü yönde) bu yüzden daha yüksek-seviyeli mesaj geçme iletişimi

kullanmak uygun olmaktadır.

• MOM (Mesaja-yönelik ara katman yazılımı, Message-Oriented

Middleware) ise, mesaj geçme iletişiminde kullanılmaktadır.

Katmanlı protokoller

• Paylaşımlı bellek yüzünden dağıtık sistemdeki tüm iletişim mesaj alışverişi ile

(düşük-seviye) olmaktadır. İletişimde IBM ‘in EBCDIC ve Amerikan ASCII

karakter set’leri kullanılmaktadır. İletişim için en genel katmanlı refereans

model ISO OSI ‘dir. Burada;

▫ ISO (Uluslar arası Standartlar Organizasyonu, International Standards

4

Organization),

▫ OSI (Açık Sistemler Ara bağlantısı, Open Systems Interconnection)

anlamına gelmektedir.

• OSI modeli, açık sistem iletişimi için oluşturulmuştur. Mesaj alış-verişindeki

kurallar protokoller olarak biçimlendirilmiştir. Ağ üzerinden iletişme geçmek

isteyen bilgisayarlar bu protokolleri kullanmalıdır. İki genel protokol tipi

bulunmaktadır. Bağlantı-yönelimli (connection-oriented) ve bağlantısız

(connectionless). Bunlardan bağlantı-yönelimli olanda gönderici ve alıcı veri

alışverişinden önce bir bağlantı kurmalı ve hangi protokolü kullanacaklarını

görüşmelidir. İşleri bitince bağlantıyı koparmaları gerekir. Telefon bağlantıyönelimli bir sistemdir. Bağlantısız protokoller kurulum gerektirmemektedir.

Gönderici ilk mesaj hazır olduğunda onu gönderiri. Mektup-posta kutusu buna

örnektir.

Katmanlı protokoller

Fig. 2.1

5

Şekil 2.1 ‘de OSI modelindeki Katmanlar, Ara yüzler ve protokoller görülmektedir.

Katmanlı protokoller

• OSI modelinde iletişim için

7 katman bulunur. Her bir katman mesajın baş

kısmına header

(başlık) bilgisini ekler. Şekil 2.2 ‘de, Ağ üzerinde gözüken tipik

bir mesaj gösterilmektedir. Böyle protokoller topluluğuna, protocol suite

(protokol demeti) veya protocol stack

(protokol yığıtı) denilmektedir.

6

Fig. 2.2

ekil 2.2 ‘de a

ğ üzerinden gönderilen bir mesajın yapısı

şematik olarak verilmi

ştir.

Katmanlı protokoller

Düşük-seviye protokolleri (Lower-level protocols):

3 adet düşük seviyeli OSI katmanı bulunur, bu üç katman beraberce bir

bilgisayarı ağ’a bağlarlar.

1. Fiziksel-katman (Physical Layer):

Bu katman

0 ve

1 ‘lerin gönderilip alınması ile ilgilenir. Bu katman elektriksel,

mekaniksel ve sinyalleşme ara yüzlerinin standartlaştırılması ile ilgili protokolü

7

içerir.

2. Veri bağı katmanı (Data link layer):

Fiziksel katman sadece bit’leri gönderir. Hataya karşı önlemi yoktur. Belirli bir

mekanizma ile hata tespiti ve düzeltmesi yapılmalıdır. Bu mekanizma veri bağı

katmanı’nın ana görevidir. Böylece bit gruplarını, çerçeve

(frame) denilen

birimlere çevirerek, her bir çerçeve‘nin doğru alınması sağlanılır. Her bir

çerçeve’nin başına ve sonuna özel bir bit örüntüsü

(pattern) ekleyen veri bağı

katmanı, böylece çerçeveleri işaretlemiş ve bir sağlama toplamı

(checksum)

hesaplamış olur.

Veri bağı katmanı, çerçevelere sağlama toplamını ekler. Bir çerçeve alıcıya

vardığında, alıcı veriden sağlama toplamını tekrar hesaplayıp, çerçeve ile gelen

sağlama toplamı ile karşılaştırır. Eğer ikisi uyuşuyor ise, çerçeve doğru olarak

algılanır ve kabul edilir. Eğer uyuşmuyorlarsa alıcı göndericiden çerçeveyi tekrar

göndermesini ister. Çerçevelerin başlarında sıra numaraları vardır.

Katmanlı protokoller

• Şekil 2.3 ‘te veri bağı katmanı’ndaki bir gönderici ile bir alıcı arasındaki görüşme

görülmektedir. Fig. 2.3

8

ekil 2.3 ‘te veri ba

ğı katmanı’ndaki bir gönderici ile bir alıcı arasındaki görü

şme

şematize edilmi

ştir.

Katmanlı protokoller

Düşük-seviye protokolleri (Lower-level protocols):

3. Ağ katmanı (Network Layer):

Bir mesajın göndericiden alıcıya seçilecek en uygun yoldan gitmesi bir sorun

teşkil eder. Bu en iyi yolun seçilmesine yönlendirme (routing) denilir ve ağ

katmanının ana görevidir. En kısa yol en iyi yoldur, çünkü yol ne kadar uzarsa

trafik o kadar artar ve kuyruktaki mesaj sayısı çoğalmasına bağlı olarak

gecikmeler de oluşur. Bazı yönlendirme algoritmaları yükteki değişimlere ayak

9

uydurmayı dener.

IP (Internet Protocol), ağ katmanının bağlantısız (connectionless) protokolüdür.

IP paketi ağ katmanı için mesaja verilen isim) gönderimi için hiçbir kuruluma

gerek yoktur. Her bir IP paketi hedefine diğerlerinden bağımsız olarak ulaşır. Bir

iç yol (internal path) seçilmez ve hatırlanmaz.

Bir bağlantı-yönelimli

(connection-oriented) protokol, ATM ağlarındaki

sanal kanal sanal kanal

(virtual virtual channel channel) olmaktadır. Bir ATM ‘deki sanal kanal, bir ) olmaktadır. Bir ATM ‘deki sanal kanal, bir tek

-

yönlü

(unidirectional) bağlantıdır (kaynaktan hedefe) ve bir çok ortamdan

geçer (ATM switch’lerden). Iki host arasında her bir sanal kanalı kurmak yerine

bir sanal yol

(virtual path) denilen sanal kanal topluluğunda gruplandırılır.

Bir sanal yol önceden tanımlı olan ve iki host arasındaki yoldur.

Katmanlı protokoller

Taşınım protokolleri (Transport protocols):

Taşınım katmanı fonksiyonu (The Function of the Transport Layer):

Paketler göndericiden alıcıya giderken yolda kaybolabilir. Bazı uygulamalar

kendi hata kurtarma ve güvenli iletişimlerine sahiptir. Transport katmanının ana

görevi bu servisi sağlamaktır. Bir mesaj alındığında, taşınım katmanı (transport

layer) onu iletime uygun daha küçük parçalara ayırır, her birine bir sıra numarası

verip ondan sonra onları yollar. Transport katmanı başlığında (header) hangi

10

paketin gönderildiği, hangisinin alındığı, daha ne kadarının alınabileceği,

hangisinin tekrar gönderilmesi gerektiği belirtilir.

Güvenli (güvenilir, reliable) transport bağlantıları (connection-oriented

anlamında), bağlantısız veya bağlantı-yönelimli servisler üzerine kurulurlar. İlk

durumda tüm paketler düzgün sırada yerlerine varırlar. Bazen de diğerlerinden

farklı yol izleyen bir paket gönderimden önce hedefe varır.

Internet Transport protokolüne TCP

(Taşınım Kontrol Protokolü,

Transmission Control Protocol) denilir. Bunun TCP/IP kombinasyonu

bulunur, IP ‘nin UDP

(Evrensel Datagram Protokolü, Universal

Datagram Protocol) protokolü bulunmaktadır. UDP bağlantısız ve güvenilmez

(unreliable) ‘dir. Ayrıca ISO’nun beş tane farklı hata yakalamyla alakalı transport

protokolü bulunur, bunlar TP

0 ‘dan TP

4’e kadar derecelidir. Bunların haricinde

gerçek-zamanlı veriler için RTP

(Gerçek-zamanlı Taşınım Protokolü,

Real-time Transport Protokol) bulunur.

Katmanlı protokoller

Taşınım protokolleri (Transport protocols):

İstemci-Sunucu TCP (Client-Server TCP):

İstemci-Sunucu etkileşimi dağıtık sistemlerde sıkılıkla taşınım protokolleri

kullanılarak yapılır. Popüler olarak TCP kullanılır. TCP ‘nin UDP ‘ye göre

avantajı güvenilir (reliable) olmasıdır. Başarım açısından UDP daha önemlidir.

TCP senkronize istek-cevap davranışını bir çok istemci-sunucu etkileşmesinde

desteklememektedir.

11

Normal şartlarda mesaj kaybolmaz. TCP yapısının istemci-sunucu etkileşiminde

kullanımı aşağıdaki şekilde gösterilmektedir.

Aşağıdaki Şekil 2.4 ‘te TCP’nin normal işlemi ve İşlemsel TCP görülmektedir.

Katmanlı protokoller

Taşınım protokolleri (Transport protocols):

İstemci-Sunucu TCP (Client-Server TCP):

Fig. 2.4

12

ekil 2.4 ‘te, a) TCP’nin normal i

şlemini, b) İşlemsel TCP ‘yi göstermektedir.

Katmanlı protokoller

Taşınım protokolleri (Transport protocols):

İstemci-Sunucu TCP (Client-Server TCP):

Şekil 2.4 ’te a) şıkkına bakacak olursak;

İlk önce istemci bir bağlantı kurulumunu başlatır. Bu üç yönlü bir el sıkışmadır.

(Bunun protokolüdür). Bu protokol iki tarafında mesaj sıra numaraları üzerinde

anlaşması ilkesine dayanır. İstemci, bağlantı kurulduktan sonra doğrudan

isteğini gönderir ve takip eden paket sunucuya bağlantıyı kapatmasını söyler.

13

Sunucu hemen “acknowledgement” (anlaşıldı/kabul edildi) mesajı ile

istemciye cevap verir, bunun arkasına (ACK mesajının) kuyruk olarak eklenen

piggybacked bir mesajla bağlantı kapatılır. Böylece sunucu daha sonra isteklere

cevap vermez.

Şekil 2.4 ’te b) şıkkına bakacak olursak;

Az maliyetli olarak hemen gelen isteğe karşılık cevap gönderilip, bağlantıyı

koparmak mümkündür. Bu tarz protokole TCP for Transactions (İşlemler için

TCP) yani T/TCP adı verilir.

T/TCP protokolü, TCP’yi geliştirmek için tasarlanmıştır. T/TCP eğer karşı taraf

aynı protokole sahip değilse normal TCP şeklinde çalışır.

Katmanlı protokoller

Yüksek-seviyeli protokoller (Higher-level protocols):

Oturum ve Sunum Protokolleri (Session and Presentation Protocols):

Oturum (Session) katmanı temelde taşınım katmanının genişletilmiş bir

versiyonudur. Diyalog kontrolü ile hangi tarafın şu an konuştuğunu belirler ve

senkronizasyonu sağlar. Senkronizasyon uzun transferler için oldukça uygundur.

Böylece çökme (crash) durumunda bir önceki konuma dönülebilir.

14

Daha düşük seviyelerin aksine gönderici ve alıcı’nın bit’lerinin güvenli ve etkin

alıp-verilmesi yerine, Sunum (Presentation) katmanı bu bit’lerin anlamı ile

ilgilenmektedir. Bir çok mesaj rasgele bit’lerden oluşmaz daha anlamlı (örneğin

kişi ismi, adres para miktarı gibi verilerden oluşur. Sunum katmanı böyle

alanların kayıtlarını tanımlayabilir ve göndericinin alıcıya böyle bir bilginin

varlığını bildirmesini sağlayabilir.

Katmanlı protokoller

Yüksek-seviyeli protokoller (Higher-level protocols):

Uygulama Protokolleri (Application Protocols):

OSI referans modeli bakış açısıyla, hemen hemen tüm dağıtık sistemler sadece

uygulamalardır. Bu modelin tek eksiği uygulamalar ile uygulamaya özgü

protokoller arasındaki ayrımdır. Örneğin, FTP

(Dosya Transfer Protokolü,

File Transfer Protocol), istemci ile sunucu arasında dosya transferini sağlar.

Protokol’ün ftp programı ile uğraşması gerekmez, keza program son

-kullanıcı

15

(end-user) uygulamasıdır.

Diğer bir örnek ise HTTP

(Üstmetin Transfer Protokolü, HyperText

Transfer Protocol), web sayfalarının yönetilmesi ve sayfa transfer idaresi ile

ilgilenir. Protokol, veb gezgin’leri (browser), veb sunucuları (server) gibi

uygulamalarla çalışır. Bununla beraber, HTTP sadece veb bağlantıları için

kullanılmak haricinde örneğin JAVA RMI ile HTTP üzerinden istek (request)

gönderimi yapılabilir.

Katmanlı protokoller

Yüksek-seviyeli protokoller (Higher-level protocols):

Ara katman yazılımı Protokolleri (Middleware Protocols):

Ara katman yazılımı, uygulama katmanında bulunan bir uygulamadır, bu

uygulama bir çok genel-amaçlı protokolü içerir. Yüksek seviyeli iletişim

protokolleri ve bir çok çeşit ara katman yazılımı servisinin kurulmasını sağlayan

protokolleri ayırt etmek gerekir.

16

Bir çok protokol ara katman yazılımı çeşitliliğini sağlar, örneğin yetkilendirme vs.

Yetkilendirme (Authentication) protokolü kullanıcı ve süreçlerin erişimlerini

doğası gereği uygulamadan bağımsız düzenler.

Başka bir örnek ise atomik yapıda olma (atomicity) ile verilebilir. Protokollerin

yönettiği süreç gruplarının belirli işlemleri yapması, tüm işin hepsini yapmaması

anlamına gelmektedir. Bu anlam ile, işlemler (transactions) için kullanılır.

(devam)

Katmanlı protokoller

Yüksek-seviyeli protokoller (Higher-level protocols):

Ara katman yazılımı Protokolleri (Middleware Protocols):

Bazı ara katman yazılımı iletişim protokolleri taşınım (transport) katmanına ait

olacak seviyededir, fakat bazı nedenlerden daha yüksek katmanlarda yer alırlar.

Örneğin güvenli çoğa yayım (multicast) servislerinin ölçeklenebilirliğinin garanti

edilmesi için uygulamalarda kullanılırlar.

17

OSI modeli ile karşılaştırma yapılabilmesi için aşağıdaki Şekil 2.5 ‘te görüldüğü

gibi, oturum ve sunum katmanı, tek bir ara katman yazılımı katmanı ile yer

değiştirilmiştir ve bu katman uygulamadan bağımsız protokolleri içerir. Bu

protokoller daha düşük seviyedeki katmanlara ait değillerdir. Orijinal taşınım

servisleri bir ara katman yazılımı servisi olarak önerilebilir (modifiye edilmeden).

Bu yaklaşım UDP ‘yi taşınım seviyesinde önermeye benzemektedir. Ara katman

iletişim servisleri mesaj-geçme (gönderme) servislerini içerebilir.

Şekil 2.5 ‘te, ağ üzerinden iletişim için uyarlanmış bir referan , ağ üzerinden iletişim için uyarlanmış bir referans modeli şematik s modeli şematik

olarak görülmektedir.

Katmanlı protokoller

Yüksek-seviyeli protokoller (Higher-level protocols):

Ara katman yazılımı Protokolleri (Middleware Protocols):

Fig 2.5

18

ekil 2.5, a

ğ üzerinden ileti

şim için uyarlanmı

ş bir referans modeli.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Bir çok dağıtık sistem, süreçler arasında açık mesaj alışverişini taban alır.

Bununla beraber, gönder

(send) ve al

(receive) prosedürleri iletişimi

gizlemezler. Bunun önemi dağıtık sistemler için erişim saydamlığı

(access

transparency) sağlanması ile ilgilidir. Buna çözüm olarak, bir A makinesindeki

süreçten B makinesindeki prosedür (yordam) çağrıldığında, A makinesindeki

arayan süreç (bekleyerek, suspend) B makinesindeki prosedürün (çağrılan

19

prosedür) çalıştırılmasın bekler. Bilgi arayandan aranılana parametre olarak

gönderilir. Bu basit yapı programcıya görünür değildir. Bu yöntem Uzak

Yordam Çağrısı

(RPC) olarak bilinir.

Çağıran ve çağrılan prosedürler farklı makinededirler, farklı adres uzaylarında

çalıştırılmaktadırlar, bu durum karışıklığa neden olmaktadır. Gönderilen

parametre ve sonuçlar, eğer makineler benzer (aynı) değilse, (mimarileri farklı

olabilir) işi karmaşıklaştırırlar. Sonuçta her iki taraftaki makineler çökebilir ve

farklı farklı hatalara neden olabilir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

Geleneksel Yordam Çağrısı (Conventional Procedure Call) :

Burada geleneksel teriminin anlamı tek bir makine anlamına gelmektedir.

Örneğin, aşağıdaki gibi bir çağrı olsun:

20

count=read(fd, buf, nbytes); (C dilinde)

dosya Tampon (karakter dizisi) Kaç byte okunaca

ğını tutan

integer

Şekil 2.6 ‘ya bakacak olursak; yerel bir prosedür çağrısında parametre geçme

olayı gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

Geleneksel Yordam Çağrısı (Conventional Procedure Call) :

21

ekil 2.6 ‘da, yerel bir prosedür ça

ğrısında parametre geçme olayı gösterilmektedir; a) Okuma (read) ça

ğrısından önce yı

ğıt (stack) durumu b) Ça

ğrılan prosedür aktif iken yı

ğıt (stack) durumu

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

Geleneksel Yordam Çağrısı (Conventional Procedure Call) :

Okuma (read) bittikten sonra, geri dönüş değeri bir kayıtçı (yazmaç, register)’a

atanır ve geri dönüş adresi silinir ve arayana tekrar kontrol geri verilir. Arayan

(caller) parametreleri yığıttan siler ve orijinal durumuna geri döner. Ayrıca call

-

22

by-value

(değer ile çağırma) ve call-by-reference

(referans ile çağırma)

durumları Şekil 2.6 ‘daki (b) şıkkında duruma uygun olabilir. Bir diğer parametre

geçme mekanizması da call-by-copy/restore

(kopya/yedek ile çağırma)

olarak anılmaktadır.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

İstemci ve Sunucu Koçanları (Client and Server Stubs) :

RPC’nin temelinde yatan fikir bir uzak prosedürü (yordam) çağrısını olabildiğince

yerel bir çağrıya benzetmektir. Diğer bir deyişle, RPC ‘yi saydam yapmaktır.

Çağrıyı yapan prosedür, çağrılan prosedürün çalıştırıldığı (başka bir makinede

23

veya tam tersi) yerden habersiz olmalıdır. Örneğin bir programın bazı veriyi bir

dosyadan okumak istediğini varsayalım.

Programcı, veriyi almak için kod içine okuma (read) çağrısını yerleştirir.

Geleneksel (tek-işlemcili) sistemde, okuma (read) rutini, bağlayıcı (linker)

tarafından kütüphane’den (library) çıkartılır ve nesne programına yerleştirilir.

Bu kısa bir prosedürdür, bu genelde denk bir okuma (read) sistem çağrısının

çağrılması ile gerçekleştirilir. Bir sistem çağrısı olarak okuma (read) için

parametreler yığıt’a konulur (push), programcı okuma (read) çağrısının alsında

temelde ne yaptığından habersizdir.

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

İstemci ve Sunucu Koçanları (Client and Server Stubs) :

RPC, saydamlığı benzer yoldan başarır. Eğer okuma (read) gerçekte bir uzak

prosedür ise, okuma (read ‘in farklı bir versiyonu istemci koçanı

(client stub)

olarak adlandırılır ve kütüphane (library) içerisine konulur.

24

Orijinal çağrıdaki gibi yerel işletim sistemi için bir çağrı yapılır. Orijinal’e

benzemeyen ise, işletim sistemine verisini vermek için sormamasıdır. Bunun

yerine, parametreleri bir mesaj ve istekler olarak paketler ve sunucuya

yollanacak mesaj haline getirir. Sonrasında gönder (send) çağrısı takip eder ve

istemci koçanı (client stub), al (receive) ‘ı çağırır, cevap gelene kadar kendisini

bloklar (bekler).

Eğer bir mesaj sunucuya ulaşırsa, sunucunun işletim sistemi onu bir sunucu

koçanı’na (server stub) iletir. Bir sunucu koçanı, istemci koçanı’nın sunucutarafındaki dengidir. Bu, ağ üzerinden gelen istekleri yerel prosedür çağrılarına

dönüştüren bir parça kod’dur. Tipik olarak sunucu koçanı, al ‘ı (receive) çağırır

ve gelecek mesajlar için bloklanmış olur.

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

İstemci ve Sunucu Koçanları (Client and Server Stubs) :

Sunucu koçanı mesajdan parametreleri açar (unpack) ve sunucu prosedürünü

normal yoldan çağırır. Sunucu bakış açısından, tüm çağrı istemci tarafından

doğrudan yapılmış olur. Parametreler ve geri dönüş adreslerinin tamamı

25

yığıtta’dır (yani nereye ait iseler ve hiçbir şey olağandışı gözükmez). Sunucu

görevini yapar ve sonucu çağrıyı yapana geri döndürür.

Eğer sunucu koçanı, çağrı tamamlandıktan sonra kontrolü geri ele geçirirse,

sonucu (tamponu) bir mesaja paketler ve gönder (send) çağrısı ile istemciye

gönderir. Daha sonra da al (receive) ile bir sonraki mesajı beklemeye başlar.

Mesaj tekrar istemci makineye gelirse, istemcinin işletim sistemi bu mesajın

istemcideki süreci gösterdiğini (addressed) görür. (Burada (veya) istemci

koçanı’nı gösterdiğini görür, fakat işletim süreci aradaki farkı anlamaz).

Mesaj, bekleme tamponuna kopyalanır ve istemci sürecinin kilidi kaldırılır

(unblocked). Böylece istemci koçanı mesajı inceler, sonucu açar (unpack) ve

çağırana kopyalar ve normal yoldan geri döner (Normal yerel çağrı gibi sonuç

üretir anlamında).

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

İstemci ve Sunucu Koçanları (Client and Server Stubs) :

Özetleyecek olursak; bir RPC aşağıdaki adımlardan oluşur:

1-) İstemci süreci, istemci koçanı’nı (client stub) normal yoldan çağırır.

2

-) İstemci koçanı, bir mesaj oluşturur ve yerel işletim sistemini çağırır.

26

2

-) İstemci koçanı, bir mesaj oluşturur ve yerel işletim sistemini çağırır.

3-) İstemcinin işletim sistemi (OS), uzaktaki işletim sistemine (OS) mesajı gönderir.

4-) Uzaktaki işletim sistemi (OS), mesajı sunucu koçan’a verir.

5-) Sunucu koçan parametreleri açar ve sunucuyu çağırır.

6-) Sunucu görevini yapar ve sonucu sunucu koçan’a (server stub) geri döndürür.

7-) Sunucu koçan bunu bir mesaja paketler ve kendi yerel işletim sistemini (OS)

çağırır.

8-) Sunucunun işletim sistemi (OS), istemcinin işletim sistemine (OS) mesajı

gönderir.

9-) İstemci’nin işletim sistemi (OS), mesajı istemci koçan’a verir.

10-) Koçan, sonucu açar (unpack) ve bunu istemciye döndürür.

Aşağıda Şekil 2.7 ‘de, bir istemci ile sunucu programı arasındaki RPC işleyişi

prensibi şematik olarak gösterilmektedir.

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Temel RPC işlemi (Basic RPC operation):

İstemci ve Sunucu Koçanları (Client and Server Stubs) :

Fig. 2.7

27

ekil 2.7 ‘de, bir istemci ile sunucu programı arasındaki RPC i

şleyi

şi prensibi

gösterilmektedir. (Bu tarz i

şleti

şim Senkronize (bloklamalı) tarz ileti

şimdir).

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Parametre geçme/gönderme (Parameter passing):

Değer parametrelerinin geçirilmesi (Passing Value Parameters) :

Parametrelerin bir mesaja paketlenmesine parameter marshaling

denilmektedir. Örneğin bir prosedür olan topla(i, j) uzak prosedürü

i ve

j

integer’larının toplamını alacak olsun. Şekil 2.8 ‘e bakacak olursak, istemci koçan

i ve

j parametrelerini alarak mesaja yerleştirir. Daha sonra istemci OS ‘tan

28

i ve

j parametrelerini alarak mesaja yerleştirir. Daha sonra istemci OS ‘tan

geçerek ağ bağlantısına gönderilir.

Ağ bağlantısı ile mesaj sunucuya ulaşınca sunucu OS onu sunucu koçan’a aktarır.

Sunucu koçan hangi prosedürün çağırıldığını anlamak için mesajı açar (unpack).

Daha sonra sunucu ilgili prosedürü çalıştırır, bu işlem bitince sunucu koçan

tekrar kontrolü ele geçirir ve sonucu mesaj olarak paketler, bu mesaj tekrar ağ

üzerinden istemci koçan’a gönderilir. Sunucu ve istemci makineler aynı ise (aynı

mimarili ise) tüm parametreler ve sonuçlar skaler tipli, integer karakterler için

sorun olmaz, iyi çalışmaktadır.

Bununla beraber, büyük dağıtık sistemlerde farklı mimariye sahip makineler

olabilir. Örneğin Intel mimarili istemci ile SPARC mimarili sunucu veya tam tersi

gibi. Bu durumda örneğin karakter seti farklı olabilir. Bu yüzden hata

oluşmaması için, kısaca integer’lar tersine çevrilmiş byte sırasında, fakat

string’lerin sırasına dokunmanın mümkün olmadığı bir problemle karşılaşılır ve

bunun halledilmesi gerekir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Parametre geçme/gönderme (Parameter passing):

Değer parametrelerinin geçirilmesi (Passing Value Parameters) :

Fig. 2.8

29

Şekil 2.8’de, uzak bir hesaplama’nın RPC üzerinden yapılmasını içeren adımlar

şematik olarak gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Parametre geçme/gönderme (Parameter passing):

Değer parametrelerinin geçirilmesi (Passing Value Parameters) :

Şekil 2.9 ‘da bir mesajın farklı mimarilerde nasıl gösterildiğine örnek verilmiştir.

Fig. 2.9

30

ekil 2.9’da,

a) Pentium makinedeki orijinal mesaj,

b) SPARC makinedeki alınan mesaj,

c) Mesajın çevrildikten sonraki hali, küçük kutulardaki (kesikli çizgili)

sayılar her bir byte’ı i

şaret eder. Görüldü

ğü gibi sıralama SPARC ’ın ki

gibi ama alınan de

ğerler Pentium’ unki gibidir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Parametre geçme/gönderme (Parameter passing):

Parametre belirleme ve Koçan Yaratma (Parameter Specification and

Stub Generation) :

RPC protokolü tanımıyla istemci ve sunucu koçanlar için belirli ara yüz

farklılıkları oluşmaktadır. Bunun için ara yüzler genel anlamı ile Ara yüz

31

Tanımlama Dili

(Interface Definition Language, IDL) ile yapılması daha

sonra uygun olarak istemci koçan’ı veya sunucu koçan’ı için derlenmelidir.

Şekil 2.10 ‘da, a) bir prosedür ve b) bunun ile ilgili mesaj ve formatı verilmiştir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Parametre geçme/gönderme (Parameter passing):

Parametre belirleme ve Koçan Yaratma (Parameter Specification and

Stub Generation) :

32

Fig. 2.10

ekil 2.10 ‘da, a) bir prosedür ve b) bunun ile ilgili mesaj ve formatı gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Doors (Kapılar):

Orijinal RPC modeli çağrıyı yapan ve çağrıların iletişimini ağ üzerinden

mesajlaşma olarak varsayar. Bununla beraber, istemci ve sunucuyu aynı

makinede varsayarsal, bu varsayımla çelişir. Normalen, yerel süreçlerarası

33

iletişim

(interprocess communication, IPC) ile aynı makinedeki

süreçlerin işletim sistemi ile çalışması mümkündür.

Yerel IPC, ağ olanaklarından daha etkindir (eğer aynı makinedeki süreçler için

kullanılıyorsa). Buna karşın farklı iletişim mekanizmaları, başarım söz konusu

olduğunda birleştirilerek sürecin aynı makinede olup olmamasına bağlı olarak

uygulanabilmektedir.

Birkaç işletim sistemi RPC ‘nin aynı makinedeki süreçler için olan eş değerlisi

doors ’u (kapılar’ı) önerir. Bir door, sunucuda yerleşmiş bulunan süreç

tarafından çağrılan bir sunucu sürecinin adres uzayındaki bir prosedür için

jenerik bir isimdir. Doors orijinal olarak Spring işletim sistemi için

tasarlanmıştır.

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Doors (Kapılar): (devam)

Doors’un ana faydası, tek bir mekanizmanın (prosedür çağrısı) bir dağıtık

sistemde iletişim için kullanılmasına izin vermesidir. Ne yazık ki, uygulama

geliştiriciler, yerel süreç içinden yapılan bir çağrıdan haberdar olmak

34

zorundadır, aynı şekilde yerelden farklı bir sürece (aynı makine veya uzaktaki

makine) haberdar olmak zorundadırlar.

Şekil 2.11 ‘de, Doors ‘u süreçlerarası iletişim (IPC) için kullanma prensibi

gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Doors (Kapılar): (devam)

35

Fig. 2.11

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Asenkron RPC (Asynchronous RPC) :

Geleneksel prosedür çağrısı olarak bir istemci bir uzak prosedürü çağırdığında

istemci bir cevap gelene kadar (return) kadar bloklanır. Bu kısıtlama geri

dönecek bir sonuç olmadığında gereksizdir. Bu sırada istemci daha yararlı bir

36

işle uğraşabilir.

Buna benzer durumlar için RPC sistemler asenkron RPC isimli bir imkan

sunarlar, buna göre bir istemci, RPC isteğinin oluşmasının ardından hemen

işine devam eder. Asenkron RPC ile sunucu, RPC isteği alındığı anda

istemciye cevabını gönderiri, daha sonra da istenilen prosedür’ü çağırı. Cevap,

sunucunun RPC sürecine geçtiği bir onaylama (acknowledgement) olarak

istemciye davranır. İstemci, yakında sunucunun onayını (acknowledgement)

alana kadar bloklanmadan işine devam eder.

Şekil 2.12 ‘de, a) Geleneksel RPC ‘de istemci ve sunucu arasındaki etkileşim,

b) Asenkron RPC ‘de istemci ve sunucu arasındaki etkileşim şematik olarak

görülmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Asenkron RPC (Asynchronous RPC) :

37

Fig 2.12

ekil 2.12 ‘de, a) Geleneksel RPC ‘de istemci ve sunucu arasındaki etkile

şim,

b) Asenkron RPC’de istemci ve sunucu arasındaki etkile

şim, görülmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Asenkron RPC (Asynchronous RPC) :

Asenkron RPC, cevap döndüğü sırada; istemcinin onu beklemeye hazır

olmadığı durumda ve hiçbir şey yapmadığı durumda (cevap gelene kadar bir

şey yapmıyorsa) kullanışlıdır.

38

Örneğin, bir istemci host kümesi’nin ağ adreslerini (network) almak

isteyebilir, böylece onlara yakındaki bir zamanda bağlanmak isteyebilir. Bir

isimlendirme servisi bu adresleri toplarken, istemci diğer şeyleri yapmak

isteyebilir. Böyle durumlarda, istemci ile sunucu arasındaki senkronizasyonu

iki asenkron RPC yapar. İstemci önce sunucuyu arar, host listesini alır (bu

listeye bakılmış olur) ve sunucu bu listenin alıcısını onaylamışken, devam

eder. İkinci çağrı sunucu tarafından yapılır, sunucu istemciyi listede

bulunduğu adreslere göre çağırır. Böyle iki asenkron RPC ’nin kombine

edilmesine deferred synchronous RPC

(ertelemeli senkronize RPC)

denilir. İstemci’nin, sunucunun onayını beklemediği (acceptance of

request) RPC ‘ler de vardır. Bunlara tek-yönlü

(one-way) RPC ‘ler denilir.

Şekil 2.13 ‘de, iki asenkron RPC sayesinde istemci ve sunucu etkileşimi

şematik olarak gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

RPC ‘ler ara katman yazılımı temeli ve dağıtık sistemlere genelde geniş çapta

uyarlanmıştır. Böyle özel bir RPC sistem olarak Dağıtık Hesaplama

Ortamı

(Distributed Computing Environment, DCE) bulunmaktadır.,

39

bunu Açık Yazılım Vakfı (Open Software Foundation, OSF) oluşturmuştur (Şu

an Open Group olarak OSF bilinmektedir). DCE diğer RPC sistemler kadar

popüler değildir, fakat şartnameleri (spesifikasyonları) Microsoft’un temel

sistemine uygundur.

DCE ‘ye giriş:

DCE gerçek bir ara katman yazılım sistemidir ve var olan (ağ) işletim sistemi

ile dağıtık uygulamalar arasında bir soyutlama katmanı olarak çalışır. İlk

olarak UNIX için tasarlanmış daha sonra VMS ve Windows NT gibi işletim

sistemlerine taşınmıştır (port edilmiştir). DCE ‘nin paröası olan belirli sayıda

servisleri vardır. Dağıtık dosya servisi, sistemdeki (aynı yoldaki) herhangi bir

dosyaya saydam bir yoldan erişimi dünya çapında dosya sistemi olarak sağlar.

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

DCE ‘ye giriş: (devam)

Dizin servisi (Directory service) ise sistemdeki tüm kaynakların takibini sağlar.

Güvenlik servisi (Security service) ise yetkili kişilerin sınırlandırılmış erişim

40

yapabilmesi ve korumayı sağlar. Dağıtık zaman servisi ise farklı makinelerde

global olarak senkronize edilmiş saatleri oluşturmayı dener. (Yani gelen

veriler için consistency

(tutarlılık) elde etmeye çalışır).

DCE’nin amaçları:

DCE RPC ‘nin ana amacı, RPC sistemin bir istemci için uzaktaki bir servise en

basit olarak erişmesi için yerel bir prosedürün çağrılmasını sağlamasıdır. Bu

arayüz istemci için (uygulama vs…) programların basit bir yoldan yazılmasını

sağlar. RPC sistemi, doğru sunucuyu otomatik olarak yerleştirir ve sonra

istemci ve sunucu yazılımlar arasında iletişimi kurar (buna binding

denilmektedir). Her iki yöne olan mesaj iletimini idare eder,

bölümlendirilmelerini ve tekrar bir araya getirilmelerini de sağlar. RPC

sistemi, istemci ile sunucu arasındaki veri tipi dönüşümünü de idare eder

(farklı mimarideki makineler için farklı byte sırası oluşturulur).

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir istemci ve bir sunucunun yazılması (Writing a Client and a

Server):

DCE RPC sistemi, diller, kütüphaneler, arkaplan süreci

(daemon) ve

41

DCE RPC sistemi, diller, kütüphaneler, arkaplan süreci

(daemon) ve

faydalı araç (utility tool) programları gibi belirli sayıda bileşenden oluşur.

Bunların hepsi, istemci ve sunucu yazılmasını olanaklı kılar. Bir RPC istemci

ve sunucusunun yazımı ve kullanımının tüm süreci aşağıdaki Şekil 2.14 ‘de

mevcuttur. Şekil 2.14 ‘te DCE RPC ‘de bir istemci ve bir sunucu yazımındaki

adımlar görülmektedir.

Bir sunucu-istemci sisteminde her şeyi bir arada tutan ara yüz tanımıdır. Ara

yüz Tanımlama Dili

(Interface Definition Language, IDL) ile

tanımlanır. ANSI C dilindeki fonksiyon prototiplerine benzer prosedür

deklarasyonlarına izin verir. IDL dosyaları tip tanımları, sabit deklarasyonları

ve diğer bilgileri içerir, bunlar parametrelerin doğru şekilde paketlenmesi

(marshal) ve sonuçların açılması (unmarshal) için gereklidir. Her IDL

dosyasının vazgeçilmez elemanı, global olarak eşsiz (unique) olan belirli bir

ara yüz tanımlayıcıdır.

Uzak Yordam Cağrısı (RPC)

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir istemci ve bir sunucunun yazılması (Writing a Client and a Server): (devam)

42

Fig 2.14

İstemci/Sunucu

yazılımının ismi:

uuidgen olarak verilmi

ş. Şekil 2.14 ‘te DCE RPC ‘de bir istemci ve bir sunucu yazımındaki adımlar görülmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir istemci ve bir sunucunun yazılması (Writing a Client and a

Server): (devam)

İstemci bir tanımlayıcıyı ilk RPC mesajında gönderir ve sunucu onun

43

doğruluğunu kontrol eder. Bu yolla, eğer bir istemci yanlışlıkla yanlış bir

sunucuyu bağlamak isterse veya doğru sunucunun eski bir versiyonuna

bağlanmak isterse, sunucu hatayı tespit eder ve bağlama gerçekleşmez.

Ara yüz tanımları ve eşssiz tanımlayıcılar DCE ile alakalıdır. Bir

istemci/sunucu uygulamasının yazımında (uuidgen programı diye bilinir) ilk

adım bir IDL prototip dosyası içeren bir ara yüz tanımlayıcının (uuidgen

tarafından herhangi bir yerde yaratılmış herhangi bir ara yüzde tekrar

kullanılmadığını garanti eden) yaratılmasıdır. Eşsiz ‘lik yaratım zamanının ve

yerinin kodlama (encode) tarafından garanti edilmesi ile sağlanır.

İkinci adım, IDL dosyasının düzenlenmesidir, uzak prosedürlerin isimlerinin

yazılması, parametrelerinin yazılması gibi. RPC tam oalrak saydam değildir.

Örneğin istemci ve sunucu global değişkenleri paylaşmaz, fakat IDL kuralları,

desteklenmeyen yapıların ortaya çıkartılmasını imkansız hale getirir (sınırları

belirler). (devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir istemci ve bir sunucunun yazılması (Writing a Client and a

Server): (devam)

Eğer IDL dosya tamamlandıysa, IDL derleyici (compiler) onu işlemek üzere

44

çağırılır. IDL derleyici’nin çıktısı üç dosyadan oluşur:

1-) Bir başlık dosyası (örneğin C dili için ”interface.h” gibi)

2-) İstemci koçanı (Client stub

)

3-) Sunucu koçanı (Server stub

)

Başlık dosyası eşsiz

(unique) tanımlayıcı, tip tanımları, sabit tanımları ve

fonksiyon prototipleri içerir. Bunlar hem istemci, hem de sunucu kodunda

Ortak bir “Header” dosyası

kullanırlar.

içerilmelidirler.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir İstemci’nin bir Sunucuya bağlanması (Binding a Client to a

Server) :

Bir istemcinin bir sunucuyu aramasını (call) mümkün kılmak için sunucunun

45

kayıt edilmiş ve gelen çağrıları kabul etmeye hazırlanmış olması gerekir. Bir

sunucu’nun yerinin tespiti ve ona bağlamayı (bind) mümkün kılar.

Sunucu yerleşimi iki adımdan oluşur:

1-) Sunucu makineyi yerleştir (locate)

2-) Sunucuyu (doğru süreci) bu makineye yerleştir (locate)

İstemci, sunucu ile iletişimdeyken uç noktayı (endpoint) bilmek ister. Farklı

süreçlerden gelen mesajları ayırt etmek için sunucu’nun işletim sistemi

tarafından böyle uç nokta’lar (genelde port olarak bilinir) kullanılır.

(devam)

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir İstemci’nin bir Sunucuya bağlanması (Binding a Client to a

Server) : (devam)

DCE ’de bir çiftler tablosu (sunucu, uç nokta) her bir sunucu makinede DCE

46

daemon (arka plan süreci) denilen bir süreç tarafından oluşturulur/bakılır.

Gelen istekler için uygun olmadan önce (tablo), sunucu bir uç nokta için

işletim sistemine sormak zorundadır. DCE daemon ile bu uç noktayı daha

sonra sunucu kaydeder. DCE daemon bu bilgiyi kaydeder (record)

[sunucunun konuştuğu protokolleri içerir], bu bilgi ileriki kullanımlar içinuç

nokta tablosu’ndadır.

Sunucu, dizin servisi ile kendinin bulunduğu makine ağ adresini ve hangi isim

ile göründüğünü kaydeder (Ağ adresi ve makine ismi). Daha sonra istemci bu

sunucuya aşağıdaki şekildeki gibi bağlanır;

Şekil 2.15 ‘te, DCE ‘de istemci-sunucu bağlama (binding) şematik olarak

gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir İstemci’nin bir Sunucuya bağlanması (Binding a Client to a

Server) : (devam)

47

Fig. 2.15

Şekil 2.15 ‘te, DCE ‘de istemci-sunucu bağlama (binding) şematik olarak

gösterilmektedir.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir İstemci’nin bir Sunucuya bağlanması (Binding a Client to a

Server) : (devam)

Örnek: Bir istemci bir video sunucusuna erişmek istiyor olsun. Adresi:

48

/local/multimedia/video/movies/

olsun.

İstemci bu ismi dizin sunucusu’na (directory server) aktarır. Dizin sunucusu

ana video sunucu’nun ağ adresini geri döndürür. İstemci sonra DCE daemon’a

(video sunucudaki, yani artık iyice bilinen uç noktadaki) gider ve video

sunucu’nun kendi uç nokta tablosu’ndaki uç noktaya bakmasını söyler/sorar.

Bu bilgi ile donanmış RPC göreve başlar, sonraki RPC ‘ler için (çalışmalar)

bakma (lookup) gerekmez. DCE böylece daha karmaşık aramaların uygun

sunucular için istemci tarafından yapılabilmesini sağlar.

Uzak Yordam Cağrısı (RPC)

Uzak Yordam Çağrısı (Remote Procedure Call, RPC):

Genişletilmiş RPC modelleri (Extended RPC models):

Örnek: DCE RPC (Distributed Computing Environment RPC) :

Bir RPC ‘nin yapılması (Perform an RPC):

DCE birkaç anlamsal seçenek sağlar. Varsayılan, en-fazla-bir-kez-işlem

(at

-most

-once

-operation) ‘dir, bunda hiçbir çağrı bir seferden fazla

49

(at

-most

-once

-operation) ‘dir, bunda hiçbir çağrı bir seferden fazla

taşınmayarak sistem çökmelerinin önüne geçilmek istenmiştir. Bir uzak

prosedürü eşgüçlü

(idempotent) olarak (IDL dosyası içinde) tanımlamak

da mümkündür, bu durumda hiçbir aksaklık olmadan bir çok kez tekrar

çalıştırılabilir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

50

Ders’in sonu…

51

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 5

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 5: Uzak Nesne Çağrısı

(Remote Object Invocation, ROI),

Mesajlaşmaya-dayalı iletişim.

Uzak Nesne Çağrısı (Remote Object Invocation)

• Nesne-tabanlı teknoloji, değerini dağıtık olmayan uygulamaların

geliştirilmesinde kanıtlamıştır. Nesne’nin en önemli özelliği (aspect),

içindekileri dış dünyadan saklamasıdır, bunu iyi tanımlı ara yüzü ile yapar.

Böylece ara yüz aynı kaldığı sürece nesneler kolaylıkla değiştirilebilir veya

uyarlanabilir.

3

Dağıtık Nesneler (Distributed Objects) :

Bir nesnenin anahtar özelliği veriyi zarflaması’dır (encapsulate), buna durum

(state) denilir ve bu veri üzerindeki işlemlere yönetmler (methods) denilir.

Yöntemler ara yüzler sayesinde elde edilebilir/kullanılabilir hale getirilirler.

Şunu anlamak gerekir ki; bir nesnenin durumu’na erişmek ve yönetme’nin, bir

nesne ara yüzü sayesinde elde edilebilen yöntemlerin çağrılması yolu dışında,

hiçbir yolu yoktur. Bir nesne, bir den çok ara yüzü gerçekleştirebilir.

Bu ayrım (ara yüz ve nesne gerçekleştirimi) dağıtık sistemler için önemlidir.

Bununla, bir nesne başka makinedeyken ara yüz’ün başka makinede

olduğu durum kastedilmektedir. Buna genellikle dağıtık nesneler

(distributed object) denilir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Dağıtık Nesneler (Distributed Objects) : (devam)

Şekil 2.16 ‘da, istemci-taraflı vekil (proxy) ile uzak bir nesnenin genel

organizasyonu şematik olarak görülmektedir.

Fig. 2.16

4

Uzak Nesne Çağrısı (Remote Object Invocation)

Dağıtık Nesneler (Distributed Objects) : (devam)

Bir istemci, bir dağıtık nesneye bağlandığında (binds) nesnenin ara yüzü’nün

bir gerçekleştirimi (buna vekil

(proxy) denilir) istemcinin adres uzayına

yüklenir. Bu vekil, RPC sistemlerdeki bir istemci koçanı’na (client stub)

benzerdir. Bunun yaptığı, tek şey yöntem invokasyon ‘larını (çağrılarını) mesaja

5

paketlemek (marshal) ve cevap mesajlarını açarak (unmarshal) istemci için

yöntem invokasyon’larının (çağrıları) sonucunu geri döndürmektir.

Geçerli nesne sunucu makinededir, bu nesne istemci makinedeki ile aynı ara

yüze sahiptir. Gelen invokasyon (çağrı istekleri ilk önce bir sunucu koçanı’na

(server stub) gönderilir, buna sıklıkla iskelet

(skeleton) denilmektedir.

Burada sunucunun nesne ara yüzünde bulunan doğru yöntem invokasyon’ları

(çağrıları) için mesaj paketi açılır (unmarshal). Sunucu koçanı ayrıca,

cevapların paketlenmesi (marshaling replies) ve istemci-tarafındaki vekil için

cevap mesajlarının iletilmesinden de sorumludur.

Dağıtık nesnelerin, durum’ları (state) dağıtık değildir. Sadece ara yüz

gerçekleştirimi nesne tarafından diğer makinede yapılmaktadır. Böyle

nesnelere uzak nesneler

(remote objects) denilir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Dağıtık Nesneler (Distributed Objects) : (devam)

Kalıcı ve Geçici Nesneler (Persistent and Transient Objects) :

Bir kalıcı nesne

(persistent object), bir sunucu sürecin adres uzayında şu

an bulunmasa dahi var olmaya devam eden nesnedir.

6

Aksine bir geçici nesne

(transient object), sunucunun (bu nesneyi)

yönettiği sürece (zaman boyunca) var olan bir nesnedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Bir İstemciyi Bir Nesneye Bağlamak (Binding a Client to an Object) :

▫ Örtülü bağlama (Implicit Binding) : İstemci, basit bir mekanizma önerir; sadece

bir nesne’nin bir ara yüzünü kullanarak doğrudan yöntem çağırmaya (invoke) izin

vermektir.

▫ Belirtilmiş bağlama (Explicit binding) : İstemci, önce özel bir fonksiyonu

7

nesne’ye bağlamak için çağırır, böylece (yani bağlanma sonrası) onun yöntemlerini

çağırabilir (invoke).

Statik ve Dinamik Uzak Yöntem Çağrıları’nın karşılaştırılması:

Bir nesneye bir istemci bağlandığında, onun yöntemlerini vekil üzerinden

çağırır. Böyle bir Uzak Yöntem Çağrısı

(Remote Method Invocation,

RMI), temelde RPC ‘ye benzerdir. Eğer önceden-tanımlı ara yüz tanımları

kullanılıyorsa buna statik çağrı

(static invocation) denilir.

Örnek

1: DCE Uzak Nesneleri:

Bir dağıtık dinamik nesne, bir istemcide yerel olarak bir sunucunun yarattığı

bir nesnedir. Ayrıca dağıtık isimli nesneler de bulunur.

Şekil 2.18 ve Şekil 2.19 ‘da bu konular ile ilgili şematikler görülmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Şekil 2.18 ‘de, bir nesne’nin referans veya değer ile geçirilmesi sırasındaki

durum şematik görülmektedir.

Fig. 2.18

8

Uzak Nesne Çağrısı (Remote Object Invocation)

Şekil 2.19 ‘da, Dağıtık nesne’ler ile ilgili şematik görülmektedir.

Fig. 2.19

9

a) DCE ‘deki da

ğıtık dinamik nesneler.

b) Da

ğıtık isimlendirilmi

ş nesneler.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli iletişim (Message-oriented communication) :

RPC ‘ye uzak nesne çağrıları, dağıtık sistemlerde iletişimin gizlenmesini sağlar,

bunu erişim saydamlığını geliştirerek yaparlar. Ne yazık ki, hiçbir mekanizma

daima uygun değildir. Çalışma zamanları önemlidir, örneğin RPC ve RMI ‘ın

senkronize doğası gereği istemcinin isteği işlenen kadar istemci bloklanır.

10

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) :

Uygulamalar daima host’lar da çalıştırılır, burada her bir host’un iletişim

sistemi için ara yüzü bulunduğu ve bunlar ile mesajları ilettiği bilinir. Örneğin

bir elektronik posta sistemi bu tasarıma benzer.

Her bir host bir posta sunucu’suna (mail server) bağlanır, kullanıcının

host’undaki ara yüz, özel bir hedefe mesajları göndermesi için kullanıcı

etmenine izin verir. Bir kullanıcı etmeni iletim için kendi host’unda bir mesajı

gönderdiğinde, host ilk olarak mesajı kendi yerel posta sunucusu’na iletir ve

buna da çıktı tamponu’nda geçici olarak mesaj depolanır.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) :

Bir posta sunucu, bir mesajı kendi tamponundan çıkararak, hedefe bakar.

Hedefe bakmak (taşınım-seviyesi), mesajın gönderileceği posta sunucusu’nun

adresini döndürür. Posta sunucu daha sonra bir bağlantı kurarak mesajı hedef

11

posta sunucu’ya gönderir.

İkinci posta sunucusu genel mesajı girdi tamponunda depolar (ilgili alıcı için).

Eğer hedef posta sunucu (geçici olarak) ulaşılamaz durumdaysa (yani kapalıysa

vs.) yerel posta sunucu mesajı depolamaya devam eder.

Alıcı host’daki ara yüz, alıcının kullanıcı etmeni için bir servisi önerir, böylece

düzenli olarak gelen posta’ları kontrol eder. Kullanıcı etmeni, yerel posta

sunucu’daki kullanıcının posta kutusunda doğrudan işlem yapabilir, fakat bir

çok durumda kendi host’unun yerel tampon’a yeni mesajları kopyalar.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) :

Kalıcı iletişim

(Persistent commmunication) ile, bir mesaj iletim için

gönderildiğinde iletişim sistemi tarafından alıcısına teslim edile kadar

depolanır. Buna e

-posta sistemi örnek verilebilir.

12

Geçici iletişim (Transient communication) ile, bir mesaj gönderen ve

alan uygulamalar çalıştığı müddetçe iletişim sistemi tarafından depolanır.

Tipik olarak tüm taşınım-seviyesi iletişim servisleri sadece geçici iletişimi

önerir. Bu durumda, bir iletişim sunucusu geleneksel bir depola-ve-ilet

yönlendirici

(store-and-forward router) ile ilgilidir. Eğer bir

yönlendirici (router) bir mesajı bir sonraki yönlendiriciye teslim edemezse

(veya hedef host’a), mesajı düşürür (drop)/yok eder.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

Asenkron iletişim (Asynchronous communication): Karakteristik

olarak, bir gönderici (sender) mesajını iletim için gönderdikten sonra hemen

işine devam eder. Bunun anlamı, mesaj, gönderici host’daki yerel tamponda

13

depolanmasıdır, aksi halde ilk önce iletişim sunucusunda depolanır.

Senkron iletişim (Synchronous communication): Gönderim, kendi

mesajı alıcı host’un yerel tamponunda depolanan veya alıcıya (henüz) teslim

edilene kadar blokludur. Bunun örneği, alıcı gelen mesajı işleyene kadar

göndericinin bloklu olduğu durumdur.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

Kalıcı

(persistent), geçici

(transient), senkronize

(synchronous) ve

asenkron

(asynchronous) iletişimin bir çok kombinasyonu mevcuttur.

14

Bunlar için başlıca 6 farklı durum söz konusudur.

1-) Kalıcı asenkronize iletişim (Persistent asynchronous communication

)

2-) Kalıcı senkronize iletişim (Persistent synchronous communication

)

3-) Geçici asenkronize iletişim (Transient asynchronous communication

)

4-) Alım-tabanlı geçici senkronize iletişim (Receipt-based transient

synchronous communication

)

5

-) Teslimat Teslimat

-tabanlı geçici senkronize iletişim ( tabanlı geçici senkronize iletişim (Delivery Delivery

-based transient based transient

synchronous communication

)

6-) Cevap-tabanlı geçici senkronize iletişim (Response-based transient

synchronous communication

)

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

1-) Kalıcı asenkron iletişim (Persistent Asynchronous

Communication) :

15

Bir mesaj, yerel hos’taki tamponda kalıcı (persistent) olarak saklanılır veya

ilk iletişim sunucusunda saklanılır. Elektronik posta tipik olarak bu tarzdır.

Şekil 2.22 (a) şıkkında, kalıcı asenkron iletişim şematik olarak

gösterilmektedir.

2-) Kalıcı senkronize iletişim (Persistent Synchronous

Communication) :

Mesajlar, sadece alıcı host’ta kalıcı olarak saklanabilir. Bir gönderici, kendi

mesajı alıcının tamponunda saklanılıncaya kadar bloklanır. Fakat alıcı

uygulamanın kendi yerel host’unda mesajı saklamak (depolamak) için

çalışması gerekmez. Bu tarz iletişimin daha zayıf formu, göndericinin kendi

mesajının alıcı host’a bağlanmış iletişim sunucusunda saklanıncaya kadar

göndericinin bloklanmasıdır.

Şekil 2.22 (b) şıkkında, kalıcı senkron iletişim şematik olarak

gösterilmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

Fig. 2.22 (a) , (b)

16

a) Kalıcı asenkron iletişim (Persistent asynchronous communication)

b) Kalıcı senkron iletişim (Persistent synchronous communication)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

3-) Geçici asenkron iletişim (Transient Asynchronous

Communication) :

17

Bu tarz iletişim tipik olarak, UDP gibi taşınım-seviyesi datagram servisleri

tarafından önerilir. Bir gönderici uygulama bir mesajı iletilmesi için

gönderdiğinde, gönderen host’un yerel tamponunda bu mesaj geçici olarak

saklanılır, hemen sonrasında gönderici işine devam eder.

Bu sırada, iletişim sistemi hedefe doğru mesajı yönlendirir (alıcıya teslim

edilmesi nerede bekleniyorsa) ve hedefte yerel bir tamponda saklanması

muhtemeldir. Mesaj alıcının host’una ulaştığı anda alıcı çalışmıyorsa iletim

durur (fails). Bu tarz iletişime asenkron RPC ‘ler örnektir.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

4-) Alım-tabanlı senkronize iletişim (Receipt-based Transient

Communication) :

18

Geçici senkronize iletişimin bir çok biçimi vardır. Bunlardan zayıf olanı,

mesaj alımına (receipt) dayalı olandır. Gönderici, mesaj alıcı host’un yerel

tamponunda saklanılıncaya kadar bloklanır. Gönderici bir kabul

(acknowledgement) alır ve işine devam eder.

Şekil 2.22 (c) ve (d) şıklarında, sırasıyla geçici asenkron iletişim, alım-tabanlı

geçici senkron iletişim görülmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

19

Fig. 2.22 (c) , (d)

c) Geçici asenkron iletişim (Transient asynchronous communication)

d) Alım-tabanlı geçici senkron iletişim (Receipt-based transient synchronous

communication)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

5-) Teslimat-tabanlı geçici senkronize iletişim (Delivery-based

Transient Synchronous Communication) :

20

Bu tarz iletişimde, gönderici, mesaj ileriki süreçler için alıcıya teslim edilene

kadar bloklanır. Buna örnek teşkil eden asenkron RPC ’de ileriki süreçler

için, istemcinin isteği kabul edilene kadar bekleyerek istemcinin sunucu ile

senkron olması bulunmaktadır.

6-) Cevap-tabanlı geçici senkronize iletişim (Response-based

Transient Synchronous Communication) :

Güçlü biçim olarak cevap

-tabanlı geçici senkronize iletişim, gönderici diğer

taraftan bir mesaj alana kadar bloklanır (aynı istemci-sunucu istek-cevap

davranışından olduğu gibi). RPC ’ler ve RMI ‘lar buna uyar.

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

21

Fig. 2.22 (e) , (f)

e) Mesaj tesliminde Teslimat-tabanlı geçici senkronize iletişim (Delivery-based

transient synchronous communication at message delivery)

f) Cevap-tabanlı geçici senkronize iletişim (Response-based transient

synchronous communication)

Uzak Nesne Çağrısı (Remote Object Invocation)

İletişimde Kalıcılık ve Senkronizasyon

(Persistence and

Synchronicity in Communication) : (devam)

Kalıcı iletişme ihtiyaç, ara katman yazılımı geliştiricilerinin uygulamarını

büyük

-ölçekli entegrasyonu ve dağılmış ara bağlantılı ağlarda kullanma

22

isteğidir. Böyle ağlar farklı yönetimsel alanlara dağılmıştır ve her zaman

erişime açık değillerdir (Yani sınırlı erişim vardır).

Geçici iletişimde ise bir hata oluştuğunda hemen gizlenilir ve kurtarma

prosedürü başlatılır. Kalıcı hata saydamlığı olarak bir tamir (sonradan

yapılan) kabul edilemez. Bununla beraber, kalıcı iletişimde uygulamalar,

gönderim ve alım arasında uzun gecikmeleri idare edecek şekilde

tasarlanmıştır.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Geçici İletişim (Message-oriented Transient

Communication) :

Berkeley soketleri (Berkeley sockets) :

Soket ara yüzleri çeşitleri bulunmaktadır. Berkeley Unix, XTI (X/Open

23

Transport Interface) gibi.

Bir soket, altta yatan ağ üzerinden gönderilecek veriyi yazan ve gelen veriyi

okuyan bir uygulamadır. Bir soket geçerli iletişim uç noktası (bu yerel işletim

sistemi tarafından özel bir taşınım protokolü için kullanılır) üzerinden bir

soyutlamadır.

Sunucu, “socket” ilkelini çağırarak yeni bir iletişim uç noktası yaratır. Daha

sonra “bind” ilkeli ile yerel bir adres yeni yaratılan sokete atanır/ilişkilendirilir.

Sonra “listen” ilkeli sadece bağlantı-yönelimli iletişim durumunda çağrılır, bu

bloklanmamış (non-blocking) bir çağrıdır. En son olarak “accept” çağrısı,

çağrıyı yapanı bir bağlantı isteği ulaşana kadar bloklar. Bir istek ulaştığında

yerel işletim sistemi yeni bir soket yaratır (orijinali ile aynı özelliklerde olan) ve

bunu çağrıyı yapana geri gönderir.

İstemci tarafında ise, “connect”, “send”, “receive” ve “close” ilkelleri bulunur.

Mesaj-yönelimli Geçici İletişim (Message-oriented Transient

Communication) :

Berkeley soketleri (Berkeley sockets) : (devam)

Đlkel Anlamı

Uzak Nesne Çağrısı (Remote Object Invocation)

24

(Primitive)

Socket Yeni bir iletişim uç noktası yaratır.

Bind Yerel bir adresi bir sokete ekler.

Listen Bağlantıları kabul etmek için gönüllülüğü ilan eder.

Accept Bir bağlantı isteği ulaşana kadar arayanı bloklar.

Connect Bir bağlantı kurmayı aktif olarak dener.

Send Bağlantı üzerinden bazı veriyi gönderir.

Receive Bağlantı üzerinden bazı veriyi alır.

Close Bağlantıyı serbest bırakır.

Fig. 2.23. TCP/IP için soket ilkelleri.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Geçici İletişim (Message-oriented Transient

Communication) :

Berkeley soketleri (Berkeley sockets) : (devam)

25

Fig. 2.24

ekil 2.24 ‘te, Soketler kullanılarak yapılan ba

ğlantı-yönelimli ileti

şim örüntüsü

(connection-oriented communication pattern).

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Geçici İletişim (Message-oriented Transient

Communication) :

Mesaj-Geçme Ara yüzü (The Message-Passing Interface, MPI) :

Yüksek başarımlı çoklu bilgisayarlarda kullanılır. Möesaj-geçme (gönderme)

26

için bir standart tanımlanmıştır. Mesaj-geçme ara yüzü (MPI), paralel

uygulamalar için tasarlanmıştır (geçici (transient) iletişim).

MPI iletişimi bir süreçler grubu (bilinen) içinde yer alır. Her bir grup için bir

tanımlayıcı atanmıştır. Bir grup içindeki süreç için bir (yerel) tanımlayıcı

atanmıştır.

Bu yerel tanımlayıcı,

A(grupID, süreçID)

şeklindedir.

MPI ’ın iletişim ilkelleri ile geçici iletişim yapılır, fakat senkronize iletişim

desteklenmemektedir. Bu tarz senkronize iletişim yapılması için özel komutlar

kullanılır.

Đlkel (Primitive) Meaning

MPI\_bsend Yerel bir gönderim tamponuna giden mesajı ekle.

Fig. 2.25

27

MPI\_send Bir mesajı gönder ve yerel veya uzaktaki tampona kopyalanana kadar bekle.

MPI\_ssend Bir mesajı gönder ve alım başlayana kadar bekle.

MPI\_sendrecv Bir mesajı gönder ve cevap için bekle.

MPI\_isend Referansı giden mesaja ekle ve devam et (işine).

MPI\_issend Referansı giden mesaja ekle ve alım başlayana kadar bekle.

MPI\_recv Bir mesaj al, eğer ortada bir tane bile yoksa bloklan.

MPI\_irecv Gelen bir mesaj var mı kontrol et, fakat bloklanma.

ekil 2.25 ‘te, MPI için mesaj geçme ilkellerinin bazıları gösterilmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Mesaj-kuyruk sistemleri veya Mesaj-yönelimli Ara katman Yazılımı (Messageoriented Middleware, MOM) oldukça önemlidir. Mesaj-kuyruk sistemleri, kalıcı

asenkron iletişim için destek sunar. Bunun anlamı ortamlar arası depolama

28

kapasitesini mesajlar için, alıcı ve gönderici mesaj iletimi sırasında aktif olmasa

dahi sağlamasıdır.

Berkeley soketleri ve MPI ’dan en önemli farkı mesaj transferi için saniye veya

milisaniyeler yerine dakikalar seviyesinde izin vermesidir.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Bir mesaj-Kuyruklama Sistemi’nin Genel Mimarisi (General

Architecture of a Message-Queuing System) :

29

Bir kuyruk “kaynak kuyruk” (source queue) olarak adlandırılır. Mesajın

iletileceği kuyruğa “hedef kuyruk” (destination queue) denilir. Bir çok makine

arasında dağıtık kuyruklar topluluğu bulunur, bir mesaj-kuyruklama

sisteminde, mesajların transfer edileceği ağ yerleşimleri için kuyrukların bir

haritası bulunmaktadır.

Bu kuyruk isimlerinin bulunduğu (muhtemelen dağıtık) bir veri tabanı

olmalıdır. Kuyruklar, kuyruk yöneticileri tarafından yönetilirler. Bazı özel

kuyruk yöneticileri yönlendirici (router) gibi davranır (relays) gelen mesajları

diğer kuyruk yöneticilerine yollarlar. Böylece bir mesaj-kuyruk sistemi

uygulama seviyesinde, bindirilmiş ağ’da (overlay network) yani var olan bir

bilgisayar ağı’nın üstündedir.

(devam)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent Communication) :

Bir mesaj-Kuyruklama Sistemi’nin Genel Mimarisi (General Architecture of a MessageQueuing System) :

Fig. 2.28

Uzak Nesne Çağrısı (Remote Object Invocation)

30

ekil 2.28 ‘de, a

ğ seviyesinde adresleme ve kuyruk-seviyesinde adresleme

arasındaki ili

şki.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Mesaj Takasçıları (Message Brokers) :

Mesaj-kuyruk sistemlerinde, farklı formatlar arasında genel bir mesaj formatı

için çevrimleri (conversions) yapan/idare eden özel düğümler (nodes)

31

için çevrimleri (conversions) yapan/idare eden özel düğümler (nodes)

bulunmaktadır. Bunlar, mesaj takasçıları (message brokers) olarak (takasçı veya

kırımcı) bilinirler. Bu mesaj takasçısı (broker) uygulama seviyesinde ağ geçidi

olarak mesaj-kuyruk sisteminde çalışır. Ana amacı, gelen mesajları hedef

uygulamanın anlayabileceği bir formata dönüştürmektir.

Bu arada, mesaj takasçısı’nın sadece başka bir uygulama olduğunu unutmamak

gerekir, yani mesaj-kuyruk sisteminin bir iç parçası değildir. Mesaj takasçısı

mesaj için bir tekrar formatlayıcı ’dır (reformatter). Bir mesaj takasçısı’nın

kalbinde bir kurallar veri tabanı yatmaktadır, böylece

X formatındaki bir mesajın

nasıl

Y formatına çevrilebileceği belirtilir.

Asıl problem kuralları tanımlamaktır. Bir çok mesaj takasçısı karmaşık kural

üretim aletlerine sahiptir, fakat alt tarafta hala kurallar veri tabanına elle

girilmektedirler. Kurallar özel bir çevrim dili ile formüle edilebilirler, fakat bir

çok mesaj takasçısı program çevrimlerinde normal programlama dillerinin

kullanımına izin verir.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Mesaj Takasçıları (Message Brokers) : (devam)

32

Fig. 2.30

ekil 2.30, Bir mesaj-kuyruklama sistemindeki bir mesaj takasçısı’nın genel

organizasyonu görülmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

IBM ’in ana çatı’larında kullanılan MQSerisi temel mimarisi kuyruk ağı’dır. Tüm

kuyruklar, kuyruk yöneticileri tarafından yönetilir. Bir kuyruk yöneticisi, kendi

33

gönderim kuyruklarından mesajların silinmesinden sorumludur.

Benzer olarak, bir kuyruk yöneticisi gelen mesajların, altta yatan ağ’dan alınıp

idare edilmesinden ve her bir mesajın uygun girdi kuyruğuna depolanmasından

sorumludur. Kuyruk yöneticileri çift taraflı olarak mesaj kanalları (message

channels) üzerinden bağlıdırlar. Bunlar taşınım-seviyesi (transport-level)

soyutlama sağlar.

Bir mesaj kanalı iki yönlü, güvenilir bağlantıyı gönderici ve alıcı kuyruk

menajerleri arasında sağlar. Bir mesaj kanalının her bir ucunu bir mesaj kanal

etmeni (mesaj kanal etmeni [MCA]) yönetir. Gönderici MCA gönderim

kuyruğunu kontrol ederek, onu bir taşınım-seviyesi paketine çevirip yollar, alıcı

MCA ise gelen mesajları dinler (gelen paketleri), gelen paketleri açar

(unwarping) sonra ilgili kuyrukta depolar.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) : (devam

)

34

Fig. 2.30

ekil 2.31 ‘de, IBM ‘in MQSerisi mesaj-kuyruklama sisteminin genel organizasyonu

görülmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

Kuyruk yöneticileri, kuyrukları yönetmek için uygulama olarak aynı süreç içinde

bağlantılandırılabilir (linked). Bu durumda, standart bir ara yüz arkasındaki

35

uygulamadan kuyruklar gizlenir, fakat etkin olarak uygulama tarafından

doğrudan yönetilirler.

Alternatif bir organizasyon ise kuyruk yöneticileri ile uygulamaların farklı

makinelerde olduğu biçimdir. Bu durumda, uygulama aynı ara yüzü önerir, bunu

kuyruk yöneticisinde aynı makinede olmasından dolayı yapar. Bununla beraber,

ara yüz bir vekil olarak gerçekleştirilirse, geleneksel RPC kullanarak kuyruk

yöneticisiyle (senkronize olarak) iletişime geçer. Bu durumda IBM MQSerisi bir

uygulamanın yerel kuyruklara erişebildiği bir model haline gelir.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

Kanallar (Channels) :

Her bir MCA, ilişkili öznitelikler/özellikler (associated attributes) kümesine

36

sahiptir. Bu özellikler kanalın davranışını tanımlar. MCA öznitelikleri gönderici

ve alıcı arasında uyumlu (compatible) olmalı ve kanal kurulmadan önce

görüşülmelidir.

Öznitelik

(Attribute) Tanım (Description)

Taşınım tipi Kullanılacak taşınım (transport) protokolünü tanımlar.

FIFO teslimatı Mesajların gönderildikleri sırada teslim (delivery) edilmesini gösterir.

Fig. 2.32

FIFO teslimatı Mesajların gönderildikleri sırada teslim (delivery) edilmesini gösterir.

Mesaj uzunluğu Tek bir mesajın maksimum uzunluğu.

Kurulum tekrar

deneme sayısı Uzaktaki MCA’nın başlatılması için maksimum deneme sayısını belirtir.

Delivery retries Alınmış mesajı kuyruğa yerleştirmek için maksimum kez MCA’nın denemesi (kaç kez olacağı)

ekil 2.32 ‘de, mesaj kanal etmenleri ile ilgili bazı özellikler görülmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

Mesaj iletimi (Message transfer) :

Bir mesajın bir kuyruk yöneticisinden bir diğerine (uzaktaki olması muhtemel)

37

transferi için her bir mesajın kendi hedef adresini (bir iletim başlığı kullanılarak)

taşıması gerekir.

Bir MQSerisi’nde bir adres iki parçadan oluşur. İlk parça kuyruk yöneticisinin

ismini (yani mesajın teslim edileceği yer) içerir. İkinci parça, bu kuyruk

yöneticisinin altında bulunan (mesajın ekleneceği kuyruk) hedef kuyruğun ismini

içerir.. Hedef adres dışında mesajın izleyeceği yol da belirlenmelidir. Çoğu

durumda yollar (routes) bir kuyruk yöneticisindeki yönlendirme tablosu’nda

(routing table) tutulur. Yönlendirme tablo’sundaki bir girdi çifti şöyledir:

(destQM, sendQM)

Burada, destQM = Hedef kuyruk yöneticisi,

sendQM = Yerel gönderim kuyruğu ’dur.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

Mesaj iletimi (Message transfer) :

Bir yönlendirme tablosu girdisi

(routing table entry) MQSerisi’nde öteki

38

Bir yönlendirme tablosu girdisi

(routing table entry) MQSerisi’nde öteki

isim veya takma isim

(alias) olarak adlandırılır.

Her bir kuyruk yöneticisinin sistem çağında eşsiz (unique) bir isme sahip olması,

etkin olarak bu kuyruk yöneticisi için bir tanımlayıcı kullanılmasıdır. Asıl

problem bu kuyruk yöneticilerinin değiştirilmesinde veya sadece isimlerinin

değiştirilmesinde, O’na mesaj gönderen tüm uygulamaların etkilenmesidir. Bu

problem yerel öteki isimler kullanılarak aşılır. Bir kuyruk yöneticisi içinde

tanımlanan bir öteki isim, diğer kuyruk yöneticisinde başka isimle anılır, fakat

sadece ilk kuyruk yöneticisinin ara yüzlüğünü yapan uygulamalar için elde

edilebilirdir.

Bu yönlendirme ve öteki isim’ler (alias) basit bir uygulamalar için program ara

yüzünü oluşturur. Böyle bir program ara yüzüne Mesaj Kuyruk Ara yüzü

(Message Queue Interface, MQI) denilir. Buna ait ilkeller aşağıda verilmiştir.

(devam)

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

Mesaj iletimi (Message transfer) :

39

Fig. 2.33

ekil 2.33 ‘de, yönlendirme tabloları ve öteki isimleri kullanan bir MQSerisi

kuyruklama a

ğına ait genel organizasyon görülmektedir.

Uzak Nesne Çağrısı (Remote Object Invocation)

Mesaj-yönelimli Kalıcı İletişim (Message-oriented Persistent

Communication) :

Örnek: IBM MQSeries (Example: IBM MQSeries) :

Mesaj iletimi (Message transfer) :

40

Đlkel

(Primitive) Tanım (Description)

MQopen Bir kuyruk aç (uzaktaki muhtemel)

MQclose Bir kuyruk kapat

MQput Açılmış bir kuyruğa bir mesaj koy (put)

Fig. 2.34

MQget Bir kuyruktan (yerel) bir mesaj al (get)

ekil 2.32 ‘de, IBM MQSerisi MQI ile kullanılabilen ilkeller gösterilmektedir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

41

Ders’in sonu…

42

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 6

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 6: Akışa-dayalı iletişim,

Süreçler (Process),

Altsüreçler (Threads).

Akışa-dayalı iletişim (Stream-oriented Communication)

• Bir isteğin, bir prosedür veya metodu (yöntemi) çağırdığı ve cevabın böyle bir

isteğe, uygulamalar arası mesaj alış-verişi ile verildiği mesaj kuyruk

sistemlerindeki iletişim için zamanda hangi noktada bulunulduğu (zamanlama) o

kadar da önemli değildir. Yani sistem çok hızlı da, çok yavaş da çalışsa dahi

bilginin/cevabın doğruluğunu etkilemez. Fakat zamanın önemli olduğu iletişim

3

biçimleri de mevcuttur.

Sürekli ortam için destek (Support for Continuous Media) :

Zamana-bağlı bilgi değiş-tokuşu sürekli medya’nın desteklenmesi olarak formüle

edilir. Hangi bilginin taşındığını bir ortam/lar belirler.

Sürekli (continuous) ortam, farklı veri elemanları arasındaki geçici ilişkilerin

bulunduğu verinin gerçekte ne olduğunun doğru olarak yorumlanmasıdır. Buna

ses dalgası örnek verilebilir.

Kesikli (discrete) ortam, farklı veri elemanları arasındaki geçici ilişkilerin

bulunduğu verinin gerçekte ne olduğunun doğru olarak yorumlanmasının

gerekmediği durumdur. Kısaca, sürekli ortam birebir veri ile ilgilenir. Kesikli

ortam verinin örek parçaları ile (dolaylı olarak asıl veri ile) ilgilenir.

(devam)

Akışa-dayalı iletişim (Stream-oriented Communication)

Veri akışı (Data stream) :

Zamana-bağlı bilgi alış-verişinde dağıtık sistemler genelde veri akışları (data

streams) destekler. Bir veri akışı, veri birimleri dizisidir. Veri akışları ayrık olarak

da, nasıl sürekli medyaya uygulanıyorsa, uygulanabilirler. Örneğin UNIX pipe

‘ları ve TCP/IP bağlantıları ayrık veri akışları (discrete veri streams) için

verilebilir.

4

Zamanlama sürekli veri akışlar için önemlidir. Bu yüzden farklı iletim mod’ları

vardır:

▫ Asenkron iletim mod’unda (asynchronous transmission mode) ; bir

akıştaki veri elemanları, biri diğerinden sonra gelecek şekilde iletilirler, fakat

iletim elemanlarının ne zaman yer alacağı ile ilgili ileriki zaman kısıt’ı

bulunmamaktadır. Bu tipik olarak ayrık veri akışlarıdır. Örneğin bir dosya bir

veri akışı olarak transfer edilir, fakat her bir eleman’ın (item) transferinin ne

zaman tamamlandığından tam olarak ilgisizdir (bağımsızdır).

▫ Senkronize iletim mod’unda (synchronous transmission mode) ; bir

veri akışında her bir birim için tanımlı bir maksimum uçtan uca

(end-toend) gecikme olmalıdır. Bir veri birimi maksimum tolere edilmiş gecikmeden

daha fazla hızlı transfer edilse de bu önemli değildir.

(devam)

Akışa-dayalı iletişim (Stream-oriented Communication)

Veri akışı (Data stream) :

▫ İzokronize iletim mod’unda (isochronous transmission mode) ; veri

birimlerinin zamanında transfer edilmesi gereklidir. Bunun anlamı, veri

transferi bir maksimum ve bir minimum uçtan uca (end-to-end) gecikmeye

uymalıdır. Buna sınırlandırılmış (gecikme) jitter ‘ının oluşumu işaret

eder.

5

Bu izokronize iletim modu, dağıtık çoklu ortam sistemleriyle ilgilidir, sesin ve

video’nun sunumunda rol oynar.

Akışlar

(streams) ; basit

(simple) veya karmaşık

(complex) olabilirler.

Bir basit akış (simple stream) sadece tek bir veri dizisinden oluşur. Bir

karmaşık akış

(complex stream) birkaç ilgili akışın bir araya gelmesi ile

oluşur, bunlara alt akış (substream) ‘ler denilir. Bu alt akış’ların karmaşık akış

içerisinde birbirleriyle zamana bağlı bir ilişkileri vardır. Örneğin stereo ses

akışı iki tane tek ses kanalı akışından oluşur. Bunun en önemli durumu, iki alt

akışın sürekli olarak senkronize edilmesidir.

Akışa-dayalı iletişim (Stream-oriented Communication)

Veri akışı (Data stream) :

Bir akış, kaynaktan hedefe (gösterimin yapıldığı hedefe “sink” denilmiş) sanal

bir kanal ile iletilir. İki farklı yolla bu iş gerçekleşir, birincisi kaynaktan alınan veri

işlenerek bir ağ üzerinden hedefe iletilir orada da işlenerek sink ‘e verilir.

İkincisinde ise kaynaktan alınan veri işlenmeden doğrudan ağ üzerinden sink ’e

verilir.

6

Ayrıca bir kaç alıcıya akış, çoğa yayım (multicast) olarak verilebilir. Fakat bu tarz

dağıtımın ana problemi, alıcıların farklı kalitede akışlar istediği durumların

olmasıdır (different requirements with respect to the quality of the stream).

Fakat her alıcı fazlaca veriyi alıp işleyecek şekilde yeteneklere sahip değildir.

Böylece akış’lar filtre denilen kalite ayarlamaları ile farklı isteklere uyarlanırlar.

Aşağıdaki Şekil 2.35 ‘te, (a) şıkkında, bir ağ üzerinden iki süreç arasında bir akış’ın

kurulması, (b) şıkkında ise iki cihaz arasında doğrudan bir akış kurulması

görülmektedir.

Aşağıdaki Şekil 2.36 ‘da ise, bir kaç alıcı (receiver) için bir akışın çoğa yayım ile

dağıtılması örneği görülmektedir.

(devam)

Akışa-dayalı iletişim (Stream-oriented Communication)

Veri akışı (Data stream) :

7

Fig. 2.35

ekil 2.35 (a)

şıkkında, bir a

ğ üzerinden iki süreç arasında bir akı

ş’ın kurulması

gösterilmektedir.

Akışa-dayalı iletişim (Stream-oriented Communication)

Veri akışı (Data stream) :

8

Fig. 2.35

ekil 2.35 (b)

şıkkında, iki cihaz arasında do

ğrudan bir akı

ş kurulması

gösterilmektedir.

Akışa-dayalı iletişim (Stream-oriented Communication)

Veri akışı (Data stream) :

9

Fig. 2.36

ekil 2.36 ’da, bir kaç alıcı (receiver) için bir akışın ço

ğa yayım ile da

ğıtılması örne

ği

gösterilmektedir.

Akışa-dayalı iletişim (Stream-oriented Communication)

Akışlar ve Servis Kalitesi (Streams and Quality of Service) :

Altta yatan ağın ve dağıtık sistemin isterlerine bağlı ve zamana-bağlı isterlere

Servis Kalitesi

(Quality of Service, QoS) denilir.

Servis Kalitesinin Belirlenmesi (Specifying QoS):

10

QoS isterlerinin karşılanması birkaç yol ile sağlanılır. Bir yaklaşım; akış belirleme

(flow specification) olarak bilinir. Bununla bandgenişliği isterleri, iletim oranları,

gecikmeler vs.. Belirtilebilir. Akış belirleme için Partridge ‘in modelin önerdiği

“Delik kova” (token bucket) algoritması, ağ trafiğine göre akışın nasıl

şekillenebileceğini belirtir. Temel fikir olarak jeton’lar (token) sabit oranla

yaratılırlar. Jeton, ağ üzerinden geçmesine izin verilmiş bir uygulamanın sabit

sayıda byte ’ına karşılık gelir. Aşağıdaki Şekil 2.38 ‘de görülmektedir.

(devam)

Akışa-dayalı iletişim (Stream-oriented Communication)

Servis Kalitesinin Belirlenmesi (Specifying QoS):

11

Fig. 2.38

ekil 2.38 ‘de, delik kova algoritması ’nın temeli gösterilmektedir.

Akışa-dayalı iletişim (Stream-oriented Communication)

Bir Akış’ın Kurulması (Setting up Stream):

Kaynakları sürekli akışlar için ayırmakta kullanılan bir QoS protokolü olarak;

Kaynak Rezervasyon Protokolü

(Resource reSerVation Protocol,

RSVP) bulunur, bu protokol taşınım-seviyesi kontrol protokolüdür ve

yönlendiriciler için kaynak rezervasyonu yapar.

12

Fakat Internet üzerindeki sadece bazı yönlendiricilerde kullanılmış olması ve tüm

yönlendiricilerin bunu desteklemek için tekrar elden geçirilmesi gerektiği için

popüler olarak kullanılmamıştır.

RSVP; veri bağı katmanı ve akış kontrolü kullanır.

Şekil 2.39 ‘da, dağıtık bir sistemde kaynak rezervasyonu için RSVP ‘nin temel

organizasyonu görülmektedir.

Akışa-dayalı iletişim (Stream-oriented Communication)

Bir Akış’ın Kurulması (Setting up Stream):

13

Fig. 2.39

ekil 2.39 ‘da, da

ğıtık bir sistemde kaynak rezervasyonu için RSVP ‘nin temel

organizasyonu görülmektedir.

Süreçler (Processes)

Alt süreçler (Threads):

Bir programı çalıştırmak için, bir işletim sistemi belirli sayıda sanal işlemciler

yaratır, her biri farklı bir programın çalıştırılması içindir. Bu sanal işlemcilerin

takibi için işletim sistemi bir süreç tablosu’na sahiptir. Bu tablodaki girdiler: CPU

kayıtçı değerleri (CPU register values), bellek haritaları, açık dosyalar, hesap

bilgileri vs…

14

Bir süreç, çalışmakta olan bir program olarak tanımlanır, bunun anlamı işletim

sisteminin sanal işlemcilerinden birinde şu an çalıştırılan bir program olmasıdır.

İşletim sistemi süreçlerin birbirlerinin çalışmasına (doğruluğuna) kötü yönde etki

etmesine izin vermez, birbirinden bağımsız süreçler olmasına dikkat eder.

Diğer deyişle, birden çok (multiple) süreç aynı CPU ve diğer donanım kaynaklarını

eşzamanlı olarak paylaşabilir, bu sayede saydamlık sağlanır. Genelde işletim

sistemi böyle bir ayrım için donanım desteğini gerektirir.

Bu eşzamanlılık saydamlığı (concurrency transparency) (Kaynağa bir çok

rakibin eriştiğinin gizlenilmesi), görece yüksek maliyetlidir. Örneğin, bir sürecin

yaratıldığı her bir seferde, işletim sistemi tam bir bağımsız adres uzayını da

yaratmak zorundadır.

(devam)

Süreçler (Processes)

Alt süreçler (Threads): (devam)

Ayrım, (Allocation) bellek segmentlerinin başlatılması (initializing) anlamına

gelebilir, örneğin bir veri segmentinin sıfırlarla doldurulması (zeroing), ilgili

programın bir metin segmenti’ne kopyalanması, geçici veri için bir yığıt’ın

ayarlanması gibi.

15

Bunun yanında her bir süreç için CPU’nun değiştirilmesi (switching) (iki süreç

arasında CPU’nun el değiştirmesi) görece pahalıdır. CPU içeriğini kaydetmekten

daha çok, işletim sistemi bellek yönetim birimi (Memory Management Unit,

MMU) ‘nin kayıtçılarını (registers) değiştirir, çevrim tarafı tamponu

(translation lookaside buffer, TLB) olarak adres çevrim ön belleklerini

geçersiz kılar.

Eğer işletim sistemi, ana bellekte eşanlı (simultaneous) olarak tutabileceğinden

daha fazla süreç destekliyorsa, süreçleri ana bellek ile disk arasında takas eder,

bunu geçerli değişikliği yapmadan önce uygular.

Bir alt süreç veya iş parçacığı (thread), süreç’in mantığına benzer, aynen bir

programın (parçasının) bir sanal işlemcide çalıştırılması mantığı vardır. Bununla

beraber, sürecin aksine, eşzamanlılık saydamlığı’nın yüksek seviyesinin

başarılması eğer bu başarımın düşüşüne neden oluyorsa denenmez.

(devam)

Süreçler (Processes)

Alt süreçler (Threads): (devam)

Öyleyse, bir alt süreç (thread) sistemi genelde sadece birkaç alt süreç tarafından

paylaşılmış olanb bir CPU ‘ya izin verecek minimum bilgiye sahiptir/bakar.

Pratikte, bir alt süreç içeriği (thread context), çoğunlukla CPU içeriğinden

başka hiçbir şeye sahip değildir, bununla beraber alt süreç yönetimi için diğer bazı

bilgiler de bulunur.

16

Örneğin, bir alt süreç sistemi durumu takip eder, bu durum (fact) bir sürecin şu

an bir mutex değişkeninde bloklanması olsun, böylece bu alt süreç çalıştırılmak

üzere seçilmez. Bilgi birden fazla alt süreç yönetimi için kısıtlı olarak gerekli değil

ve göz ardı edilir. Bu nedenle uygunsuz alt süreç erişimleri karşısında verinin

korunması (bir tek süreç içinde) tamamen uygulama geliştiricilere bırakılır. Bu

yaklaşımın iki önemli gerçekleştirimi vardır. İlki bir çoklu alt süreç

(multithread) uygulamanın başarımı, kendi tek alt süreçli karşılığından

(counterpart) daima daha kötüdür.

Bu durum, bir çok durumda çoklu alt süreç işlemi başarım kazancı sağlar.

İkincisi, alt süreçler otomatik olarak birbirlerinden korunmazlar (süreçler

otomatik korumaktadır). Bu yüzden çoklu alt süreç uygulamalar daha fazladan

düşünsel çaba gerektirir. Doğru tasarım ve konuları basit tutmak, çok yardımcı

olur. Şu an bu prensip tam anlaşılmış değildir.

(devam)

Süreçler (Processes)

Dağıtık olmayan sistemlerde alt süreçlerin kullanımı

(Threads Usage in Nondistributed Systems) :

Tek bir alt süreç’li süreç’in en önemli yararı, bir sistem bloklama çağrısı

çağırıldığında, süreç’in tamamının bloklanmasıdır. Örneğin bir spreadsheet

(örneğin: microsoft excel) programında, farklı hücrelerin değerleri devamlı

değiştiriliyorsa ve bunlar arasında bir formülsel bağlantı varsa, tüm ilgili

17

hücrelerde değişikliğe göre otomatik olarak güncellenir.

Eğer sadece bir tane kontrol alt süreci varsa, hesaplama, program girdi için

beklerken devam etmez (cannot proceed). Benzer olarak, bağımlılıklar

hesaplanırken girdi sağlanması kolay değildir. En kolay çözüm en az iki tane

kontrol alt sürecine sahip olmaktır, böylece biri kullanıcı ile etkileşimi idare

ederken diğeri (bu spreadsheet (excel) örneği için) spreadsheet tablosunu

günceller.

Çoklu alt süreç ‘in bir diğer avantajı da paralelleştirme’nin kullanılmasının

mümkün olmasıdır, bunun için bir programın bir çoklu işlemcili sistemde

çalıştırılması gerekir. Bu durumda, paylaşımlı ana bellekte depolanan paylaşımlı

veri varken her bir alt süreç farklı bir CPU ‘ya atanır. Eğer iyi tasarlanmışsa, böyle

paralelleştirme (parallelism) saydam olabilir; süreç oldukça iyi olarak bir tek

işlemcili (uniprocessor) sistemde de çalıştırılabilir, fakat daha yavaş olur.

(devam)

Süreçler (Processes)

Dağıtık olmayan sistemlerde alt süreçlerin kullanımı

(Threads Usage in Nondistributed Systems) : (devam)

Paralelleştirme için çoklu alt süreç, ucuz çoklu işlemcili iş istasyonlarının görece

elde edilebilir olmasından dolayı önem kazanmaktadır. Böyle bilgisayar sistemleri

genellikle istemci-sunucu uygulamalarında çalışan sunucular için kullanılır.

18

Çoklu alt süreç şeklindeki büyük uygulamaların içeriği için kullanışlıdır. Böyle

uygulamalar sıklıkla yardımcı programların (her biri ayrı bir süreç tarafından

çalıştırılan programlardır) bir topluluğundan oluşur. Bu yaklaşım bir UNIX

ortamı için tipiktir.

Programlar arası yardımlaşma (ortak çalışma = cooperation) süreçler arası

iletişim (interprocess communication, IPC) mekanizmasıyla

gerçekleştirilir. UNIX sistemler için bu mekanizmalar tipik olarak

(isimlendirilmiş) pipe’lar, mesaj kuyrukları, paylaşımlı bellek segmentlerini içerir.

Tüm IPC mekanizmaların eksik kalan kısmı, iletişimin sıklıkla içerik değiştirme

(switching = anahtarlama) gerektirmesidir. Aşağıdaki Şekil 3.1 ‘de üç farklı

noktada bu gösterilmektedir.

(devam)

Süreçler (Processes)

Dağıtık olmayan sistemlerde alt süreçlerin kullanımı

(Threads Usage in Nondistributed Systems) : (devam)

19

Fig. 3.1

ekil 3.1 ‘de, IPC ‘nin sonucu olarak içerik değiştirme (context switching)

görülmektedir.

Süreçler (Processes)

Dağıtık olmayan sistemlerde alt süreçlerin kullanımı

(Threads Usage in Nondistributed Systems) : (devam)

IPC ‘nin çekirdek müdahalesi (kernel intervention) gerektirmesi yüzünden, bir

süreç genelde ilk önce kullanıcı modu’ndan (user mode) çekirdek modu’na

(kernel mode) geçmelidir. Bu durum, Şekil 3.1 ‘de S1 ile gösterilmektedir. Bu

durum, Bellek Yönetim Birimi’ndeki (Memory Management Unit, MMU) bellek

20

haritası’nın değişmesini gerektirir, ayrıca çevrim tarafı tamponu’nda

(translation lookaside buffer, TLB) temizlenmesi (flushing) anlamına gelir.

Çekirdek içinde, bir süreç içeriği değişimi oluşur (şekilde S2 ile gösterilmiştir),

daha sonra da diğer parça, çekirdek modu’ndan kullanıcı modu’na tekrar

değiştirilmek suretiyle aktif hale getirilir. İkinci anahtar MMU’nun haritasının

değiştirilmesini ve TLB ‘nin temizlenmesini gerektirir. Süreçleri kullanmak

yerine, bir uygulama ayrı alt süreçler tarafından böyle farklı parçalardan

oluşturulabilir.

Bu parçalar arası iletişim tamamen paylaşımlı veri ile olur. Alt süreç değişimi

(thread switching) tamamen kullanıcı uzayında olur, diğer gerçekleştirimlerde

olsa da, çekirdek, alt süreçlerden haberdar olur ve onları planlar (schedule). Bu

etki başarımı arttırır. Alt süreç topluluğu uygulamayı yapısal olarak kolaylaştırır,

birbirinden bağımsız işlerin gerçekleştirilmesini sağlar.

(devam)

Süreçler (Processes)

Dağıtık olmayan sistemlerde alt süreçlerin kullanımı

(Threads Usage in Nondistributed Systems) : (devam)

Örneğin, kelime işleme için ayrı alt süreçler kullanıcı girdisini idare eder, kelime

ve heceleme kontrolü yapar, indeks oluşturulmasını gerçekleştirir.

Alt süreç Gerçekleştirimi (Thread Implementation) :

21

Alt süreçler genelde alt süreç paketi biçiminde sağlanırlar. Böyle bir paket, mutex

ve koşul değişkenleri (conditional variables) gibi senkronizasyon

değişkenleri ile olabildiğince iyi işlem yapılması olarak alt süreçlerin

oluşturulması ve yok edilmesi için olan işlemleri içerir. Bir paketin gerçekleştirimi

için iki yaklaşım bulunur. İlk yaklaşım bir alt süreç kütüphanesini oluşturmaktır.

Bu kütüphane tamamen kullanıcı mod’unda çalışır. İkinci yaklaşım, çekirdeğin alt

süreçlerden haberdar olması ve onları planlaması’dır (schedule).

Bir kullanıcı-seviyesi alt süreç kütüphanesi bir çok avantaja sahiptir. İlk olarak,

alt sürecin yaratımı ve yok edilmesi için oldukça ucuzdur, çünkü tüm alt süreç

yönetimi (thread administration) kullanıcı adres uzayında tutulur. Bir alt

sürecin oluşturulmasının maliyeti, bir alt süreç yığıt’ı (stack) kurulumu için bellek

ayırma maliyeti tarafından ilk başta belirlenir. Benzer olarak, bir alt sürecin yok

edilmesi yığıt için belleğin serbest bırakılmasını içerir. Bu yığıt daha fazla

kullanılmamaktadır. Her iki işlem de ucuzdur.

(devam)

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

Kullanıcı-seviyesi alt süreçlerin ikinci bir avantajı, alt süreç içerik değiştirme’nin

(switching) sıklıkla sadece birkaç (az sayıda) komutla (instruction)

yapılabilmesidir. Temelde, CPU kayıtçı’larının (register veya yazmaç olarak

bilinmektedir) değerleri sadece depolanması gereklidir ve sonrasında önceden

depolanmış alt süreç değerleri ile tekrar yüklenerek değiştirilmiş olur (switched).

22

Bellek haritalarının değiştirilmesine, TLB ‘nin temizlenmesine ve CPU hesabı’nın

yapılmasına (do CPU accounting) gerek yoktur. Alt süreç içeriği’nin (thread

context) değiştirilmesi/anahtarlanması, iki alt sürecin senkronize edilmesi

gerektiğinde yapılır. Örneğin paylaşımlı verinin bir bölümüne girildiğinde yapılır.

Bununla beraber, bir kullanıcı-seviyesi alt sürecin en büyük eksiği, bir bloklama

sistem çağrısının, alt sürecin ait olduğu sürecin tamamını hemen bloklamasıdır ve

bu süreçteki tüm diğer alt süreçleri de bloklamasına neden olur. Alt süreçler

özellikle büyük uygulamaların oluşturulmasında kullanışlıdır, bu

uygulamalardaki parçalar aynı zamanda mantıksal olarak çalıştırılırlar ve

uygulamayı oluştururlar.

Bu durumda, I/O (girdi/çıktı) sistemi’nin bloklanması diğer parçaların şu anda

çalıştırılmasını engellememelidir.

(devam)

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

Böyle uygulamalar için kullanıcı-seviyesi yardımcı olmaz. Bu problemler

çoğunlukla işletim sistemi’nin çekirdeğinde alt süreçlerin gerçekleştirilmesi ile

atlatılır (circumvented). Ne yazık ki, ödenecek yüksek bir bedel vardır.

Her bir alt süreç işlemi (yaratma, silme, senkronizasyon vs.) çekirdek tarafından

yapılmakta, bu da bir sistem çağrısını gerektirmektedir. Alt süreç içeriklerinin

23

değiştirilmesi/anahtarlanması, süreç içeriklerinin değiştirilmesi/anahtarlanması

kadar pahalı hale gelecektir. Bunun bir sonucu olarak süreçler yerine alt süreç

kullanılmasının bir çok faydası kaybolur.

Kullanıcı-seviyesi’nde (user-level) ve çekirdek-seviyesi’nde (kernel-level) bulunan

alt süreçleri melez biçiminde bulunan bir çözüm genelde hafif-sıklet süreçler

(lightweight processes, LWP) olarak bilinirler. Bir LWP, tek bir sürecin

içeriğinde (ağır içeriğinde (ağır-sıklet süreç, sıklet süreç, heavy-weight process weight process) çalışır. Her bir süreç için ) çalışır. Her bir süreç için

birkaç LWP bulunabilir. LWP ‘lere sahipo olmanın yanı sıra bir sistem kulanıcıseviyesi (user-level) alt süreç paketini de önerir. Önerilen uygulamalar alt süreç

oluşturma ve yok edilmesi için genel işlemlerdir. Ayrıca paket, alt süreç

senkronizasyonu için, mutex ve koşul değişkenleri gibi (onlara benzer) imkanlar

sağlar.

(devam)

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

24

Fig. 3.2

ekil 3.2 ‘de, çekirdek seviyesi hafif-sıklet (lightweight) süreçler ile kullanıcıseviyesi alt süreçlerin kombinasyonu görülmektedir.

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

En önemli konu alt süreç paketinin tamamen kullanıcı uzayında

gerçekleştirilmesidir. Diğer değişle, tüm işlemler çekirdek müdahalesi (kernel

intervention) olmadan alt süreçler üzerinde yapılmaktadır.

Alt süreç paketi, birden çok LWP tarafından paylaşılabilir. Bu yukarıdaki şekilde

görülmektedir. Bunun anlamı her bir LWP, kendi sahip olduğu (kullanıcı- 25

seviyesi) alt süreçleri çalıştırabilir. Sonrasında çoklu alt sürece sahip uygulamalar

alt süreç yaratımları ile oluşturulabilir, sonra da her bir alt süreç bir LWP ‘ye

atanabilir. Bir alt süreç’in bir LWP ‘ye atanması normalen saklı (implicit) ve

programcıdan gizlidir.

Alt süreçlerin (kullanıcı-seviyesi) ve LWP ’lerin kombinasyonu şöyle çalışır. Alt

süreç paketi tek bir rutin’e bir sonraki alt süreç’i planlamak için sahiptir

(schedule). Böyle bir LWP yaratıldığında (bir sistem çağrısı yapılması anlamında

bir yaratım ile), LWP kendi yığıtı’na sahiptir ve çalıştırmak üzere aradığı bir alt

süreç için planlama rutin’ini çalıştırmak üzere komut almıştır (instructed).

Eğer birkaç LWP varsa, o zaman bunların her biri planlayıcıyı (scheduler)

çalıştırır. Alt süreç tablosu, LWP ’ler tarafından paylaşılır ve bu tablo geçerli (şu

anki) alt süreç kümesini takip etmekte kullanılır.

(devam)

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

Bu tablonun iki taraflı (mutually) açık olmayan erişmini garanti etmek için

korunması, mutex’ler ile yapılır, bunlar tamamen kullanıcı uzayında

gerçekleştirilirler. Diğer bir deyişle, LWP ‘ler arası senkronizasyon için herhangi

bir çekirdek (kernel) desteği gerekmez.

Bir LWP çalıştırılabilir bir alt süreç bulduğunda, içeriği (context) bu alt süreç’e

26

aktarır (switches). Bu sırada, dğer LWP ‘ler diğer çalıştırılabilir alt süreçlere

bakabilirler. Eğer bir alt süreç, bir mutex veya koşul değişkenini bloklamak

isterse, gerekli yönetimi ve gerekçeli planlama rutininin çağrısını yapar. Başka bir

alt süreç bulunduğunda (çalıştırılabilir alt süreç), bu alt sürece bir içerik

aktarımı (context switching) yapılır. Tüm bunların güzel tarafı, LWP ‘nin

çalıştırdığı alt süreç’in bilgilendirilmesinin gerekmemesidir. İçerik aktarımı

tamamen kullanıcı uzayında gerçekleştirilir ve LWP için normal program kodu

olarak görünür.

Bir alt süreç, bir bloklama sistem çağrısı yaptığında, çalışma kullanıcı modu’ndan

çekirdek modu’na doğru değişir, fakat hala geçerli LWP ‘nin içeriğinde devam

eder, şu anki LWP ‘nin olduğu noktada daha fazla devam edilmez ve işletim

sistemi içeriği başka bir LWP ‘ye aktarmaya (switch) karar verebilir, bu LWP bir

içerik aktarımının kullanıcı modu’na geri dönülerek gerçekleştirilmesini yapar.

Seçilmiş LWP, daha önce nerede kaldıysa oradan basitçe devam eder.

(devam)

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

Bir kullanıcı-seviyesi alt süreç paketi ile kombinasyonda LWP ‘lerin

kullanılmasının birkaç avantajı bulunur. İlk olarak, yaratım, yok etme ve

senkronize olma alt süreçleri görece ucuz ve tümünde hiçbir çekirdek müdahalesi

içermemektedir.

İkincisi, yeterince LWP ‘ye sahip bir süreç sağlanması, bir bloklama sistem

27

çağrısının tüm bir süreci bekletmemesini sağlar (tüm çalışmanın

engellenmemesi).

Üçüncüsü, LWP ‘ler hakkında bilgiye sahip olmak için bir uygulamaya ihtiyaç

olmamasıdır. Tümü kullanıcı-seviyesi alt süreçler olarak görülür.

Dördüncüsü, LWP’ler kolaylıkla çoklu işleme (multiprocessing)

ortamlarında kullanılabilirler, bu farklı CPU ‘larda farklı LWP ‘lerin çalıştırılması

ile yapılır.

Bu çoklu işleme, tamamen uygulamadan gizlenebilir. Hafif süreçlerin tek eksiği

(kullanıcı-seviyesi alt süreçlerle olan kombinasyondaki tek eksiği) hala LWP ‘lerin

yaratır ve yok ederken pahalı olan çekirdek modu’na ihtiyaç olmasıdır. Bununla

beraber, LWP ‘lerin yaratılması ve yok edilmesi işleminin sadece bazen yapılması

gerekir ve sıklıkla tamamen işletim sistemi kontrolünde olur. LWP ‘lere benzer bir

yaklaşım da planlayıcı aktivasyonları’dır (scheduler activations).

(devam)

Süreçler (Processes)

Alt süreç Gerçekleştirimi (Thread Implementation) : (devam)

LWP ‘ler ile planlayıcı aktivasyonları arasındaki temel fark, bir sistem çağrısı ile

bir alt süreç bloklandığında, çekirdek alt süreç paketine bir üst çağrı yapar, etkin

olarak planlayıcı (scheduler) rutininin bir sonraki çalıştırılabilir alt süreç’i

seçmesi için çağrılmasıdır.

Aynı prosedür bir alt süreç’in bloklaması kaldırıldığında da tekrarlanır. Bu

28

yaklaşımın avantajı çekirdek tarafından LWP’lerin yönetimini kaydetmesidir/

kurtarmasıdır (save). Bununla beraber bu yaklaşım, katmanlı sistem yapısına

(izin verilen sonraki düşük-seviye çağrıları) uymaz.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Alt süreçlerin en önemli özelliği, klasik anlamıyla, onların tüm süreç

bloklanmaksızın bloklama sistem çağrısına izin vermeleridir, bu sırada alt süreç,

süreç içerisinde çalışmaktadır. Bu özellik, alt süreçleri özellikle dağıtık

sistemlerde kullanılmak için çekici kılmaktadır, çünkü daha kolay ekspres (hızlı)

29

bir iletişim biçimini bir çok mantıksal bağlantıyı aynı zamanda yapacak şekilde

sağlar. Buna çoklu alt süreçli istemci ve sunucular ile bakılabilir.

Çoklu alt süreç istemcileri (Multithreaded Clients) :

Dağıtım saydamlığının (distribution transparency) yüksek bir derecesi’nin

sağlanması için dağıtık sistemler geniş alan ağlarında (süreçlerarası uzun mesaj

yayılımı (propagation) zamanlarına ihtiyaç duyar) çalışırlar. Bir geniş alan

ağındaki gidiş-geliş gecikmesi (round-trip delay) yüz milisaniyeler veya bazen

saniyeler civarındadır.

İletişim gecikmelerini gizlemenin en genel yolu, iletişimi başlatmak ve hemen bir

şeyler ile işletmektir. Örneğin Veb gezginleri (Web Browsers) gibi. Bir çok

durumda bir Veb dokümanı resim, ikonlar ve metin içeren bir HTML dosyasından

oluşur. Bir Veb dokümanındaki elemanları ele geçirmek için Internet gezgini bir

TCP/IP bağlantısı oluşturarak gelen veriyi okur ve onu görüntü bileşenine

gönderir.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç istemcileri (Multithreaded Clients) : (devam)

Bir bağlantı kurulması kadar gelen verinin de okunması bloklanmış işlemlerdir.

Uzun soluklu iletişimde her bir işlemin görece bitirilmesinin uzaması bir

dezavantajdır.

30

Bir Veb gezgini sıklıkla HTML sayfayı alır ve sonra bunu gösterir. İletişim

gecikmelerinin saklanılması (gizlenilmesi) için bazı Veb gezginleri mümkün

olduğunca gelen veriyi geldiğinde göstermeye başlar (gelir gelmez). Metin,

kullanıcı için elde edilebilir yapıldığında, aşağı-yukarı kaydırma (scrolling) vb.

Veb gezgini sayfayı oluşturan diğer dosyaları almaya (örneğin resimler) devam

eder. İkincisi alındığında gösterilir. Kullanıcı buna, sayfa tamamlanmadan önce

bütün sayfa bileşenlerinin hepsinin alınmasına kadar beklememek için ihtiyaç

duyar.

Bunun bir etkisi olarak HTML dosyası alındığında, ayrı alt süreçler aktive olarak

ilgili parçaların alımıyla (resim, ses vs.) ilgilenirler. Her bir alt süreç sunucu ile

ayrı bir bağlantı kurarak veri alış-verişi sağlar. Sunucu ile bağlantı ve veri alma,

programlanmış standart (bloklanmış) sistem çağrıları ile olur. Bir bloklama

çağrısı tüm süreci etkilemez.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç istemcileri (Multithreaded Clients) : (devam)

Çoklu alt süreçlere sahip gezginlerin bir diğer önemli faydası birkaç bağlantının

aynı anda eşzamanlı (simultane) olarak kurulabilmesidir. Fakat bağlantılar aynı

sunucuya yapılırsa yük yoğunluğu başarımı etkiler. Bununla beraber, bir çok

31

durumda Veb sunucuları bir çok makine üzerinden kopyaları (replikaları)

alınmıştır, bu kopyalarda Veb doküman setinin tamamen bir kopyası bulunur.

Kopya sunucular (replica servers) aynı sitede bulunur, aynı isim altında bilinirler.

Bir Veb sayfasına bir istek geldiğinde, istek bu sunuculardan birine iletilir. Sıklıkla

bir round-robin stratejisi veya diğer bazı yük dengeleme teknikleri kullanılır.

Çoklu alt süreçlere sahip bir istemci kullanıldığında, bağlantılar farklı kopyalara

kurulabilir, böylece verinin paralel olarak etkin bir biçimde transfer edilmesine

izin verilir.

Bu sayede etkin olarak tüm Veb dokümanı’nın daha az zamanda (replika

yapılmamış bir sunucuya göre daha az zamanda) tamamen gösterilmesi sağlanılır.

Bu yaklaşım, istemcinin gerçekten gelen verinin paralel akışlarını idare edebildiği

durumlarda oluşur. Alt süreçler bu amaç için idealdir.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç sunucuları (Multithreaded Servers) : (devam)

Çoklu alt süreçlere sahip istemcilerin bir çok yararları vardır. Dağıtık sistemlerde

çoklu alt süreçleme’nin (multithreading) ana kullanımı sunucutarafındadır. Pratikler göstermiştir ki, çoklu alt süreçleme göz önüne alınana

32

sunucu’nun kodunu sadece basitleştirmez, fakat ayrıca sunucu

üretimini/geliştirmesini daha kolay hale getirir, böylece tek işlemcili

(uniprocessor) sistemlerden bile daha yüksek performansın parallelliğin

uygulanması ile kazanımını sağlar.

Bununla beraber, çoklu işlemcili (multiprocessor) bilgisayarlar geniş çapta

genel amaçlı iş istasyonları olarak mevcutturlar. Paralellik için çoklu alt süreçleme

daha kullanışlıdır. Sunucu kodu yazımında alt süreçlerin yararlarını anlamak için,

bir dosya sunucusunun organizasyonunu göz önüne alalım, burada sunucu diski

beklemek için bloklanılır. Dosya sunucusu normalen gelen bir dosya işlemi

isteğini bekler, sonra da isteği işler ve geri cevabı gönderir. En popüler

organizasyon aşağıdaki şekilde gösterilmektedir. Burada bir alt süreç dağıtıcı

(dispatcher) olarak görev yapar ve bir dosya işlemi için gelen isteği okur.

Dispatch kelimesi, dağıtıcı veya sevk edici anlamına gelmektedir.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç sunucuları (Multithreaded Servers) : (devam)

33

Fig. 3.3

ekil 3.3 ‘te, bir dağıtıcı/işçi modeli ile organize olmuş bir çoklu alt süreç sunucusu.

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç sunucuları (Multithreaded Servers) : (devam)

İstekler, istemciler tarafından bu sunucunun çok iyi bilinen uç noktası (end point

veya port) için gönderilir. İsteğin çalıştırılmasından sonra sunucu bir boşta

bekleyen (idle olan) (yani bloklanan) bir işçi alt süreci seçer ve isteği ona iletir

34

(hands it the request).

İşçi bir bloklanmış okumayı yerel dosya sisteminde gerçekleştirerek işlemektedir,

bu alt sürecin veri diskten alınana kadar bekletilmesine neden olabilir. Eğer alt

süreç bekliyorsa, başka bir alt süreç çalıştırılmak üzere seçilir. Örneğin dağıtıcı

(dispatcher) daha fazla iş ele geçirmek için seçilebilir. Alternatif olarak başka bir

işçi alt süreç, şu an çalışmaya hazırsa, seçilebilir.

Dosya sunucusu alt süreçlerin yokluğunda nasıl yazılabilir durumdadır ona

bakacak olursak; olası bir şekilde, tek bir alt süreç olarak çalışabilir. Dosya

sunucusunun ana döngüsü bir istek alı, onu çalıştırır, bir sonrakini almadan önce

bunun çalıştırılmasının bitmesini sağlar. Disk için beklerken, sunucu

beklemededir ve diğer herhangi bir isteği işlemez. Sonuçta, diğer istemcilerden

gelen istekler idare edilmezler.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç sunucuları (Multithreaded Servers) : (devam)

Ayrıca, eğer dosya sunucusu adanmış bir makinede (dedicated machine)

çalıştırılıyorsa, genel olarak bu durumda, CPU basitçe, dosya sunucusu disk için

beklerken boşta beklemektedir (idle). Net sonuç çok az istek/saniye (saniye

35

başına istek) ‘in işlenmesidir. Böyle alt süreçler kayda değer başarım kazanımı

sağlar, fakat her bir alt süreç seri olarak programlanmıştır (normal yoldan).

İki mümkün tasarım bulunmaktadır. Bir çoklu alt süreç dosya sunucusu ve bir

tek-alt süreçli dosya sunucusu. Alt süreçlerin elde edilebilir olmadığını, fakat

sistem tasarımcıları tek bir alt süreçli işlemin neden olduğu başarım kaybının

kabul edilemez olduğunu kabullendiğini varsayalım. Üçünü bir olasılık ise büyük

bir sonlu-durum makinesi (finite-state machine) olarak sunucunun çalışmasıdır.

Bir istek geldiğinde sadece bir tek alt süreç çalıştırılır. Eğer önbellek yeterli gelirse

sorun yoktur, fakat yeterli gelmezse bir mesaj diske gönderilmek zorundadır.

Bununla beraber, bloklamanın yerine, geçerli isteğin durumunu bir tabloya

kaydeder ve daha sonra devam ederek bir sonraki mesajı alır. Bir sonraki mesaj,

yeni bir iş için bir istek olabileceği gibi önceki bir işlem hakkında dsikten gelen bir

cevapta olabilir. Eğer yeni bir iş ise bu iş başlatılır.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç sunucuları (Multithreaded Servers) : (devam)

Eğer bir diskten gelen cevap ise, ilgili bilgi tablodan alınır ve cevap işlenir, sonrasında

istemciye gönderilir. Bu şemada, sunucu alım ve gönderim için bloklanmış çağrıların

kullanılmasına sahiptir.

36

Bu tasarımda, “sıralı süreç (seri süreç)” modelini iki durumda kaybederiz. Hesaplama

durumu açıkça (explicitly) kaydedilmeli ve her gönderilip alınan mesaj için tabloda

yenilenmelidir (restore). Burada alt süreçlerin ve onların yığıtlarının simulasyonu zor

yoldan yapılır. Süreç, bir olay (event)’ın alınması ve sonra ona tepki verilmesi (burada tepki

o olay’da ne olduğuna bağlıdır) şeklindeki sonlu-durum (finite-state machine) olarak

çalışmaktadır.

Alt süreçlerin önerdiği şey daha açık hale gelmiştir. Seri süreç fikrini sürdürmek bloklanmış

sistem çağrıları (disk ile konuşan bir RPC gibi) yaparak ve hala paraleleştirmeyi başararak

yapmak mümkündür. Bloklanmış sistem çağrıları programlamayı daha kolay hale getirir ve

paralelleştirme başarımı (performance) artırır.

Tek alt süreçli sunucu bloklanmış sistem çağrılarının kolaylığını sürdürür, fakat

performanstan vazgeçer. Sonlu durum makinesi yaklaşımı paraleleştirme sayesinde yüksek

başarım elde eder fakat bloklanmamış çağrıları kullandığından programlaması zordur.

(devam)

Süreçler (Processes)

Dağıtık Sistemlerde Alt Süreçler (Threads in Distributed Systems) :

(devam)

Çoklu alt süreç sunucuları (Multithreaded Servers) : (devam)

37

Fig. 3.4.

Model Karakteristik

Alt süreçler (Threads)

Paralelleştirme var, Bloklanmış sistem

çağrısı (Parallelism, blocking system

calls)

Tek bir alt süreçli süreç

(Single-threaded process)

Paralelleştirme yok, Bloklanmış sistem

çağrıları (No parallelism, blocking

system calls) (Single-threaded process) system calls)

Sonlu-durum makinesi

(Finite-state machine)

Paralelleştirme var, bloklanmamış

sistem çağrıları (Parallelism,

nonblocking system calls)

ekil 3.4’te, bir sunucuyu oluşturmak için var olan üç yol verilmektedir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

38

Ders’in sonu…

39

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 7

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 7: İstemciler (Clients),

Sunucular (Servers).

Đstemciler (Clients)

• Bir çok istemcinin ana vazifesi, bir insan kullanıcıyla ve uzaktaki bir sunucuyla

etkileşmektir. Bir çok istemci için kullanıcı ara yüzünü desteklemek ana

özelliktir. Bir çok durumda bir kullanıcı ile bir uzaktaki sunucu arasında görece

basit ve istemci donanımı ile tümleşik bir ara yüz bulunmaktadır. Örneğin cep

telefonları geleneksel tuş takımının ekranla kombine edilmesine sahiptir.

3

Daha önemli bir sınıf ise grafik kullanıcı ara yüzüdür. En basit örnek X window

sistemidir.

X window sistemi:

X window sistemi genelde X olarak anılır, bit-haritalı terminalleri kontrol

etmede kullanılır, bunlar bir monitor, klavye, mouse gibi çevre bileşenleridir. X,

işletim sisteminin bir parçası olarak görülür, böylece bu bileşenleri kontrol

eder. Sistemin kalbini X kernel oluşturur. Bu, terminale özgül (terminalspecific) aygıt sürücülerini içerir ve genelde donanıma oldukça bağımlıdır.

X kernel görece ekranı kontrol etmek için düşük seviyeli ara yüze sahiptir. Bu

arayüz Xlib adı verilen bir kütüphane olarak uygulamalar için elde edilebilir

durumdadır.

(devam)

Đstemciler (Clients)

X window sistemi:

Fig. 3.5

4

Şekil 3.5 ‘te, X window sisteminin temel organizasyonu görülmektedir.

Đstemciler (Clients)

• Bunun genel organizasyonu aşağıdaki şekilde verilmektedir

(X terminolojisinde, X kernel, X sunucu anlamındadır, bunun özelliklerini

kullanan programlar ise X istemcisi olarak adlandırılır, kafa karışıklığına yol

açmamak için X sunucu, X istemci denilmiyor).

5

• X window iki çeşit uygulama programının ayrımını yapar: normal uygulamalar

ve pencere yöneticileri. Normal uygulamalar genelde (Xlib üzerinden) ekranda

pencere yaratılmasını ister, daha sonra girdi ve çıktı işlemlerini yaparlar.

Ayrıca, X bir uygulamanın penceresinin aktif olduğuna emin olmaktadır, bunun

anlamı, mouse pencere içine geldiğinde klavye ve mouse kontrolü uygulamaya

geçmektedir.

• Bir pencere yöneticisi (window manager) tüm ekranı yönetmek üzere izin

verilmiş (özel izin) bir uygulamadır. Normal uygulamalar, ekran yönetimi

sınırlamalarına (pencere yöneticisi tarafından gerçekleştirilen) uymalıdır.

Örneğin, pencerelerin asla üst üste örtüşmeyeceğine veya pencerelerin daima

aynı renkte gözükeceğine pencere yöneticisi karar verebilir. Sonuçta sistemin

görünümüne pencere yöneticisi karar verir.

(devam)

Đstemciler (Clients)

• X sistemi özellikle, X kernel ile olan olaylar ve Xlib ‘in değiş-tokuş edebileceği

verinin bir kopyasıyla ilgili ağ-tabanlı iletişim protokolü olan X protokolünü

sağlar. X kernel ’a göre konfigüre edilmiş terminaller X terminal olarak

anılırlar.

6

Bileşik Belge (Compound Documents)

Bileşik belge; metin, ses, grafik, görüntü gibi farklı uygulamalardan gelen

verileri içeren ve bu verilerin gerek belge içinde yer aldığı, gerekse bir bağ (link)

ile bağlı olduklarında otomatik olarak güncellendiği belge cinsine

denilmektedir.

Modern kullanıcı ara yüzleri daha çok seçeneğe sahiptir. Örneğin sürükle-bırak

(drag-and-drop) işlemleri yerinde-düzenleme (in-place editing) gibi. Sürüklebırak ile bir dosya kolayca pencereler (klasörler) arasında taşınabilir. Yerindedüzenleme ile ise o an Word, Excel, Powerpoint gibi “Ne görürsen O’nu

bulursun” (What you see what you get, WYSWYG) tarzı programların dosya

türlerinde) ilgili bölümün üzerinde ilgili bölümle yapılabilecek değişiklikler akla

gelir.

(devam)

Đstemciler (Clients)

Bileşik Belge (Compound Documents)

Bir Bileşik belge notasyonu kullanıcı ara yüzü arkasındaki anahtar fikirdir.

Bunun anlamı, kullanıcı ara yüzü seviyesinde birbirinden farklı dosya türlerinin

görünmez/hissedilmez olarak bütünleştirildiği dokümanlar topluluğu

olmasıdır.

7

Bir kullanıcı ara yüzü bileşik dokümanları (belgeleri), farklı uygulamaların

dokümanın farklı parçaları üzerinde işlem yapması olayını gizleyerek, idare

eder.

Kullanıcı için, tüm parçalar görünmez/hissedilmez bir yoldan tümleştirilmiştir.

Bir parça değiştirildiğinde, diğerlerine de etki eder ve kullanıcı ara yüzü uygun

ölçümleri yapabilir, örneğin ilgili uygulamaları haberdar edebilir.

Bir bileşik doküman ile ilişkili uygulamalar istemici ’nin makinesinde (X server

yani X kernel ’a uzaktan erişim için bahsedilmiş) çalıştırılamaz. Bununla

beraber kullanıcı ara yüzleri, bileşik dokümanları desteklerler.

(devam)

Đstemciler (Clients)

Dağıtım saydamlığı için istemci-taraflı yazılım (Client-Side Software

for Distribution Transparency)

Bir çok durumda işlem parçaları ve bir istemci-sunucu uygulamasında veri

seviyesi istemci-tarafında oldukça iyi çalıştırılır. Gömülü istemci yazılımı özel

bir sınıf biçimindedir. Buna örnek olarak ATM (Otomatik para çekme makinesi,

8

automatic teller machine), barkod okuyucular, TV set-üstü kutuları verilebilir.

Bunlarda (çoğu durumda) kullanıcı ara yüzü istemci yazılımın görece küçük bir

parçasıdır, aksine yerel işleme ve iletişim olanakları daha büyük parçalarıdır.

Kullanıcı ara yüzü ve diğer uygulamayla ilişkili yazılımların yanı sıra, istemci

yazılımı dağıtım saydamlığını başarmak için bileşenler bulundurur. İdeal

olarak, bir istemci uzaktaki süreçler ile iletişimden haberdar olmamalıdır.

Aksine, doğruluk ve başarım nedenlerinden dağıtım sunucular için daha az

saydamdır. Örneğin kopya (replika) sunucular bazen iletişim için belirli bir

sırayı izlemek zorunda kalırlar.

(devam)

Đstemciler (Clients)

Dağıtım saydamlığı için istemci-taraflı yazılım (Client-Side Software

for Distribution Transparency)

Erişim saydamlığı (access transparency), genelde istemci koçanı’nın (client

stub) bir ara yüz tanımından yaratılması sayesinde idare edilir. Bu tanımı

sunucu önerir. Koçan (Stub) sunucuda bulunan ara yüzün aynısını sağlar, fakat

9

olası makine mimarisi farklılıklarını gizler (geçerli iletişim’de olabildiğince iyi

olarak).

Yerleşimi, göç etmeyi ve tekrar yerleşim (location, migration, relocation)

saydamlığını idare etmenin bir çok farklı yolu vardır. Bir geleneksel

isimlendirme sisteminin kullanımı önemlidir; bir çok durumda istemci-sunucu

yazılımla yardımlaşma önemlidir (cooperation).

Örneğin, bir sunucu hali hazırda bir sunucuya bağlanmışsa istemcinin ara

katmana yazılımı (middleware) kullanıcıdan sunucunun şu anki yerini gizler,

saydam olarak sunucuyu gerekiyorsa tekrar bağlar. En kötü ihtimalle, istemci

uygulama geçici bir başarım kaybını belirtebilir.

(devam)

Đstemciler (Clients)

Dağıtım saydamlığı için istemci-taraflı yazılım (Client-Side Software

for Distribution Transparency)

Benzer yoldan, bir çok dağıtık sistem kopyalama (replikasyon) saydamlığını,

istemci-tarafındaki çözümler anlamıyla gerçekleştirir. Örneğin, uzaktaki

nesneleri mevcut olan bir dağıtık sistemi düşünelim. Uzaktaki nesnenin

10

kopyalanması (replikasyonu), her bir kopyaya (replika’ya) bir çağrı (invocation)

isteğinin iletilmesi ile başarılır. Bu olay Şekil 3.6 ‘da gösterilmektedir.

İstemci’nin vekil’i (proxy) tüm cevapları (response) saydam olarak toplar ve tek

bir yeri dönüş değerini istemci uygulamaya döndürür.

Son olarak, hata (failure) saydamlığı vardır. Sunucu ile oluşan iletişim

hatalarının gizlenmesi tipik olarak istemci ara katman yazılımı sayesinde

yapılır. Örneğin istemci ara katman yazılımı bir sunucuya bir çok kez

bağlanmayı denemeyi tekrarlayacak şekilde ayarlama (konfigüre) yapılabilir,

veya başka bir sunucuyu deneyebilir. Her iki durumda istemci ara katman

yazılımı önceki bir oturumda önbelleklenmiş bir veriyi geri döndürür, bazen de

Veb gezgini (Web browser)’ ları tarafından bir sunucuya bağlanma hatası

olabilir.

(devam)

Đstemciler (Clients)

Dağıtım saydamlığı için istemci-taraflı yazılım (Client-Side Software

for Distribution Transparency)

Fig. 3.6.

11

Şekil 3.6 ‘da, bir istemci-tarafında çözüm kullanan bir uzak nesne’nin saydam

biçimdeki kopyalanması için bir yaklaşım gösterilmektedir.

Đstemciler (Clients)

Dağıtım saydamlığı için istemci-taraflı yazılım (Client-Side Software

for Distribution Transparency)

Eşzamanlılık (concurrency) saydamlığı, özel ortam (intermediate) sunucuları

sayesinde idare edilir, işlem (transaction) monitörleri bulunur ve istemci

yazılımın desteği daha az gerekir. Kalıcılık (persistence) saydamlığı tamamen

12

sunucuda idare edilir.

Kalıcılık veya süreklilik anlamına gelen “persistence” saydamlığı, veri’nin diskte

mi bellekte mi olduğunun kullanıcıdan gizlenmesidir.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Bir sunucu, istemciler topluluğunun adına özel bir servisi gerçekleştiren bir

süreçtir. Her sunucu aynı yoldan organize olur: istemciden gelen bir isteğin

beklenilmesi ve sonrasında isteğin dikkate alınarak bir sonraki isteğe kadar

13

beklenilmesi.

Sunucuları organize etmek için bir çok yol mevcuttur. İteratif sunucu

durumunda sunucu isteği kendisi idare eder ve gerekirse, istekte bulunan

istemciye bir cevap geri döndürür. Bir eşzamanlı (concurrent) sunucu isteği

kendi idare etmez, fakat onu ayrı bir alt süreç veya sürece gönderir, daha sonra

hemen bir sonraki gelecek istek için beklemeye başlar.

Bir çoklu alt süreç’li sunucu, eşzamanlı sunucuya örnektir. Bir eşzamanlı

sunucu gerçekleştirimine alternatif her bir gelen istek için yeni bir süreç

başlatmalıdır. Bu yaklaşım bir çok UNIX sistemde uygulanır. Alt süreç veya

süreç isteği, isteği yapan istemciye bir cevap döndürmek için sorumludurlar.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Başka bir konu ise, istemcilerin bir sunucuya bağlandığındaki durumdur. Tüm

durumlarda, istemciler bir uç noktaya (endpoint) (buna port da denilebilir)

istek gönderir. Sunucu’nun çalıştığı makinede bu port bulunur. Her bir sunucu,

14

özel bir uç noktayı (port) dinler. Bir istemci’nin bir servisinin uç noktasını

(portunu) bilmesini açıklayan bir çok yaklaşım vardır. Bir yaklaşım, iyi bilinen

servisler için global olarak uç noktaların atanmasıdır.

Örneğin, FTP istekleri için TCP port 21 devamlı dinlenir, benzer olarak HTTP

sunucusu WWW için daima TCP port 80 ‘ini dinler. Bu uç noktaların ataması

Internet Atanmış Numaralar Otoritesi

(Internet Assigned Numbers

Authority, IANA) tarafından yapılmıştır. Atanmış uç noktalarla, istemci

sunucunun çalıştığı makinenin ağ adresini sadece bulmak ister. İsimlendirme

servisleri bu amaçla kullanılır.

Önceden atanmış bir uç noktaya ihtiyacı olmayan bir çok servis bulunmaktadır.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Örneğin, gün zamanı sunucusu bir uç noktayı dinamik olarak kendi yerel

işletim sistemine atayarak kullanabilir. Bu durumda, bir istemci ilk önce uç

noktaya bakar. Bir çözüm, DCE’de gördüğümüz gibi, özel bir arkaplan

15

süreci’nin (daemon) her bir makinede sunucuları çalıştırmak için

çalıştırıldığıdır.

Arkaplan süreci, sunucuda yerleşmiş her bir servisin geçerli uç noktasını takip

eder. Arkaplan sürecinin kendisi iyi bilinen bir uç noktayı dinler. Bir istemci, ilk

önce arkaplan süreci ile iletişime geçer, uç noktayı ister, daha sonra özel sunucu

ile iletişime geçer.

Genelde özel bir servisle bir uç nokta ilişkilendirilir. Bununla beraber her bir

servis ayrı bir sunucu anlamında gerçekte uygulanarak kaynak israfına neden

olabilir. Örneğin, tipik bir UNIX sistemde, bir çok sunucu eşanlı (simultane)

olarak çalışmaktadır ve bir çoğu pasif olarak bir istemcinin isteği gelene kadar

beklemektedir.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları: Fig. 3.7

16

Şekil 3.7 ’de görülenler:

a) İstemci-sunucu (client-to-server) bağlannmasında DCE’deki bir arkaplan (daemon)

sürecinin kullanımı.

b) İstemci-sunucu (client-to-server) bağlanmasında UNIX’deki bir superserver kullanımı.

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Bir çok pasif süreci beklemek yerine daha etkin olarak tek bir superserver isimli

sunucu, özel bir servis ile ilişkili her bir uç noktayı dinler. Bu yaklaşıma örnek

olarak UNIX sistemlerde “inetd” arkaplan süreci bulunmaktadır. Inetd çok iyi

17

bilinen port’ları Internet servisleri için dinler. Bir istek geldiğinde, isteğin

sonraki durumlarını dikkate alan bir süreç oluşturulur. Bu süreç işi bittikten

sonra sonlanır.

Bir sunucu tasarımı yapılırken bakılan başka bir konu da, bir sunucunun nasıl

kesintiye uğratılacağıdır. Örneğin, bir FTP sunucusuna çok büyük bir dosya

yüklemeye karar vermiş bir kullanıcıyı göz önüne alalım. Aniden bu dosyanın

yanlış bir dosya olduğunu fark etsin, böylece daha sonraki (ilerleyen) veri

iletimini sunucuya doğru kesmek (interrupt) istesin. Bunu yapmak için birkaç

yol mevcuttur. Bu yaklaşım şu anki Internet’te sadece çalışmaktadır (bazen tek

alternatiftir). İstemci uygulamadan kullanıcı aniden çıkarsa (sunucuya olan

bağlantı otomatik olarak kesilir) hemen tekrar bağlanılarak sanki hiçbir şey

olmamış gibi davranılır. Sunucu eski bağlantıyı koparır ve istemcinin

çöktüğünü (crash) düşünür.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Daha iyi bir yaklaşım ise, iletişim kesintilerini idare etmektir. İstemci ve sunucu

geliştirmek için bant dışı

(out-of-band) veri göndermek olasıdır, bu veri

istemcinin herhangi diğer bir verisinden önce sunucu tarafından işlenen

18

veridir.

Buna bir çözüm olarak, istemcinin gönderdiği bant dışı (out-of-band) verisi için

ayrı bir kontrol uç noktasını sunucu dinler, aynı zamanda (daha düşük bir

öncelikle) normal veri geçişlerini de uç nokta üzerinden dinler. Başka bir çözüm

olarak da, bant dışı veriyi aynı bağlantı üzerinden yani istemcinin isteğini

gönderdiği bağlantı üzerinden göndermektir. TCP ‘de, örneğin, acil veri

göndermek mümkündür.

Acil veri sunucuda alındığında, ikincisi kesintiye uğrar (interrupted) (UNIX

sistemlerde bir sinyal sayesinde), bu durum veri’nin gözlenerek ona göre idare

edilmesinden sonra olur.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

En son ve önemli tasarım konusu, sunucunun stateless

(durumsuz) olup

olmadığıdır. Bir stateless (durumsuz) sunucu, istemcilerin durumu hakkında

bilgi tutmaz ve herhangi bir istemcisi hakkında bilgi sahibi olmadan kendi

19

durumunu değiştirir. Örneğin bir Veb sunucu, stateless olmaktadır.

Bir istek geldiğinde, Veb sunucusu istemcisini tamamen unutur. Bir Veb sunucu

dosyalar topluluğunu (bir dosya sunucusu ile yardımlaşması olasıdır) yönetir,

benzer olarak burada da istemcileri bilgilendirmeden dosyalarla ilgili değişiklik

yapabilir.

Aksine, bir stateful

(durumlu) sunucu, istemcileri hakkında bilgi sahibi olur.

Tipik örnek olarak, bir dosya sunucusu bir dosyanın yerel bir kopyasını bir

istemcinin bulundurmasına izin verir, böylece güncelleme (update) işlemlerinin

gerçekleştirilmesi sağlanılır. Böyle bir sunucu (istemci, dosya) girdilerine sahip

bir tablonun bakımını yapar. Böyle bir tablo, hangi dosya için güncelleme iznine

sahip olduğunu (şu an için) sunucunun takip etmesine olanak tanır. Buradaki

dosyadan kastımız, bu dosyanın en güncel halidir.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Bu yaklaşım okuma-yazma işlemleri için başarım (performance) artışı sağlar.

Stateless sunucular üzerinden başarım artışı, stateful tasarımların önemli bir

faydasıdır. Fakat stateful sunucuların büyük bir eksiği de bulunmaktadır. Eğer

20

sunucu çökerse, girdi tablosunu (yani (istemci, dosya)) kurtarır, diğer hallerde

bir dosya üzerindeki en son güncellemeleri işlemeyi garanti etmez.

Genelde, böyle bir stateful sunucu, çökmeden önceki tüm durumunu

kurtarmak (yani tablosunu) ister. Kurtarım oldukça karmaşıktır. Fakat stateless

tasarımda özel bir ölçü tüm çökmüş sunucu kurtarımları için gerekmemektedir.

Bir sunucu tasarlanırken, sunucu tarafından sağlanan servislere, tasarımın

stateless veya stateful olması etki etmez. Örneğin dosya daha önce okuma veya

yazma için açıldıysa stateless sunucu bir yönden veya başka yönden davranışını

korur. Genel bir çözüm, sunucu ilgili dosyayı okuma veya yazma isteğine cevap

olarak açar ve sonra geçerli okuma-yazma işlemi yapılır ve iş bitince hemen

dosya kapatılır.

(devam)

Sunucular (Servers)

Sunucular (Servers) için

Genel tasarım konuları:

Diğer durumlarda, bir sunucu bir kayıt’ı istemci davranışı hakkında tutmak

isteyebilir, bu daha etkin olarak isteğe cevap olabilir. Örneğin Veb sunucuları

bazen istemciye favori sayfalarını (istemcinin favorisi) doğrudan hemen verme

21

olasılığını önerir.

Bu yaklaşım, sunucu bu istemci için bir tarih (history) geçmiş bilgisi tutuyorsa

mümkündür. Önceki erişimler için bilgi cookie’lerde tutulur. Bu bilgi istemcide

tutulan saydam bir bilgidir. Cookie’ler (çerezler) browser tarafından

çalıştırılmaz, sadece depolanır.

Sunucular (Servers)

Nesne sunucuları (Object servers):

Bir nesne sunucusu (object server), dağıtık nesneleri desteklemek üzere

biçimlendirilmiştir. Genel bir nesne sunucusu ile diğer (daha geleneksel)

sunucular arasındaki en önemli fark, nesne sunucusunun özel bir servisi

gerçekten kendisinin sağlayamamasıdır. Nesneler tarafından gerçekleştirilen

22

özel servis sunucuda kalır. Temelde, sunucu sadece yerel nesnelerin çağrılması

(invoke) anlamında, uzaktaki istemcilerin isteklerini temel alan çağrıları sağlar.

Bu görece basittir; nesne ekleme ve çıkarma ile servis değiştirilmesi basittir.

Bir nesne sunucusu nesnelerin yaşadığı bir yer gibi davranır. Bir nesne iki

parçadan oluşur: veri sunumu ((state) yani durum) ve kod biçimi (yöntemlerin

gerçekleştirimi).

Bu parçaların ayrılması veya yöntem (method) gerçekleştirimi çoklu nesneler

tarafından paylaşılır, bu nesne sunucusuna bağlıdır. Kendi nesnelerini çağıran

bir nesne sunucusunun izlediği yolda farklılıklar bulunur. Örneğin çoklu alt

süreçli bir sunucuda, her bir nesne ayrı bir alt süreç’e atanmış olabilir veya ayrı

bir alt süreç her bir invokasyon (çağrım) isteği için kullanılabilir.

(devam)

Sunucular (Servers)

Nesne çağırım alternatifleri (Alternatives for invoking objects):

Bir nesne için çağrılma (invoke), nesne sunucusunun hangi kodu çalıştıracağını

bilmesini gerektirir (hangi veri üzerinde çalışacağını kodun çalıştırılacağını)

invokasyon’a dikkat edecek ayrı bir alt süreç başlatmasa da bu bilgiyi bilmesi

gerekir.

23

Basit bir yaklaşım olarak, tüm nesnelerin aynı göründüğünü ve bir nesneyi

çağırmak (invoke) için tek bir yol olduğunu varsayalım. Temelde bunu DCE

yapar. Ne yazık ki, böyle bir yaklaşım genellikle esnek değildir ve sıklıkla dağıtık

nesnelerin üreticilerini sınırlandırır. Daha iyi bir yaklaşım bir sunucunun farklı

kuralları (policies) desteklediği durumdur.

Örneğin geçici (transient) nesneler gibi. Geçici bir nesne, sadece kendi

sunucusu var olduğu sürece var olan bir nesnedir,fakat kısa bir süre için olması

olasıdır. Bir dosyanın bellekte yer alan salt okunur kopyası tipik olarak bir

geçici nesne olarak gerçekleştirilmiş olabilir.

(devam)

Sunucular (Servers)

Nesne çağırım alternatifleri (Alternatives for invoking objects):

Benzer olarak, bir hesap makinesi (yüksek-başarımlı bir sunucuda çalışması

olasıdır) bir geçici nesne olarak gerçekleştirilmiş olabilir. Kabul edilebilir bir

kural (policy), ilk invokasyon isteği geldiğinde geçici bir nesnenin yaratılması ve

artık hiç bir istemci ona bağlanmayacağında ise onu yok etmek anlamına gelir.

24

Bu yaklaşımın avantajı, geçici nesnenin sunucunun kaynaklarını sadece nesne

için gerçekten gerektiği sürece istemesidir. Bir invokasyon’un eksiği,

tamamlanmak için vakit almasıdır, çünkü nesnenin önce yaratılması gerekir.

Öyleyse, alternatif bir kural (policy) bazen tüm geçici nesnleri, sunucu

başlatıldığında (initialized) yaratmaktır, bu sayede hiçbir istemci nesneyi

kullanmadığı sıradaki tüketilecek kaynak maliyetinde bu yaratım yapılmış olur

(yani en az/boşta kullanım varken).

Benzer bir yol ise, bir sunucunun her bir nesnenin kendi bellek segmenti’nde

yerleştiği bir kuralı takip etmesidir. Diğer bir deyişle, nesneler ne kodu ne de

veriyi paylaşmazlar. Böyle bir kural, bir nesne gerçekleştirimi ayrı bir kod ve

veri ayrıştırmı yapmadığında veya güvenlik nedenleriyle nesnelerin ayrılması

gerektiğinde, gerekli olabilir.

(devam)

Sunucular (Servers)

Nesne çağırım alternatifleri (Alternatives for invoking objects):

İkinci durumda, sunucu özel ölçütlerin sağlanmasını isteyebilir veya altta yatan

işletim sisteminin desteğini isteyebilir, böylece segment sınırlarının

aşılmadığına emin olunur. Alternatif bir yaklaşım, en azından kodlarını

paylaşmaları için nesnelere izin vermektir.

25

Örneğin, bir veritabanı aynı sınıfa ait nesneleri içeriyorsa etkin olarak sunucuya

sadece bir kez sınıf gerçekleştiriminin yüklenilmesiyle gerçekleştirilebilir. Bir

istek bir nesne invokasyon’u için geldiğinde, sunucu sadece nesne’nin

durumunu veritabanından ele geçirmek ister ve istenilen yöntemi çalıştırır.

Benzer olarak, alt süreçleme için bir çok farklı kural (policies) vardır. En basit

yaklaşım sunucu ile sadece tek bir kontrol alt süreci’nin gerçekleştirimidir.

Alternatif olarak, sunucu birkaç alt süreç’e sahip olabilir, kendi nesnelerinin her

biri için bir tane olacak şekilde alt süreçlere sahip olabilir. Bir invokasyon isteği

bir nesne için geldiğinde sunucu isteği bu nesneden sorumlu olan alt süreçe

geçirir/gönderir. Eğer alt süreç şu an meşgulse, istek geçici olarak kuyruğa

alınır (queued). Bu yaklaşımın avantajı, nesnelerin otomatik olarak eş zamanlı

erişimden korunması, yani invokasyonlar nesne ile ilişkilendirilmiş tek bir alt

süreç sayesinde serileştirilmiş olur.

(devam)

Sunucular (Servers)

Nesne çağırım alternatifleri (Alternatives for invoking objects):

Doğaldır ki, her bir invoksayon isteği için ayrı bir alt süreç kullanmakta olasıdır,

eş zamanlı erişimden hala korunan nesneler istenmektedir. Nesne başına alt

süreç veya yöntem başına alt süreç kullanımının bağımsızlığı, alt süreç’lerin

istek halinde yaratılması veya sunucunun bir alt süreç havuzu oluşturması

26

seçilmese dahi vardır. Tek bir en iyi kural yoktur.

Nesne bağdaştırıcı (Object adapter):

Aktivasyon kuralları (activation policies) olarak tanımlanan genelde bir

nesnenin nasıl çağrılacağının kararıdır. Bunun üzerinde durmak bir çok

durumda nesnenin kendisi ilk önce sunucunun adres uzayından (yani aktive

edilmiş) gerçekte çağrılmadan (invoked) önce getirilmiş olmalıdır. Her kural

için nesneleri gruplayan bir mekanizma bulunur. Böyle bir mekanizma bir

nesne bağdaştırıcı (object adapter) veya nesne paketleyici/sarıcı (object

wrapper) olarak adlandırılır, fakat çoğunlukla sadece nesne sunucusu

inşasında kullanılan aletler kümesinden gizlidir.

Bir nesne bağdaştırıcısı (adapter) özel bir aktivasyon kuralının yazılımsal

gerçekleştirimi olarak en iyi anlaşılır. Ana konu nesne bağdaştırıcıların özel bir

kural için sadece konfigüre edildikleri, genel bir bileşen olduklarıdır.

(devam)

Sunucular (Servers)

Nesne bağdaştırıcı

(Object adapter): Fig. 3.8

27

Şekil 3.8 ‘de, bir nesne

sunucusunun farklı aktivasyon

kurallarını destekleme

organizasyonu görülmektedir.

Sunucular (Servers)

Nesne bağdaştırıcı (Object adapter):

Bir nesne bağdaştırıcısı (object adapter) bir veya daha fazla nesneyi kontrolü

altında tutar. Çünkü sunucu farklı aktivasyon kuralları gerektiren nesneleri

eşzamanlı olarak destekleme yeteneğine sahip olmalıdır, bu farklı kurallardaki

nesneler için birkaç nesne bağdaştırıcısı aynı sunucuda aynı zamanda

28

bulunabilir. Bir sunucuya bir invokasyon isteği geldiğinde istek ilk önce uygun

nesne bağdaştırıcısına sevk edilir (dispatch), Şekil 3.8 ‘de bu durum

görülmektedir.

Nesne bağdaştırıcılarının kontrol ettiği özel (spesifik) nesne ara yüzlerinden

haberdar olmaması da önemli bir gözlemdir. Aksi halde, asla genel olmazlar.

Nesne bağdaştırıcılar için tek önemli konu, bir invokasyon isteğinden bir nesne

referansını açar ve sonra referans edilmiş nesneye istek sevk edilir, fakat şimdi

özel bir aktivasyon kuralını takip ederler. İstek doğrudan nesneye

gönderildikten sonra, bir bağdaştırıcı bir nesnenin sunucu tarafındaki koçanı

(stub) için invokasyon isteğini ele alır.

Koçan (stub), iskelet (skeleton) olarak isimlendirilir ve normal olarak

nesne’nin ara yüz tanımlarından yaratılır, isteği açar (unmarshal) ve uygun

yöntemi çağırır (invoke).

(devam)

Nesne Bağdaştırıcı (1)

/\* Definitions needed by caller of adapter and adapter \*/

#define TRUE

#define MAX\_DATA 65536

/\* Definition of general message format \*/

29

struct message {

long source /\* senders identity \*/

long object\_id; /\* identifier for the requested object \*/

long method\_id; /\* identifier for the requested method \*/

unsigned size; /\* total bytes in list of parameters \*/

char \*\*data; /\* parameters as sequence of bytes \*/

};

/\* General definition of operation to be called at skeleton of object \*/

typedef void (\*METHOD\_CALL)(unsigned, char\* unsigned\*, char\*\*);

• Burada header.h dosyası bağdaştırıcı tarafından ve bu bağdaştırıcıyı

çağıran herhangi bir program tarafından kullanılmaktadır.

typedef void (\*METHOD\_CALL)(unsigned, char\* unsigned\*, char\*\*);

long register\_object (METHOD\_CALL call); /\* register an object \*/

void unrigester\_object (long object)id); /\* unrigester an object \*/

void invoke\_adapter (message \*request); /\* call the adapter \*/

Nesne Bağdaştırıcı (2)

typedef struct thread THREAD; /\* hidden definition of a thread \*/

thread \*CREATE\_THREAD (void (\*body)(long tid), long thread\_id);

30

/\* Create a thread by giving a pointer to a function that defines the actual \*/

/\* behavior of the thread, along with a thread identifier \*/

void get\_msg (unsigned \*size, char \*\*data);

void put\_msg(THREAD \*receiver, unsigned size, char \*\*data);

/\* Calling get\_msg blocks the thread until of a message has been put into its \*/

/\* associated buffer. Putting a message in a thread's buffer is a nonblocking \*/

/\* operation. \*/

• The thread.h file used by the adapter for using threads.

Nesne Bağdaştırıcı (3)

31

• Nesne-başına-alt

süreç politikasını

gerçekleştiren bir

bağdaştırıcının

ana parçası.

Sunucular (Servers)

Nesne bağdaştırıcı (Object adapter):

Örnek olarak, bir nesne bağdaştırıcısı’nın belli sayıda nesneyi yönettiğini göz

önüne alalım. Bağdaştırıcı kuralı gerçekleştirir, kural kendi nesnelerinin her

biri için tek bir kontrol alt sürecine sahiptir. Nesneye-özel (özgül) iskeletlerle

(skeletons) etkileşmek için istekler paketlenir ve açılır (marshal and

32

unmarshal). Her bir skeleton’ın işlemi gerçekleştirmesi beklenilir (umulur).

Burada in\_args, koçan (stub)tarafından açılması gereken bir byte dizisidir. Dizi

yöntemin bir tanımını tüm parametreleri için değerleri ile beraber içerir.

Dizinin tam formatı sadece stub tarafından bilinir, bu stub geçerli invokasyon

için de sorumludur. Parametre in\_size, in\_args ‘ın uzunluğunu belirtir (Dizinin

İnvoke(unsigned in\_size, char in\_args[], unsigned\* out-size, char\* out\_args[])

için de sorumludur. Parametre in\_size, in\_args ‘ın uzunluğunu belirtir (Dizinin

büyüklüğünü). Benzer olarak, tüm çıktılar stub tarafından bir diziye

paketlenilir, bu dizi out\_args isimlidir ve koçan tarafından dinamik olarak

oluşturulur.

Bu dizinin uzunluğu (büyüklüğü) ise çıktı parametresi out\_size tarafından

belirtilir.

(devam)

Sunucular (Servers)

Nesne bağdaştırıcı (Object adapter):

Burada invoke fonksiyonu (prosedürü) dinamik invokasyon için kullanılmıştır.

Mesaj tanımının en önemli kısmı uzaktaki istemcilerle bağdaştırıcı’nın

alışverişidir. Her bir istemci bir invokasyon isteğinin beş tane alana sahip bir

mesaja paketlenmesini umar.

33

Benzer olarak, bağdaştırıcı aynı yapıda bir mesajı cevap olarak geri döndürür.

Başlık dosyası, METHOD\_CALL tipli tanım anlamıyla bhir nesnenini sunucu

taraflı koçan’ında bağdaştırıcının yapacağı çağrının tanımını içerir. Bağdaştırıcı

iki prosedürü sağlar, bunlar bir sunucu tarafından bağdaştırıcıda nesnelerin

kayıtlanması (register) ve kayıttan silinmesi (unregister) olarak adlandırılır.

Bağdaştırıcının gerçekleştirimi için gerekli olarak, alt süreçlerin oluşturulması,

silinmesi ve iletişimi için gerekli bir alt süreç paketinin var olduğu varsayılır. Alt

süreçler arası iletişim tamponlarla sağlanılır. Özellikle her alt süreç kendi ilişkili

tamponuna sahiptir, bu tampondan get\_msg ile mesaj siler, put\_msg ile

tampona mesaj ekler.

Burada get\_msg bloklanmış bir işlem, put\_msg ise bloklanmamış bir işlemdir.

(devam)

Sunucular (Servers)

Nesne bağdaştırıcı (Object adapter):

Invokasyonların idaresindeki nesnelerin bağımsızlığı bağdaştırıcı’nın

gerçekleştiriminde önemlidir. Örnekteki gerçekleştirimde, nesneye-özgül

(spesifik) kod eklenilmemiştir.

Sonuçta, genel nesne bağdaştırıcılarının inşası mümkün hale gelir ve ana fikir

34

olarak bu bağdaştırıcılar ara katman yazılımı katmanına yerleştirilir. Nesne

sunucuları geliştiricileri sadece nesne geliştirmeye konsantre olmaya ihtiyaç

duyarlar, bunu basitçe bu nesnelerin invokasyonu hangi bağdaştırıcının kontrol

edeceğini belirleyerek yaparlar. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

35

Ders’in sonu…

36

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 9

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 9: Kod taşıma (göçü) (Code migration),

Yazılım etmenleri (Software agents).

Kod taşıma (göçü) (Code migration)

• Dağıtık sistemlerin genel veri göndermeye dayalı iletişiminden bahsedilir.

Bununla beraber, programların gönderilmesi ile bazen onların çalışması

sırasında gönderilebilmesi bir dağıtık sistemin tasarımını basitleştirir.

3

• Kod göçü için en zor durum heterojen sistemlerin durumudur.

Kod göçü yaklaşımları (Approaches to Code Migration) :

Kod göçünün bir çok farklı biçimi mevcuttur, bunlar sayesinde kod göçü’nün

neden yararlı olabileceği anlaşılmaktadır.

(devam)

Kod göçü yaklaşımları

Kod göçünün nedenleri (Reasons for Migrating Code) :

Geleneksel olarak, kod göçü dağıtık sistemlerde süreç göçü (process migration)

biçiminde yer alır, buna göre bütün bir süreç bir makineden bir diğerine

hareket eder. Çalışan bir sürecin farklı bir makineye taşınması masraflı ve

karmaşık bir iştir, bunu yapmak için iyi bir nedene ihtiyaç vardır. Bunun nedeni

4

daima, başarım olmaktadır.

Tüm sistemin başarımını artırabilmek için temel fikir, süreçlerin fazlaca-yüklü

makinelerden daha az-yüklü makinelere taşınmasıdır. Yük genelde, CPU

kuyruk uzunluğu veya CPU kullanılırlığı terimi olarak ifade edilir, fakat diğer

başarım göstergeleri de çok iyi kullanılır.

Yük dağıtım algoritmaları, işlemciler kümesine nazaran görevlerin/işlerin

ayrılması ve tekrar dağıtımı (allocation and redistribution) ile verilecek

kararlarda, hesaba-duyarlı (compute-intensive) sistemlerde önemli bir rol

oynar. Bununla beraber, bir çok modern dağıtık sistemde, hesaplama

kapasitesinin optimizasyonunun azlığı, iletişimin minimize edilmesinin

denemesine göre daha önemsizdir. Heterojenlik (altta yatan platform ve

bilgisayar ağlarının heterojenliği) nedeniyle, başarım artırımı, matematik

modeller yerine niteliksel (qualitative) nedenler ve/veya çıkarsama tabanlı olan

kod göçü sayesindedir.

(devam)

Kod göçü yaklaşımları

Kod göçünün nedenleri (Reasons for Migrating Code) :

Bir istemci-sunucu sistemini göz önüne alalım, sunucu büyük bir veritabanını

yönetsin. Eğer bir istemci uygulama, büyük miktarda veri içeren bir çok

veritabanı işlemi yapıyorsa, bu sunucu için istemci uygulamanın bir parçacı

olarak oluşturulabilir ve sadece cevap ağ üzerinden gönderilebilir. Aksi halde,

5

ağ sunucudan istemciye veri transferi ile meşgul durumda kalabilir. Bu

durumda, kod göçü, veri nerede bulunuyorsa oraya yakın bir yerde bu veriyi

işlemek üzere olan varsayım tabanlıdır.

Aynı sebep, sunucu’nun parçalarının göç etmesinde kullanılabilir (istemciye

doğru). Örneğin, bir çok etkileşimli veritabanı uygulamalarında, istemci

veritabanı işlemlerinin serisine daha sonra çevrilecek formun doldurulmasını

ister. Form’un istemci tarafında işlenmesi ve sunucuya tamamlanmış bir

formun sadece gönderilmesi, bazen görece büyük sayıda küçük mesajların ağ

üzerinden gönderilmemesini sağlar. Sonuç, istemcinin daha iyi başarım elde

etmesini sağlar, aynı zamanda formu işleme ve iletişim için daha az zaman

harcar.

Kod göçünü desteklemek, paralelleştirmeyi kullanarak başarım artışına

yardımcı olabilir, fakat genel karmaşıklık olmadığında (paralel

programlamanın zorluğu) bu iş yardımcı olur. (devam)

Kod göçü yaklaşımları

Kod göçünün nedenleri (Reasons for Migrating Code) :

Veb ‘te bilgi aramak buna tipik bir örnektir. Bu görece basittir.

Bir arama sorgusunun (query’sinin) küçük bir mobil program biçiminde

gerçekleştiriminde programın site site dolaşması basittir.

Böyle site site dolaşan mobil bir arama programının birkaç kopyasının her biri

6

farklı sitelere gönderilir, bu sayede doğrusal bir hızlanma, sadece bir program

örneği (instance)/kopyası kullanmaya göre daha başarılı olabilir.

Başarım artırmanın yanında, kod göçünü desteklemenin başka nedenleri de

bulunur. En önemlilerinden biri esnekliktir. Geleneksel olarak dağıtık

uygulamaları oluşturma yaklaşımı, uygulamayı farklı parçalara bölmek ve daha

sonra bu parçaların nerelerde çalıştırılacağına karar vermektir. Bu yaklaşım,

örneğin, çok taraflı (multitiered) farklı istemci

-sunucu uygulamalarında

mevcuttur.

Bununla beraber, eğer kod farklı makineler arasında taşınıyorsa, dağıtık

sistemin dinamik olarak konfigüre edilmesi mümkün olur. Örneğin bir

sunucunun standardize edilmiş bir ara yüzünü bir dosya sistemi için

gerçekleştirdiğini varsayalım.

(devam)

Kod göçü yaklaşımları

Kod göçünün nedenleri (Reasons for Migrating Code) :

Uzaktaki istemcinin dosya sistemine erişmine izin vermek için, sunucu uygun bir

protokol kullanır. Normalen, istemci-taraflı dosya sistemi gerçekleştiriminde (bu

protokolü temel alan), istemci uygulamasıyla bağlantılandırılmış (linked) olmalıdır.

Bu yaklaşım istemci uygulama geliştirildiğinde istemci de yazılım olarak hazır

halde olmasını gerektirir.

7

İstemci gerçekleştirimini gerekmedikçe sunucunun sağlamaması da bir

alternatiftir. Bir istemci sunucuya bağlanınca ancak gerçekleştirim olur. Bu

noktada istemci dinamik olarak gerçekleştirimi/uygulamayı download eder, gerekli

adımları takip ederek sunucuya çağrıda bulunur. Bu model, download etmek için ve

kodu başlatmak için (initialized) için gereken protokolün standardize edildiği bir

uzaktaki sunucudan kodun dinamik olarak taşındığı modeldir.

Download edilmiş kodun istemcinin makinesinde çalıştırılabilmesi gerekir. Bu

modelin en önemli avantajı istemci-taraflı yazılımın dinamik olarak download

edilmesidir, bu sayede (tüm yazılımın önceden kurulumu yapılmasıyla) sunucuyla

konuşmak için böyle bir ön kurulum yapılmasına gerek kalmaz. Bunun yerine

yazılım gerekirse taşınır ve bir daha gerekmiyorsa kaldırılır (discarded). Başka bir

avantajı da olabildiğince uzun süre ara yüzlerin standartlaştırılmasıdır, istemcisunucu protokolünü ve onun gerçekleştirimleri de sıklıkla değiştirilebilir.

(devam)

Kod göçü yaklaşımları

Kod göçünün nedenleri (Reasons for Migrating Code) :

Fig. 3.12

8

Şekil 3.12 ‘de, bir sunucu ile iletişimdeki bir istemcinin dinamik olarak konfigüre

edilme prensibi, istemci ilk önce gerekli yazılımı alır (fetch) ve sonra sunucuya

çağrıda (invoke) bulunur.

Kod göçü yaklaşımları

Kod göçü için modeller (Models for Code Migration) :

Kod göçü makineler arasındaki kod’un sadece taşınmasını önerse de, bu terim

daha geniş bir alanı kapsar. Geleneksel olarak, dağıtık sistemlerde iletişim,

süreçler arasında veri alış-verişini göz önüne alır. Kod göçü, makineler arasında

programların taşınmasını kapsayan geniş bir açıya sahiptir. Bu programlar

9

hedefte çalıştırılabilmelidirler.

Bazı durumlarda, süreç göçünde olduğu gibi, bir programın çalıştırılma

durumu, sinyalleşme müddetince, diğer çevresel parçalarında oldukça iyi

taşınmasına bağlıdır.

Kod göçünün bir çok farklı modelinin daha iyi anlaşılması için bir çatı

(framework) kullanılmaktadır. Buna göre; bir süreç üç segment’ten oluşur.

Kod segmenti, programın çalıştırılabilmesini sağlayan komut setini içerir.

Kaynak segmenti, süreç tarafından ihtiyaç duyulan harici kaynakların

referanslarını içerir, bunlar dosyalar, yazıcılar aygıtlar ve diğer süreçler olabilir.

Çalıştırma (execution) segment ise özel veri, yığıt (stack) ve program

sayacı (counter) içeren bir sürecin şu anki çalıştırılma durumunu depolamakta

kullanılır.

(devam)

Kod göçü yaklaşımları

Kod göçü için modeller (Models for Code Migration) :

Temel/çıplak (bare) minimum kod göçü sadece zayıf hareketlilik (weak

mobility) sağlar. Bu modelde sadece kod segmenti transfer edilmesi olasıdır,

belki bazı ilklendirme (initialization) verisi de transfer olabilir. Weak mobility

‘nin karakteristiğinden dolayı transfer edilmiş program ilk durumda (en

10

baştan) başlatılır. Buna örnek Java Applet’lerdir. Bu yaklaşımın avantajı,

basitliktir. Weak mobility sadece hedef makineyi (kodu çalıştıran hedef makine)

gerektirir, bu genelde taşınabilirliğin oluşturulmasını azaltır.

Weak mobility’nin aksine, güçlü hareketlilik (strong mobility) sağlayan

sistemlerde çalıştırma segmenti (execution segment) daha iyi transfer edilebilir.

Strong mobility’nin ana karakteristiği, çalışan bir sürecin durdurulabilmesi ve

sonra başka bir makineye taşınması ve ondan sonra da nerde kaldıysa oradan

çalıştırılmaya devam edilmesidir. Açıkça, strong mobility weak mobility’den

daha güçlüdür, fakat gerçekleştirimi daha zordur.

D’Agents isimli sistem strong mobility’yi de destekler. Mobility’nin weak veya

strong (zayıf veya güçlü hareketlilik) olmasına bakılmaksızın, daha ileri bir

karar göndericinin-başlattığı (sender initiated) göç ve alıcının-başlattığı

(receiver initiated) göç arasında yapılabilir.

(devam)

Kod göçü yaklaşımları

Kod göçü için modeller (Models for Code Migration) :

Göndericinin-başlattığı

(sender-initiated) göçte, göç kodun şu an

bulunduğu veya çalıştırıldığı makinede başlatılır. Tipik olarak, göndericinin

başlattığı göç programların bir hesap sunucusuna upload edilmesiyle olur.

Başka bir örnek bir Veb arama programına Internet üzerinden bir Veb

11

veritabanı sunucusuna sunucudaki sorguları gerçekleştirmek için

gönderilmesidir.

Alıcının-başlattığı (receiver-initiated) göçte, kod göçünü başlatmak hedef

makine tarafından yapılır. Java applet’ler buna da bir örnektir. Alıcının

başlattığı göç, sıklıkla göndericinin başlattığı göçten gerçekleştirim olarak daha

basittir. Bir çok durumda, kod göçü bir istemci ile sunucu arasında olur, burada

istemci göç için insiyatifi ele alır.

Kodun sunucuya güvenli (securely) upload edilmesi, göndericinin başlattığı göç

ile yapılır, sıklıkla istemcinin daha önceden sunucuda kayıtlanmış (registered)

ve yetkilendirilmiş (authenticated) olmasını gerektirir. Diğer bir deyişle,

sunucunu tüm istemcilerini bilmesi gerekir, bunun nedeni istemcinin

sunucunun kaynaklarına (örneğin disk) erişmek isteyebileceğidir. Böyle

kaynakları korumak esastır. Aksine, kodu download etmek, alıcının başlattığı

durumda, sıklıkla anonim olarak yapılır.

(devam)

Kod göçü yaklaşımları

Kod göçü için modeller (Models for Code Migration) :

Dahası, sunucu genelde istemcinin kaynakları ile ilgilenmez. Bunun yerine, kod

göçü istemci için sadece istemci-tarafında başarım iyileştirilmesi için yapılır.

Sonuçta, sınırlı sayıda kaynak korunur, bunlar bellek ve ağ bağlantılarıdır.

Weak mobility durumunda, eğer göç eden kod hedef süreç tarafından veya ayrı

12

bir süreç başlatılarak çalıştırılıyorsa, bir farklılık bulunmaktadır. Örneğin Java

applet’leri basitçe Veb gezgini tarafından download edilerek bu gezginin adres

uzayında çalıştırılır. Bu yaklaşımın yararı, ayrı bir süreç başlatmaya gerek

olmamasıdır.

Böylece hedef makine ile iletişime gerek kalmaz/kaçınılır. Bunun ana eksiği,

hedef sürecin zararlı veya bildirimsiz kod çalıştırımlarına karşı korunması

gerekliliğidir. Basit bir çözüm, bu göç etmiş kodun çalıştırılması için ayrı bir

sürecin yaratılması işinin işletim sisteminin kontrolüne bırakılmasıdır, fakat bu

çözümde kaynak erişmi problemlerini çözmez, sadece sözde kalır.

Çalışan bir süreci taşımak yerine, süreç göçü olarak belirtilen uzaktakini

kopyalamak (klonlamak, cloning) tarafından desteklenen strong mobility (güçlü

hareketlilik) ile yapılan göç olabilir.

(devam)

Kod göçü yaklaşımları

Kod göçü için modeller (Models for Code Migration) :

Süreç göçünün aksine, orijinal sürecin klonlanması onun tam bir kopyasını

verir, fakat şimdi bu kopya farklı bir makinede çalışmaktadır. Klonlamış süreç,

orijinal sürece paralel olarak çalışır.

UNIX sistemlerde, uzaktakini klonlama için bir çocuk (child) sürecin başlatılıp

13

(oluşturulup) onun uzaktaki bir makinede çalışmasına devam etmesini

sağlamaktır. Klonlamanın yararı, modelin halen bir çok uygulamada kullanılan

bir şekle yaklaşmasıdır.

Tek fark klonlanmış sürecin farklı bir makinede çalışmasıdır. Bu bakışla,

klonlama yoluyla göç, dağıtım saydamlığının iyileştirilmesinin en basit bir

yoludur.

Kod göçünün alternatifleri şöyle özetlenebilir:

(devam)

Kod göçü yaklaşımları

Kod göçü için modeller (Models for Code Migration) :

14

Fig. 3.13

Şekil 3.13 ‘de, kod göçü alternatifleri gösterilmektedir.

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

Kod göçünün gerçekleştirimlerini anlamak için kaynak segmenti üzerinde üç

adet süreçten kaynağa bağlama tipine bakmak lazımdır (process-to-resource

binding).

En güçlü bağlama, bir süreç bir kaynağın tanımlayıcısını işaret ettiğinde (refer)

15

oluşur. Bu durumda, süreç işaret edilen kaynağı ister, başka hiçbir şey istemez.

Örneğin URL kullanımı tanımlayıcı-ile-bağlanma

(binding by indentifier)

için bir örnektir. Aynı nedenlerle, yerel iletişim uç noktaları (endpoint veya port

diye adlandırılır) işaret etmek (refer) tanımlayıcı ile bağlanmadır.

Süreçten-kaynağa bağlanmanın daha zayıf bir biçimi, bir kayanğın sadece

değeri istenildiğinde olur. Bu durumda, süreç çalıştırılması, eğer başka bir

kaynak aynı değeri veriyorsa etki etmemektedir. Tipik bir örnek olarak değer

-

ile-bağlama

(binding-by-value) için bir program standart kütüphanelere

dayanıyorsa ve C veya Java ile programlanmışsa değer ile bağlanmış olur.

Böyle kütüphaneler yerel olarak daima elde edilebilir olmalıdırlar, fakat yerel

dosya sistemindeki tam yerleşimleri siteler arasında farklı olabilir. Özel

dosyalar olmaz, fakat içerikleri sürecin düzgün çalışması için önemlidir.

(devam)

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

Eğer bir süreç tek özel bir tip kaynağı ihtiyaç olarak gösteriyorsa bu traz

bağlanmaya tip-ile-bağlanma (binding-by-type) denilir. Bu bağlanmada

yerel aygıtlar; monitör veya yazıcı gibi cihazlar kastedilmektedir.

Kod göç ederken, kaynaklara olan referanslar sıklıkla değiştirilmek istenebilir,

16

fakat süreçten kuyruğa bağlama tipinde etki etmez. Kesin ve tam olarak bir

referansın nasıl değişseçeği, kaynağın kodla beraber hedef makineye

taşınmasına da bağlıdır. Kaynağın-makineye bağlanmasında birkaç durum

bulunur:

Bağlanmamış (eklenmemiş) kaynaklar

(Unattached resources), farklı

makineler arasında kolayca hareket ettirilebilir, bunlar tipik olarak (veri)

dosyalarıdır ve göç ettirilen programla sadece ilişkilidirler. Aksine bir

Bağlanmış kaynak ’ın (fastened resource) kopyalanması veya taşınması olası

olabilir, fakat sadece görece yüksek maliyetlidir. Tipik bağlanmış kaynaklar için

örnekler yerel veri tabanları ve tam içerikli Veb siteleridir. Böyle kaynaklar

teoride şu an bulundukları makineye bağımlı değildir ve sıklıkla onları başka

bir ortama (environment) taşımak uygun değildir (infeasible).

(devam)

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

Sabit kaynaklar

(Fixed resources), ima yoluyla özel bir makine veya ortama

bağlanmışlardır ve taşınmazlar (yani belirli bir kurala göre bağlıdırlar). Sabit

kaynaklar genelde yerel aygıtlardır. Sabit kaynaklara başka bir örnek ise yerel

bir iletişim uç noktası’dır (endpoint veya port olarak adlandırılır).

17

Dokuz farklı kombinasyon, üç farklı süreçten-kaynağa ve kaynaktan-makineye

bağlantıda mevcuttur.

(devam)

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

18

Unattached Fastened Fixed

Resource-to machine binding

Fig. 3.14

By identifier

By value

By type

MV (or GR)

CP ( or MV, GR)

RB (or GR, CP)

GR (or MV)

GR (or CP)

RB (or GR, CP)

GR

GR

RB (or GR)

Process-toresource

binding

GR = Establish a global system reference

MV = Move the resource

CP = Copy the value of the resource

Şekil 3.14 ‘de, kod başka bir makineye göç ederken yerel kaynakların

referanslarına göre yapılan eylemler görülmektedir.

CP = Copy the value of the resource

RB = Rebind process to locally available resource

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

Bir sürecin bir kaynağa tanımlayıcı ile bağlanması olasılığını göz önüne alacak

olursak; kaynak bağlanmamış (unattached) ise en iyisi göç eden kodla beraber

onu taşımaktır. Bununla beraber, kaynak diğer süreçler tarafından

paylaşılıyorsa bir alternatif olarak global bir referans oluşturmak, böylece bu

19

referansın makine sınırlarını geçmesini sağlayabilir. Böyle bir referansa örnek,

URL olarak verilebilir.

Kaynak bağlanmış (fastened) veya sabit (fixed) ise en iyi çözüm bir global

referans oluşturmaktadır. URL kullanmaktan daha fazlası haline gelebilecek bir

global referans oluşturmada, bir referans bazen engelleyici olacak kadar pahalı

olabilir. Yüksek kaliteli görüntüleri (image) işleyen bir çoklu ortam iş

istasyonunun hesaplama için sunucuya veri gönderip almasındaki yoğun veribandgenişliği isterleri buna örnek olabilir.

Başka bir örnek ise, global bir referans oluşturmanın kolay olmadığı, bir sürecin

göç ettirilirken bir yerel iletişim uç noktası’nın kullanıldığı durumdur. Bu

durumda, sabit (fixed) bir kaynak bir sürece tanımlayıcısı ile bağlanmış olabilir

(process is bound by the identifier).

(devam)

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

Temelde iki çözüm bulunur. Birinci çözüm, süreç göç ettikten sonra kaynak

makineye bir bağlantı kurması ve kaynak makinede ayrı bir süreç kurarak

(install) gelen tüm mesajları basitçe iletmesidir (forward). Bu yaklaşımın temel

eksiği kaynak makinedeki arızalarda, göç etmiş süreçle iletişimin kopmasıdır.

20

Alternatif çözüm ise, göç etmiş süreçle iletişimde olan tüm süreçleri içerir,

onların global referanslarını değiştirip, hedef makinedeki yeni iletişim uç

noktasına mesajlar gönderilir.

Değer-ile-bağlanma

(binding-by-value) ele alındığında durum farklıdır.

Sabit bir kaynakla değer-ile-bağlanma kombinasyonu olduğunda, örneğin bir

süreç belleğin süreçler arasında paylaşıldığını varsayar. Bu durumda global bir

referans oluşturmak için dağıtık paylaşımlı bellek mekanizmaları oluşturmak

gerekir.

Bağlanmış (fastened) kaynaklar değerleri ile bağlanabilirler, bunlar tipik olarak

çalışma zamanı kütüphaneleri’dir (runtime library). Normalen, böyle kaynaklar

hali hazırda hedef makinede de bulunur veya göçten önce kopyalanır. Bir global

referansın oluşturulması, büyük miktarda verinin kopyalanmasında (sözlük

veya metin işleme sistemleri gibi) en iyi alternatiftir.

(devam)

Kod göçü yaklaşımları

Kod göçü ve yerel kaynaklar ((Code) Migration and Local Resources) :

Bağlanmamış kaynaklar

(unattached resources) en kolay durumdur. En iyi

çözüm kaynağı belli sayıda süreç paylaşmadan yeni hedefe taşımak veya

kopyalamaktır. Tip ile bağlama da bu durum da önem kazanır.

Heterojen sistemlerde göç ((Code) Migration in Heterogeneous

21

Systems) :

Genelde, dağıtık sistemler her birinin kendi makine mimarisi ve işletim

sistemleri bulunan heterojen platformlar topluluğundan oluşur. Böyle

sistemlerde göç etmeyi her bir platformun da desteklemesi gerekir, bunun için

kod segmenti’nin her bir platform da çalışabilir olması gerekir, belki orijinal

kaynağın daha sonra tekrar derlenmesi (bu farklı platformlarda) gerekebilir.

Böylece, çalıştırma (execution) segmenti’nin düzgün olarak her bir platformda

sunulduğundan emin olmak gerekir (represented).

Problemler, zayıf hareketlilik (weak mobility) söz konusu olduğunda sadece

azalmaktadır. Bu durumda, hiçbir çalışma zamanı bilgisi’nin makineler

arasında iletilmesi gerekmez, kaynak kodun derlenmesi yeterli olur, fakat farklı

kod segmentleri’nin oluşturulması her bir potansiyel hedef platform için gerekir

(Weak mobility’de kod segment’i iletilmektedir).

(devam)

Kod göçü yaklaşımları

Heterojen sistemlerde göç ((Code) Migration in Heterogeneous

Systems) :

Güçlü hareketlilik

(strong mobility) durumunda, en büyük problem

çalıştırma (execution) segmentinin transfer edilmesidir. Bu segment oldukça

fazla platform-bağımlı (sürecin çalıştırılacağı yer) olmaktadır. Sadece hedef

makine aynı mimari ve tam olarak aynı işletim sistemiyle çalışıyorsa bu

22

segment’i hiç değişiklik yapmadan göç ettirmek mümkündür.

Süreç için özel veriyi çalıştırma (execution) segmenti içerir, onun geçerli yığıtı

(stack) ve program sayacı (counter) için de bu veri özeldir (içerilmektedir). Yığıt

kısmen geçici veri içerir, bunlar yerel değişkenlerin değerleridir, fakat kayıtçı

(yazmaç veya register olarak bilinir) değerleri gibi platform-bağımlı bilgi

içerebilirler. En önemli gözlem, platform-özgül (spesifik) veriye bağlı bir

çalıştırmadan kaçınmak, segmentin farklı bir makineye transfer edilmesinden

ve orada çalıştırılmasından daha kolay olduğudur.

C ve Java gibi prosedürel dillerde çalışan bir çözüm bulunur (şekilde

gözükmektedir). Kod göçü programın çalıştırılmasında özel noktalarda

sınırlandırılmıştır. Özellikle, bir sonraki subrutin çağrıldığında (sadece) göç

etme olur. C dilinde bir fonksiyon, subrutin, Java dilinde bir metod ise subrutin

olarak söylenebilir.

(devam)

Kod göçü yaklaşımları

Heterojen sistemlerde göç ((Code) Migration in Heterogeneous

Systems) :

Çalışma zamanı

(runtime) sistemleri, program yığıtı’nın (stack) kendi

kopyalarını bulundururlar/bakarlar, fakat makine-bağımsız bir yoldan bunu

yaparlar. Bu kopyaya göç yığıtı (migration stack) adı verilir. Göç yığıtı bir

subrutin (fonksiyon veya metod) çağrıldığında veya bir çalıştırma (execution)

23

bir subrutin’den geri döndüğünde güncellenir.

Bir subrutin çağrıldığında, çalışma zamanı (runtime) sistemi en son çağrıdan

beri yığıta atılmış (pushed) olan veriyi paketler (marshall). Bu veri yerel

değişkenlerin değerlerini sunar, bunlarla beraber yeni çağrılmış subrutin için

parametre değerleri de bulunur.

Paketlenmiş (marshalled) veri, daha sonra göç yığıtına (migration stack) atılır

(pushed), bunun yanında çağrılan subrutin’in bir tanımlayıcısı (identifier) ‘de

bulunur. Ayrıca, çalıştırma nerede devam edecekse oranın adresi, arayan

(caller) subrutinden döndüğünde bir sıçrama (jump) etiketi biçiminde oldukça

iyi olarak göç yığıtı’na (migration stack) atılır (pushed).

(devam)

Kod göçü yaklaşımları

Heterojen sistemlerde göç ((Code) Migration in Heterogeneous

Systems) :

Bir subrutin çağrıldığı noktada kod göçü olmuşsa, çalışma zamanı (runtime)

sistemi ilk önce tüm global programa-özgül veriyi, yani çalıştırma (execution)

segmentinin parçası biçiminde olan veriyi paketler (marshall). Makineye-özgül

veri göz ardı edilir, olabildiğince şu anki yığıtta göz ardı edilir.

24

Paketlenmiş veri göç yığıtı (migration stack) ile beraber hedefe transfer edilir.

Ayrıca, hedef kendi makine mimarisi ve işletim sistemine uygun binary ‘leri

içeren uygun kod segmenti’ni yükler. Paketlenmiş olan çalıştırma (execution)

segmenti’ne ait veri’nin paketi açılır (unmarshaled) ve yeni bir çalışma zamanı

(runtime) yığıtı, göç yığıtı’nın paketinin açılmasıyla (unmarshaling) oluşturulur.

Çalıştırma (execution) kaldığı yerden çok basitçe orijinal site (yer) olarak

adlandırılan subrutin’e girilerek devam ettirilebilir. Açıktır ki, bu yaklaşımın

sadece ve sadece derleyici, bir subrutin’e girilmiş veya çıkılmış olduğunda göç

yığıtı’nı güncelleyecek kodu oluşturursa uygun olduğu görülür.

Bir subrutin’in bir (makine-bağımsız) sıçrama olarak gerçekleştirilebildiği

halden bir geri dönüş’e (subrutine return) çağrıyı yapan (caller) ‘ın kodundaki

etiketlerin derleyici tarafından oluşturulmasıyla izin verilmesi gerçekleşir.

(devam)

Kod göçü yaklaşımları

Heterojen sistemlerde göç ((Code) Migration in Heterogeneous

Systems) :

Ayrıca, bir uygun çalışma zamanı (runtime) sistemine de ihtiyaç vardır. Belli

sayıda sistem bunu başarı ile yapmıştır. Heterojenlik ile gelen problemler,

taşınabilirlikle gelen problemlere göre aynıdır.

25

Heterojen sistemlerde kod göçü, yüksek seviye taşınabilirliğe sahip Java gibi

diller ile sağlanır. Tüm böyle çözümler sanal makine tabanlıdır (rely on a virtual

machine). Bununda ana eksiği sanal makine yaklaşımından dolayı sadece özel

bir dile saplanıp kalmaktır, bu hareketliliği (mobility) kısıtlar.

(devam)

Kod göçü yaklaşımları

Heterojen sistemlerde göç ((Code) Migration in Heterogeneous

Systems) :

26

Fig. 3.15

Şekil 3.15 ‘te, bir heterojen ortamda bir çalıştırma (execution) segmentinin

göçünü destekleyen bir göç yığıtı’nın oluşturulma/bakım prensibi

görülmektedir.

Yazılım etmenleri (Software agents)

• Programların ve süreçlerin hareketliliğinden bahsedildiğinde, süreçler hakkında

birden çok görüş ortaya atılmaktadır.

• Yazılımsal etmenler (software agents), bir işi yapmak üzere birbirleri ile

ortaklaşa çalışabilen otonom (autonomous) birimlerdir, yani kendi başlarına

27

hareket edebilirler, bunlar büyük ihtimalle uzakta bulunmakta (yerelde değil)

olan etmenlerdir.

• Dağıtık sistemlerde etmenler gün geçtikçe daha çok rol oynar hale gelmiştir.

Bununla beraber, bir süreç tam olarak ne ise etmen de ona benzer bir yapıyı

barındırabilmektedir.

Yazılım etmenleri (Software agents)

Dağıtık sistemlerde yazılımsal etmenler:

Bir etmenin tam olarak neyi ifade ettiği ile ilgili bir çok yaklaşım vardır. Bazı

çalışmalarda, yazılımsal etmen, kendi çevresinde (ortamında) değişiklikler

oluşturan ve çevresi ile etkileşen, diğer kullanıcılar veya diğer etmenler ile

ortaklaşa işler yapabilen bir otonom süreç olarak tarif edilmektedir.

28

Bir etmeni bir süreçten ayıran başlıca özellik, kendi kendine etkileşmeye

girebilmesi ve insiyatifi ele alabilmesidir.

Daha geniş çapta bir tanım yapacak olursak; farklı tipte süreçler kolayca bir

etmen olarak nitelendirilebilir. Daha iyi bir tanım yapacak olursak; farklı tipte

etmenlerin bulunduğunu belirtebiliriz. Buna göre; literatürde bir çok yazılımsal

etmen taksonomisi (kategorize etme çalışması) bulunmaktadır, fakat tek bir

taksonomiye ulaşılması oldukça güç görülmektedir.

Yazılım etmenleri (Software agents)

• Otonom olmanın yanı sıra, etmenlerin önemli bir cephesi de diğer etmenler ile

ortaklık yapabilmeleridir. Bu otonomluğun ve ortaklaşmanın kombinasyonu,

işbirlikçi etmenler (collaborative agents) sınıfının oluşturulmasını sağlamıştır.

• Bir işbirlikçi etmen, çoklu etmene sahip bir sistemin bir parçasıdır. Bu sistemde

29

etmenler genel bir amaca işbirliği yaparak ulaşmaya çalışırlar. Tipik bir

uygulamada işbirlikçi etmenler bir görüşme için bir araya gelirler. Her bir

katılımcı bir etmen tarafından temsil edilir.

• Bu etmenler kullanıcının kişisel ajandasına erişebilmektedirler. Bireysel kısıtlar,

zaman, yerleşim, gezi gibi kısıtlardır. Dağıtık sistemler bakış açısıyla, hangi

bilginin değiş tokuş edildiği ve nasıl işlendiği daha az önemlidir. En önemli konu

iletişimin nasıl yapıldığıdır.

Yazılım etmenleri (Software agents)

• Mobil etmenler, diğer etmen tiplerinden ayrılmaktadırlar. Bir mobil etmen

basitçe farklı makineler arasında hareket edebilme yeteneğine sahiptir. Kod

göçü konusu bununla ilişkilidir. Mobil etmenler sıklıkla güçlü hareketliliğin

(strong mobility) desteklenmesini isterler.

30

• D’Agents systemi örnek olarak ele alındığında, etmenlerin ve zayıf hareketliliğin

(weak mobility) kombinasyonu kullanışlıdır. Mobil etmenlerin bir diğer

kullanımına örnek olarak, heterojen ağlardan oluşan Internet üzerinde

dağıtılmış bilgiyi toplayarak bundan çıkarım yapan mobil etmenler verilebilir.

Yazılım etmenleri (Software agents)

• Bu tarz etmenlerin haricinde; ara yüz etmenleri

(interface agents), bilgi

etmenleri

(information agents) gibi tipler de bulunmaktadır.

31

Özellik (Property)

Tüm etmenler için genel

olma?

(Common to all agents?)

Tanımlama (Description)

Otonom

(Autonomous) Evet Kendisi gibi davranır (Can act on its own)

Reaktif (Reactive) Evet

Çevresindeki de

ğişimlere süreli olarak cevap

verir (Responds timely to changes in its

environment)

Proaktif (Proactive) Evet Çevresine etki eden aksiyonları ba

şlatır

(Initiates actions that affects its environment)

İleti

şimsel

(Communicative) Evet

Bilgiyi di

ğer kullanıcı ve etmenler ile de

ğiş

toku

ş edebilme (Can exchange information

(Communicative) Evet toku

ş edebilme (Can exchange information

with users and other agents)

Sürekli

(Continuous) Hayır Görece uzun bir ömrü vardır (Has a relatively long lifespan)

Mobil (Mobile) Hayır Bir alandan di

ğerine göç edebilirler (Can

migrate from one site to another)

Uyarlanabilir

(Adaptive) Hayır

Ö

ğrenme yetenekleri vardır (Capable of

learning)

Yazılım etmenleri (Software agents)

Etmen teknolojisi:

Etkin olarak bir etmen sistemini geliştirmek için ortada bir destek

bulunmamaktadır. Bunun yanı sıra, bir dağıtık sistemde genel-kullanımlı etmen

bileşenlerini izole etmemiz gerektiğinde, bu bileşenleri dahil edilmesidir, buna

örnek olarak ara katman yazılımı verilebilir.

32

Bir başlangıç noktası olarak, Zeki Fiziksel Etmenler için Kurum (Foundation for

Intelligent Physical Agents, FIPA) yazılımsal etmenler için genel bir model

üretmiştir. Bu modelde, bir etmen platformu herhangi bir çoklu sistem için

temel servislerin sağlanmasında gereklidir.

Yazılım etmenleri (Software agents)

Etmen iletişim dili (Agent Communication Language) :

Dağtıtık sistemler bakış açısıyla, normal süreçlerden etmenleri ayırdığımızda

onların uygulama-seviyesindeki bir iletişim protokolü sayesinde

etkileşebildikleri görülmektedir.

33

Bu protokole, Etmen iletişim dili (Agent Communication Language, ACL)

denilmektedir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

34

Ders’in sonu…

35

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 10

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 10: İsimlendirme (Naming),

Varlıkların isimlendirilmesi (Naming Entities).

Đsimlendirme (Naming)

Bilgisayar sistemlerinin erişim açısından isimlendirilmesi önemli bir konudur.

Dağıtık bir sistemde, bir isimlendirme sisteminin gerçekleştirimi sıklıkla birden

çok makine arasında dağıtılması ile olur. Bu dağıtım, isimlendirme sisteminin

etkinliği ve ölçeklenebilirliğinde büyük rol oynar.

İsimlendirme için insan

-dostu organizasyon ve gerçekleştirim mevcuttur. Tipik

3

örnek olarak dosya sistemleri ve Dünya çapında Veb (World Wide Web, WWW)

verilebilir.

İkinci olarak, mobil varlıklar içinde özellikle isimlendirme önemlidir. Sıklıkla

mobil telefon konusunda, yerleşimden-bağımsız tanımlayıcılar kullanılmaktadır.

Üçüncü olarak isim organizasyonu’nda artık kullanılmayan ve/veya erişim

yapılmayan referans isimlerin kaldırılması gerekmektedir.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayındaki farklı varlıkların nasıl isimlendirileceği önemli bir konudur.

Örneğin Internet Alan İsim Sistemi (Domain Name System) ve OSI ‘nin X.500 ‘ü

büyük-ölçekli isimlendirme servislerine örnektir.

4

İsimler, Tanımlayıcılar ve Adresler (Names, Identifiers and Addresses)

Bir varlığı işaret etmede (refer) kullanılan dağıtık sistemdeki bir isim bir bit veya

karakter katarı (string) olur. Bir dağıtık sistemdeki bir varlık pratikte her şey

olabilir. Host’lar, yazıcı’lar, disk’ler, dosyalar gibi kaynaklar tipik örneklerdir.

Diğer en iyi bilinen örnekler açıkça isimlendirilmiş süreçler ve kullanıcılar, posta

kutuları, haber grupları, Veb sayfaları vs. gibi varlıklardır.

Varlıklar üzerinde çalışılabilinir. Örneğin bir yazıcı gibi bir kaynak bir

doküman’ın basılması için işlemleri içeren bir ara yüzü önerir ve bir yazıcı işinin

durumunu isteyebilir. Bir ağ bağlantısı için veri gönderim-alımında da

kullanılabilirler. Bir varlığı işletmek için (to operate on an entity) ona erişmek

gerekir, bunun için bir erişim noktası’na (access point) ihtiyaç duyulur. Bir

erişim noktası bir başkası, fakat özel olan, dağıtık sistemdeki bir çeşit varlıktır.

Bir erişim noktası bir adres olarak isimlendirilir.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsimler, Tanımlayıcılar ve Adresler (Names, Identifiers and Addresses)

Bir varlık birden fazla erişim noktasını önerebilir. Karşılaştırma olması için, bir

kişiye ait telefon onun erişim noktası olarak gözükür, böylece telefon numarası

bir adres gibidir. Bu günlerde bir çok insanın birden çok telefonu mevcut ve her

5

bir numara ona nasıl erişildiğini belirleyen bir noktadır. Dağıtık bir sistemde, bir

erişim noktası örneği özel bir sunucuda koşan bir host (özel bir makine’de) için

bir IP-adresi ve port numarası kombinasyonundan oluşan adres biçimindedir.

Bir varlık erişim noktasını zaman içinde değiştirilebilir. Örneğin bir mobil

bilgisayar başka bir yerleşime taşınabilir ve sıklıkla farklı bir IP adresine

(önceden sahip olduğundan farklı) sahip olabilir. Bir insan başka bir şehre

taşınmasına benzemektedir. Bir adres, özel bir çeşit isimdir, bir varlığın erişim

noktasını işaret eder. Bir erişim noktası sıkıca bir varlık ile ilişkilidir.

Örneğin bir dağıtık sistemdeki FTP isteklerine göre idare uygulanması artık

önceki yerinde olmayan bir sunucu için tekrar organize olmayı gerektirir. Erişim

noktası anlık değişirse geçersiz bir referans’a sahip olunabilir veya bir varlık

tekrar atanabilir.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsimler, Tanımlayıcılar ve Adresler (Names, Identifiers and Addresses)

Böyle durumlarda FTP servisinin ayrı bir isimle bilinmesi, ilgili FTP

sunucusunun adresinden bağımsız olması daha iyi bir sonuç verir. Benzer olarak,

bir varlık bir den çok erişim noktasına sahip olabilir. Daha kolay ve esnek bir

6

kullanım için varlığın ismi onun adresinden bağımsız olmalıdır. Böyle bir isme,

yerleşimden bağımsız

(location independent) denilir. Adresler ayrıca, özel bir

davranış şeklini isim tiplerine göre hak ederler ve bir varlığı eşşiz (unique) olarak

ifade etmede isim gibi kullanılırlar. Gerçek bir tanımlayıcı (identifier) aşağıdaki

özelliklere sahiptir.

1.) Bir tanımlayıcı en fazla bir varlığı işaret eder.

2.) Her bir varlık en fazla bir tanımlayıcı tarafından işaret edilir.

3.) Bir tanımlayıcı daima aynı varlığı işaret eder (yani asla tekrar kullanılmaz).

Tanımlayıcı kullanımı ile bir varlığın işaret edilmesi daha kolay hale gelir.

Örneğin, iki süreç’in her biri bir varlığı, bir tanımlayıcı anlamıyla işaret eder. Bu

süreçler aynı varlığı işaret edip etmediğini kontrol etmek için iki tanımlayıcının

aynı olup olmadığına bakılır.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsimler, Tanımlayıcılar ve Adresler (Names, Identifiers and Addresses)

Böyle bir test, eğer iki süreç düzgün tanımlanmamış isimler kullanıyorsa yeterli

olmaz. Yani Ahmet, Mehmet (genel) tek bir kişi için eşşiz (unique) bir isim

değildir. Eğer bir adres farklı bir varlığa tekrar atanırsa, bir adresi bir tanımlayıcı

7

olarak kullanamayız. Örneğin bir A kişisinin telefon numarası, A kişisi başka bir

şehre taşındığı için B kişisine atanmış olsun. Artık A kişisini arayanların

karşısına B kişisi çıkmaktadır.

Adresler ve tanımlayıcılar iki önemli isim tipleridir ve bir çok farklı amaçla

kullanılırlar. Birçok bilgisayar sisteminde adresler ve tanımlayıcılar, makine

tarafından okunabilir biçimde (sadece) sunulurlar, bunun anlamı bit katarı

(string) biçiminde olmalarıdır. Örneğin Ethernet adresleri 48 bit uzunluğundadır

= MAC ID.

İnsanların kullanımına uygun “human-friendly names” bulunmaktadır,

insanın okuyabileceği gibi karakter katarları ile ifade edilen isimlerdir. Örneğin

UNIX sistemlerde karakter katarları 255 karaktere kadardır ve tamamen

kullanıcı tarafından belirlenir. Benzer olarak DNS isimleri de görece basit

karakter katarlarıdır.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayı (Name spaces):

İsimler dağıtık sistemde isim uzayında organize edilirler. Bir isim uzayı,

etiketlendirilmiş, iki tane düğüm tipine sahip yönlü graf (çizge) olarak sunulur.

Bir yaprak düğüm (leaf node), devam eden ucu (köşesi) olmayacak özellikte

8

isimlendirilmiş bir varlıktır. Bir yaprak düğüm genelde sunumunu yaptığı varlık

hakkında bilgi depolar (örneğin onun adresini) böylece bir istemci ona erişebilir.

Alternatif olarak bu varlığın durumunu depolar, bu durumda dosya sistemindeki

sunulan bütün dosyayı içeren bir yaprak düğüm bulunur.

Bir yaprak düğümün aksine, bir dizin düğümü (directory node) belli sayıda

devam eden uca (köşe) sahiptir, her bir uç bir isimle etiketlenmiştir. Bir

isimlendirme çizgesinde her bir düğüm dağıtık sistemdeki başka bir varlık olarak

ele alınır ve özellikle ilişkilendirilmiş bir tanımlayıcıya sahiptir. Bir dizin

düğümü, devam eden ucun (outgoing edge) bir çifti gösterdiği bir tabloyu

depolar. Bu çift şöyledir: (edge label, node identifier).

Böyle bir tablo bir dizin tablosu (directory table) olarak adlandırılır. Aşağıdaki

şekilde n

0 düğümü, sadece dışa doğru giden (outgoing) uçlara sahip, fakat içeri

doğru gelen (incoming) uçlara sahip değildir. Buna kök düğüm (root node)

denilir ve bir çok isim çizgesinde mevcuttur. (devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayı (Name spaces):

9

Fig. 4.1

Şekil 4.1 ’de, tek bir kök düğüme sahip genel bir isimlendirme çizgesi

görülmektedir.

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayı (Name spaces):

İsim çizgesinde her ucun bir etiketi bulunur.

N: <label\_

1, lable\_

2, …, label\_

n

>

10

Burada N, katedilen yoldaki ilk düğümdür. Böyle bir dizi (diziliş) bir yol ismi

(path name) olarak isimlendirilir. Eğer bir yol ismi’ndeki ilk düğüm isimlendirme

çizgesinin kök’ü ise, buna mutlak yol ismi (absolute path name) denilir. Aksi

halde buna göreli yol ismi (relative path name) denilir. Yani kök ile bağlayan bir

adres (yol olarak) varsa mutlak (absolute), fakat aksine çocuk düğümden

başlanmışsa bu göreli (relative) bir adres (yol olarak) kabul edilir.

İsimler daima isim uzayında organize edilirler, bunun sonucu olarak, bir isim

daima özellikle bir dizin düğümü için tanımlanır. Bu bakışla, mutlak isim yanlış

olur. Bazen de yerel ve global isimlerde karmaşa meydana gelebilir. Bir global

isim, bir sistem de bir ismin kullanıldığı yerin önemi olmadan bu ismin aynı

varlığı belirtmesidir.

Diğer bir deyişle, global bir isim hep aynı dizin düğümü’nü belirtecek şekilde

yorumlanılır. Aksine bir yerel isim (local name), bu ismin kullanıldığı yere

bağımlı olarak yorumlanılır. Aradaki fark, bir yerel ismin temelde bir göreli isim

olması ve içerildiği dizinin (klasör) kesin olarak (implicitly) bilindiğidir.

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayı (Name spaces):

Bir çok dosya sisteminde isimlendirme çizgesi bu tarzdır. Bununla beraber, her

bir uç (edge) etiket dizisinin bir yol ismi’ni gösterdiği yazım yerine, dosya

sistemindeki yol isimleri özel bir ayraçla ayrılan [(“/”) slash] etiketlerden oluşur.

11

Bu karakter (“/”) ayrıca bir yol ismi’nin mutlak olup olmadığını da gösterir.

Örneğin yukarıdaki şekilde n

0: <home, steen, mbox> olarak verilen katar

“/home/steen/mbox” katarı şeklinde gösterilebilir. Birkaç yol da aynı düğümü

gösterilebilir, bununla beraber bu düğüm farklı yol isimleri ile de gösterilebilir.

Plan

9 ’da tüm kaynaklar (süreçler, makineler (hosts), I/O aygıtları, ağ ara

yüzleri) normal geleneksel dosya isimlendirmesine göre isimlendirilmişlerdir. Bu

yaklaşım, bir dağıtık sistemde tüm kaynaklar için tek bir isimlendirme çizgesi

gerçekleştirimine benzerdir. İsim uzayını organize etmek için bir çok yol

mevcuttur. Bir çok durumda bir isim uzayı hiyerarşik yapıda bir ağaç şeklinde

organize edilmiştir. Yukarıdaki şekildeki örnek bir yönlü döngüsel olmayan

çizge

(directed acyclic graph) olur. Böyle bir organizasyonda döngüye izin

verilmez.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayı (Name spaces):

Geleneksel UNIX isimlendirmesinde, bir isimlendirme çizgesi, bir dosya dizinini

gösteren bir dizin düğümü bulunur. Buradaki bir yaprak düğümü bir dosyayı

göstermekte/temsil etmektedir.

12

İsimlendirme çizgesinde kök düğüm tarafından gösterilen tek bir kök dizin

olabilir. Dosya sisteminin tüm gerçekleştiriminin bir iç parçası olarak

isimlendirme çizgesi bulunabilir. Gerçekleştirim blok dizileri (sürekli) ile

oluşturulan ve bu blok dizilerinin (mantıksal disk’e ait) bir boot blok, bir

superblok, bir index düğümleri dizisi ve dosya veri blokları’na

bölünmesiyle oluşturulur.

Şekil 4.2 ’de, sürekli disk bloklarına ait mantıksal bir disk üzerinde UNIX dosya

sisteminin gerçekleştirimi için genel organizasyon görülmektedir.

(devam)

Fig. 4.2

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim uzayı (Name spaces):

Özel bir veri bloğu olan boot blok, sistem açıldığında (booted) otomatik olarak

ana belleğe yüklenilir. Boot blok, işletim sisteminin ana belleğe yüklenilmesinde

kullanılır. Superblok tüm dosya sisteminin bilgisini içerir. Dosya sisteminin

13

büyüklüğünün bilgisi hangi blokların diskte henüz ayrılmadığı (allocated), hangi

inode’ların henüz kullanılmadığı gibi bilgilerdir. Inode’lar, bir index numarası ile

işaret edilen, sıfır numarasından başlayan (bu inode kök dizini göstermektedir)

düğümlerdir.

Her bir inode, disk’te bulunan dosya ile ilişkili verinin nerede olduğunun tam

olarak bilgisini içerir. Ayrıca, bir inode kendi sahibi hakkında, yaratılma zamanı

ve en son üzerinde değişiklik yapılma zamanı, korunması hakkında bilgi içerir.

Sonuçta, bir inode’un index numarası verildiğinde ilişkilendirildiği dosyaya

erişim sağlanabilir. Her bir dizin dosya kadar iyi olarak gerçekleştirilebilir. Kök

dizin dosya isimleri ile inode’ların dosya numaraları arasında bir haritalamayı

içerir. Bir inode index numarası bir isimlendirme çizgesindeki bir düğüm’ün

tanımlayıcısından sorumludur.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim Çözümleme (Name resolution):

Verilen bir yol ismi için düğümde depolanan herhangi bir bilgi için bakılması

sürecine isim çözümleme

(name resolution) denilir.

14

Örneğin;

N : <label\_

1, label\_

2, …, label\_

n>

yol ismine bakacak olursak, burada ilk önce N node’unun isminin çözümlenmesi

ile başlanarak, daha sonra dizin tablosu’ndaki label\_1 düğümü için bulunan isme

bakılır, bu işlem label\_

1 ‘in işaret ettiği düğümün tanımlayıcısını geri döndürür.

İsim çözümleme daha sonra tanımlanan düğümde devam ederek label\_

2 ‘ye

dizin tablosu’ndan bakarak olayı devam ettirir. Bir isim bakma (lookup) isim

çözümleme süreci nerede devam ediyorsa oradaki bir düğümün tanımlayıcısını

geri döndürür.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

İsim Çözümleme (Name resolution):

Özellikle, tanımlanmış node’un dizin tablosu’na erişmek gereklidir. UNIX

sistemlerde bir düğüm tanımlayıcı bir inode’un index numarası olarak

gerçekleştirilebilir.

15

Bir dizin tablosuna erişimin anlamı, geçerli veri diskte nerede ise ilk önce inode

tarafından bulunarak okunması, daha sonrada dizin tablosunu içeren veri

bloklarının okunmasıdır.

Kapatma Mekanizması (Closure Mechanism)

İsim çözümlenmesinin nerede ve nasıl başlayacağı genelde bir kapatma

mekanizması

(closure mechanism) olarak belirtilir. Temelde, bir kapatma

mekanizması bir isim uzayındaki ilk node’u (isim çözümlemesinin başladığı isim

uzayı) seçmekte uğraşır. Bu mekanizmanın zorluğu bazı parçaların örtülü

(dolaylı) ve çok farklı olabileceği ve birbirleri ile karşılaştırılmalarının

güçlüğüdür. Örneğin bir UNIX dosya sisteminde isimlendirme çizgesi’nin isim

çözümlenmesi, kök dizin‘in inode’unun mantıksal disk gösterimindeki dosya

sisteminde ilk inode olmasıdır.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

Kapatma Mekanizması (Closure Mechanism)

Geçerli byte offset’i superbloğun diğer alanlarındaki değerlerden hesaplanılır,

superblok’un iç organizasyonunda bulunan işletim sisteminde katı-kodlanmış

(hard

-coded) bilgi ile bu hesap yapılır. Örneğin bir dosyanın isim gösterimi

16

“/home/steen/mbox” olsun. Bu ismi çözmek için kök düğüm’ün dizin tablosu’na

(uygun isimlendirme çizgesindeki kök düğüm) erişmek gerekmektedir.

Global ve yerel isimlerin dağıtık sistemlerde kullanımına bakacak olursak; bir

yerel isim, bir çevresel değişkendir. Örneğin UNIX sistemlerde HOME isimli

değişken bir kullanıcının ev dizinini gösterir. Her bir kullanıcı bu değişkenin

kendi kopyasına sahiptir, bu global olarak başlatılır, sistem çapında o

kullanıcının ev dizini ile alakalıdır. Kapatma mekanizması, değişkenin isminin

bir kullanıcıya

-özgül (spesifik) tablodan bakılarak çözülmesinden emin olur.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

Bağlantılandırma ve Bağlama (Linking and Mounting):

Takma isim’lerin (alias) kullanımı isim çözümleme ile oldukça fazla alakalıdır. Bir

alias bir varlık (aynı varlık) için farklı bir isimdir. Bir alias örneği olarak bir

çevresel değişken verilebilir. Bir alias’ın gerçekleştirimi için iki farklı yol bulunur.

17

İlk yaklaşımda, birden çok mutlak yol isim’lerinin bir isimlendirme çizgesinde

aynı düğümü işaret etmesine basitçe izin verilir. Örneğin UNIX sistemlerde

“/keys” ve “/home/steen/keys” yol isimleri kalıcı bağlantılar

(hard links)

olarak belirtilir.

İkinci yaklaşımda, bir yaprak düğüm tarafından bir varlık gösterilir. N olarak

söylenilir, fakat varlığın durumu veya adresini depolamak yerine, düğüm mutlak

yol ismi’ni depolar. N ‘yi veren bir mutlak yol ismi’nin çözümlenmesinde, isim

çözümleme N ‘de depolanan yol ismini geri döndürür, böylece bu noktadan isim

çözmeye devam eder. UNIX sistemlerde bu prensip sembolik bağlantılar

(symbolic links) olarak kullanılır.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

Bağlantılandırma ve Bağlama (Linking and Mounting):

İsim çözümleme sadece tek bir isim uzayında yapılmaz, birden çok isim uzayı da

kullanılabilir, birbirinden farklı isim uzayları saydam bir yoldan bir araya

getirilebilir.

18

Bağlanmış bir dosya sistemini göz önüne alacak olursak, yabancı bir isim uzayı

olarak belirtilen farklı bir isim uzayı’ndan bir dizin düğümü’nün tanımlayıcısını

depolayan bir dizin düğümü’nün belirtilmesinde bir bağlanmış dosya sistemi

sorumludur.

Dizin düğümü’nün depoladığı düğüm tanımlayıcı bir bağlama noktası

(mount

point) olarak isimlendirilir. Buna göre; yabancı isim uzayındaki dizin düğümü bir

bağlanma noktası

(mounting point) olarak isimlendirilir. Normal olarak,

bağlama noktası bir isim uzayının kökü’dür.

İsim çözümleme boyunca, bağlama noktasına bakılır ve onun dizin tablo’suna

erişim yoluyla çözümleme yapılmıştır. Bağlanma noktası gerekli tanımlama ve

erişim bilgilerini depolar, bir çok dağıtık dosya sisteminde ve Jade isimlendirme

sisteminde de bu yaklaşım takip edilmiştir.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

Bağlantılandırma ve Bağlama (Linking and Mounting):

Farklı makinelere dağıtılmış isim uzayları topluluğuna bakacak olursak; özellikle

her bir isim uzayı farklı bir sunucu tarafından gerçekleştirilmektedir ve her

birinin ayrı bir makinede çalışması mümkündür. Sonuçta, NS

2 gibi yabancı bir

19

isim uzayını NS

1 gibi bir isim uzayına bağlamak (mount etmek) istersek, NS

1 ‘in

bulunduğu makine eğer farklı bir yerde çalışan bir sunucu ise, NS

2 ile iletişimi

bir ağ üzerinden yapmak gerekebilir.

Bir yabancı isim uzayını dağıtık bir sisteme bağlamak (mount etmek) için en

azından şu bilgiler gerekmektedir:

1-) Bir erişim protokolünün ismi.

2

-) Sunucunun ismi.

3-) Yabancı isim uzayındaki bağlama noktasının ismi.

Burada her bir ismin çözülmesi gerekmekte olduğu unutulmamalıdır. Bir erişim

protokolünün isminin, yabancı isim uzayındaki sunucuyla iletişimde kullanılan

protokolün gerçekleştirimi için çözümlenmesi gerekmektedir. Sunucu’nun ismi,

erişilen sunucu nerede ise oranın adresi için çözümlenmesi gerekir.

(devam)

Đsimlendirme (Naming)

Varlıkların İsimlendirilmesi (Naming Entities)

Bağlantılandırma ve Bağlama (Linking and Mounting):

İsim çözümlemenin son parçası olarak, bağlanma noktası’nın ismi yabancı isim

uzayındaki bir düğüm tanımlayıcısı için çözümlenmesi gerekir. Dağıtık-olmayan

(nondistributed) sistemler de bunlara gerek yoktur.

20

Dağıtık dosya sistemlerine örnek olarak, Ağ Dosya Sistemi (Network File System,

NFS) bir protokolle beraber bir istemci’nin bir NFS dosya sunucusu’na (uzaktaki)

nasıl erişebileceğini tanımlar ve erişimi sağlar. NFS URL sayesinde NFS Internet

üzerinden tam yerleşim belirtilerek erişimi sağlayabilir. Örneğin,

“nfs://flits.cs.vu.nl/home/steen” gibi.

Burada nfs ismi, isim çözümlenmesinde dünya çapında çok iyi bilinen bir

isimdir. Başka bir yabancı (farklı) isim uzaylarını bir araya getirme yaklaşımı da

DEC ‘e ait Global İsim Servisi (Global Name Service, GNS) olarak bilinir. GNS ‘nin

potansiyel problemi eski kök (root) isimlerinin yeni yabancı isim uzayı ile

birleşmeden sonra elden geçirilmesi gerekliliğidir.

(devam)

Đsimlendirme (Naming)

Bir isim uzayının gerçekleştirimi

(The Implementation of a Name Space):

İsimlendirme servisini isim uzayı şekillendirir, bu servis kullanıcı ve süreçlere

ekleme, çıkarma ve isimlere bakma (lookup) için izin verir. Bir isimlendirme

servisi isim sunucuları tarafından gerçekleştirilir. Eğer bir dağıtık sistem bir yerel

21

alan ağı’na sınırlandırılmışsa, tek bir isim sunucusu ile isim servisi’nin

gerçekleştirimi sıklıkla mümkündür. Bununla beraber, büyük-ölçekli dağıtık

sistemlerde (bir çok varlığa sahip) bu sistem coğrafik geniş alana yayılmış olabilir,

isim uzayının bir çok isim sunucularından (onların üzerinden) gerçekleştirilmesi

gerekir.

İsim uzayı dağıtımı (Name Space Distribution) :

Büyük-ölçekli isim uzayları için, dünya çapında dağıtık sistem olması muhtemel,

organize olmuş (hiyerarşik olarak) olmaları genel bir durumdur. Böyle bir isim

uzayının sadece tek bir kök düğüme sahip olduğunu öncelikle varsayacak olursak,

böyle bir isim uzayının etkin gerçekleştirimi için bu uzayı mantıksal katmanlara

ayırmak gerekir.

(devam)

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

Bu katmanlar şöyle sayılabilir:

▫ Global katman (Global layer) :

Yüksek-seviyeli düğümlerden oluşur, kök düğümü ve diğer dizin düğümleri

mantıksal olarak kök’e yakındırlar, bunlar onun çocuklarıdır. Bu katmandaki

22

düğümler sıklıkla stabil olmaları karakteristiğine sahiptirler, yani dizin

tablo’larını gösterebilir, böylece isim uzayında isimleri depolanır.

▫ Yönetilen katman

(Administrational layer) :

Tek bir organizasyon içinde beraberce yönetilen dizin düğümleri ile

oluşturulmuştur. Dizin düğümlerinin karakteristik bir özelliği (bu katmanda

bulunan), aynı organizasyon veya yönetimsel birime ait varlık gruplarını

göstermesidir. Örneğin, bir organizasyondaki her bir departman (iş yerindeki

ofisler) için bir dizin düğümü olabilir veya bir dizin düğümü’nde tüm makineler

(hosts) bulunabilir. Başka bir dizin düğümünde tüm kullanıcıların

isimlendirilmesinde başlangıç noktası olabilir. Yönetimsel katmandaki

düğümler görece stabildir, bunun yanında genelde global katmandaki

düğümlerden daha sık değişikliklere sahiptirler (oluşurlar).

(devam)

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

Bu katmanlar şöyle sayılabilir:

▫ Yönetici katman (Managerial layer) :

Sıklıkla (düzenli olarak) değişen düğümlerden oluşur. Örneğin düğümler, bu

katmana ait yerel ağdaki makineleri (hosts) gösterir (temsil eder). Aynı

23

nedenden, katman; paylaşımlı dosyaları (kütüphane veya binary dosyaları)

gösteren düğümleri içerir. Düğümlerin diğer bir önemli sınıfı, kullanıcı-tanımlı

dizinleri ve dosyaları gösterenlerdir. Global ve yönetilen (administrational)

katmanın aksine, yönetici (managerial) katmandaki düğümler sadece sistem

yöneticisi tarafından bakımı (yönetimi) yapılmaz, dağıtık bir sistemin son

kullanıcıları (bireyler) tarafından yapılır.

(devam)

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

Internet üzerinden erişilen DNS isim uzayları, örtüşmeyen (üst üste) parçalara

bölünür, bunlara bölgeler

(zones) denilir. Bir “zone” isim uzayı’nın bir parçasıdır

ve ayrı bir isim sunucusu tarafından gerçekleştirilir.

24

Başarım ve elde edilebilirlik (availability) yönünden bakarsak her bir katmandaki

isim sunucularının farklı isterleri bulunur. Yüksek elde edilebilirlik özellikle global

katmandaki isim sunucuları için kritiktir. Eğer bir isim sunucusu çökerse, isim

uzayının büyük bir kısmı ulaşılamaz hale gelir, çünkü çökmüş sunucunun ötesinde

isim çözümlemesi gerçekleşemez.

Başarım bazen değişir. Global katmandaki düğümleri düşük değişiklik oranı

yüzünden bakma (lookup) işleminin sonuçları uzunca bir süre değişmeden kalır.

Sonuçta, bu sonuçlar etkin olarak önbelleklenir (yani yerel olarak depolanır). Bunu

istemciler yapar. Bir dahaki sefere bu bakma (lookup) işlemi gerçekleştiğinde,

sonuçlar istemcinin önbelleğinden alınır, isim sunucusunun sonuçları

döndürmesine gerek kalmaz. Bunun bir sonucu olarak, isim sunucuları (global

katmandaki) tek bir bakma (lookup) isteği için çabuk bir cevaba sahip olmazlar.

Diğer yandan, işlem hacmi (throughput) önemli olabilir. Özellikle büyük

sistemlerde milyonlarca kullanıcı söz konusuyken önemli olabilir.

(devam)

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

Global katmandaki isim sunucularının başarım ve elde edilebilirlik (availability)

isterleri kopya (replicating) sunucular ile karşılanabilir, buna istemci taraflı

önbellekleme kombinasyonu da eklenebilir. Bu katmanda etkiler hemen ortaya

çıkmaz (tepkiler) o yüzden kopya’ların tutarlılığının sağlanması daha kolaydır.

25

Yönetilen katmandaki isim sunucuları için elde edilebilirlik (availability) isim

sunucusu ile aynı organizasyondaki istemciler için ilk olarak önemlidir. Eğer isim

sunucusu çökerse, organizasyon içindeki bir çok kaynak ulaşılamaz hale gelecektir,

çünkü bunlar bakma (lookup) işlemine tabi tutulmamaktadırlar. Diğer yandan,

organizasyon dışından kullanıcılar için bu kaynakların geçici olarak ulaşılamaz

olması daha az önemli olabilir (Bu katman global’e göre daha yerel kalmaktadır).

Başarım açısından, yönetilen (administrational) katmandaki isim sunucuları global

katmandakilere benzer karakteristiğe sahiptir. Çünkü düğümler de değişiklik

sıklıkla olmaz ve bakma (lookup) sonuçlarının önbelleklenmesi yüksek seviyede

etkindir, başarım daha az kritik hale gelir. Bununla beraber, global katmanın

aksine, yönetilen (administrational) katman bakma (lookup) sonuçlarıın birkaç

milisaniye içinde döndürülmesine dikkat etmek zorundadır, bu sonuçlar

sunucudan doğrudan veya istemcinin önbelleğinden alınır.

(devam)

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

Global katmana göre güncellemeler (updates) daha hızlı işlenir. Örneğin yeni bir

kullanıcının hesabının oluşturulmasının saatlerce süremsi kabul edilemez bir

durumdur.

26

Bu isterler yüksek başarımlı makineler ile karşılanır, ayrıca istemci-taraflı

önbellekleme toplam elde edilebilirliği (overall availability) artırmak için

kullanılır.

Yönetici (Managerial) katmandaki isim sunucularının elde edilebilirlik isterleri

genel olarak daha az öğeye dayanır. Geçici elde edilemezlik (unavailability)

riskinde isim sunucusunu çalıştıran tek bir makine (adanmış = dedicated)

kullanımı yeterli olmaktadır. Bununla beraber, başarım acımasızdır, kullanıcılar

işlemlerinin hemen gerçekleşmesini isterler, çünkü güncellemeler düzenli ve sık

olur istemci-taraflı önbellekleme sıklıkla daha az etkindir (özel ölçütler

alınmaksızın).

Şekil 4.7 ‘de isim uzayları arasında bir karşılaştırma görülmektedir.

(devam)

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

27

Ö

ğe Global Yönetilen (Administrational) Yönetici (Managerial)

Fig. 4.7.

A

ğ’ın co

ğrafik ölçüsü (Geographical

scale of network)

Dünya

çapında Organizasyon Departman

Dü

ğümlerin toplam sayısı (Total

number of nodes) Az Bir çok Çok sayıda

Bakma i

şlemlerine kar

şı verilen cevap

(Responsiveness to lookups) Saniyeler Milisaniyeler Anında

Güncelleme yayılımı (Update

propagation) Tembel Anında Anında

Şekil 4.7 ‘de, büyük-ölçekli bir isim uzayındaki düğümlerin gerçekleştirimi için

isim uzayları arasında bir karşılaştırma, burada isim uzayı global, yönetilen

(administrational) ve yönetici (managerial) katman olarak belirtilmiştir.

propagation)

Kopya sayısı (Number of replicas) Bir çok Hiç veya az Hiç

İstemci-taraflı önbellekleme

uygulanıyor mu? Evet Evet Bazen

Đsimlendirme (Naming)

İsim uzayı dağıtımı (Name Space Distribution) :

Önemli:

Dağıtık sistemlerde global ve yönetilen katmanların gerçekleştirimi zordur.

Kopyalama ve önbellekleme’de başarım ve elde edilebilirlik isterleri tutarlılık

(consistency) sorunlarına yol açar Bu problemlerin bazıları kopyaların ve

28

önbelleklerin alan ağı çapında yayılması ile aşılabilir, bu sefer de uzun iletişim

gecikmeleri gidermek için senkronizasyon gerekliliği daha zor olmaktadır.

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

İsim çözümlemesine, isim uzayının bir çok makine arasında dağıtımı etki eder.

İsim çözümlemesinin gerçekleştiriminde büyük ölçekli isim servisleri için isim

sunucularının kopya (replika) edilmediği ve istemci

-taraflı önbellek

29

kullanılmadığını varsayarsak, her bir istemci yerel bir isim çözümleyici’ye (name

resolver) erişerek, isim çözümleme işleminin başarıldığından emin olur. İsim

çözümlemenin iki farklı yolu bulunur:

1-) Adımsal isim çözümleme (Iterative name resolution):

Bir isim çözümleyici, kök isim sunucusu için tam ismi ele alır. Adresin kök

sunucunun bağlı olduğu (iyi bilinen) yer olarak varsayılması sayesinde, kök

sunucu olabildiğince uzağa kadar ki yol ismini çözebilir ve sonucu istemciye geri

döndürür. Yani yol ismindeki her bir etiketi tek tek deneyerek sonuç istemciye geri

döndürülür. İlk etiket çözülünce, geri kalan etiketlere bakılır her adımda çözme

işlemi başarıldıktan sonraki etikete geçilir. Bunun için istemcinin isim

çözümleyicisi bir sonraki isim sunucusuna bağlanır (contact).

(devam)

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

30

Fig. 4.8.

Şekil 4.8 ‘de, Adımsal isim çözümlenmesi prensibi gösterilmektedir.

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

2-) Özyineli isim çözümleme (Recursive name resolution):

Bir isim sunucusu bulduğu sonuçları bulduğu bir sonraki isim sunucusuna geçirir

31

(gönderir). Örneğin kök isim sunucusu isim sunucusunun adresini (Label\_1 isimli

düğümü gerçekleştiren isim sunucusunun adresi) bulurken, yol ismini çözmesi

gerekir ve bunu tekrarlı (recursive) bir şekilde her bir isim sunucusu arasında

yapılması ile elde eder. Bu bir sonraki sunucu tüm yolu çözerek “index.txt” isimli

dosyayı kök sunucuya geri döndürür, daha sonra bu dosya istemci’nin isim

çözümleyicisine gönderilir. Özyineli isim çözümün’de de son adımda (örneğin ftp

isteği için) ilgili sunucu ile iletişime geçilir ve genelde istemci tarafından ayrı bir

süreç ile bu iş yapılır.

(devam)

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

32

Fig. 4.9.

Şekil 4.9 ‘da, Özyinelemeli isim çözümlenmesi prensibi görülmektedir.

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

Özyinelemeli (recursive) isim çözümlemenin ana eksiği her bir isim sunucusundan

yüksek başarım beklentisidir. Temelde bir isim sunucusu bir yol isminin tam

çözümünü idare etmek ister, bunun yanında değer isim sunucuları ile ortak

33

çalışmada yapılabilir. Global katmandaki isim sunucuları sadece adımsal isim

çözümlemeyi desteklerler.

Özyinelemeli isim çözümlemenin iki önemli avantajı vardır. İlk avantaj, adımsal

isim çözme ile karşılaştırıldığında, sonuçları önbellekleme daha etkin olmaktadır.

İkinci avantaj ise, iletişim maliyetleri düşmektedir. Özyineli isim çözme, her bir

isim sunucusunun düşük-seviye düğümlerin gerçekleştirimi için sorumlu her bir

isim sunucusunun adresini öğrenmesine (seviyeli) izin verir.

Bunun bir sonucu olarak, başarım artırmada önbellekleme önem kazanır. Global

ve yönetilen (administrational) katmanlarda daha az değişiklik olduğundan, kök

isim sunucusu geri dönen adresi etkin olarak önbellekler. Hemen yapılan bakma

(lookup) işlemlerinin sonuçları da benzer olarak daha etkin olur. Bu ikinci avantaj

(özyineli isim çözmenin ikinci avantajı) daha ucuz olmasını iletişim maliyetinin

azlığına borçludur.

(devam)

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

Adımsal isim çözme’de önbellekleme istemcinin isim çözücüsünün gerekli olduğu

şekilde sınırlamasında kalır. Bir çok organizasyonda (yerel olanlar) ara bir isim

sunucusu adresleri tüm istemcilerle paylaşılır.

34

Bu yerel isim sunucusu gelen tüm isimlendirme isteklerini alır/idare eder ve

sonuçları önbellekler. Böyle bir ara sunucu yönetimsel açıdan gerekli/kullanışlı

olur. Sadece kök isim sunucusunun yerini bilmek isteyen sunucuların varlığında,

diğer makinelerin bu bilgiye ihtiyacı yoktur.

Özyinelemeli isimlendirmede ilk etiketteki isim sunucusu ile istemci arasındaki

iletişim (mesaj alış-verişi), iletişim maliyetini belirler. Yani adımsal isim çözme’de

bir çok kez sunucu (sunucular) ile istemci arası iletişim olurken, özyinelemeli de

tek noktada bu olur.

(devam)

Đsimlendirme (Naming)

İsim çözümlenmesi gerçekleştirimi

(Implementation of Name Resolution) :

35

Fig. 4.11

Şekil 4.11 ‘de, Özyinelemeli ve adımsal isim çözümlemenin iletişim maliyeti

açısından karşılaştırılması gösterilmiştir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

36

Ders’in sonu…

37

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 11

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 11: Hareketli varlıkların yerleştirilmesi

(Locating Mobile Entities),

Referanssız varlıkların yok edilmesi

(Removing Unreferenced Entities).

Hareketli varlıkların yerleştirilmesi

(Locating Mobile Entities)

• İsimlendirme servisleri genelde sabit yerleşimli varlıklar için kullanılmaktadır.

Geleneksel isimlendirme sistemleri isimden-adrese bir haritalandırmanın düzenli

olarak değişimini (yani mobil varlıkları) desteklemek için uygun değillerdir.

Varlıkları yerleştirmeye karşılık isimlendirme

3

(Naming versus Locating Entities) :

Üç farklı tipte isim bulunur, bunları ayırt edersek; insan-dostu isimler,

tanımlayıcılar, adresler olarak sayabiliriz. Dağıtık sistemler insanlar tarafından

kullanıldığı için erişebilecek bir varlık adresine sahip olmalı ve sanal olarak tüm

isimlendirme sistemleri bir insan-dostu isimden-adrese haritalamaya sahip

olmalı/bakımını yapmalıdırlar.

DNS gibi büyük

-ölçekli isim uzayları etkin olarak üç katmana (Global, yönetilen

ve yönetici) bölümlenir. Global ve yönetilen katmanında isimler çok sık değişmez

(seyrek değişir), fakat yönetici (managerial) katmanda çok sık değişebilir.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Varlıkları yerleştirmeye karşılık isimlendirme

(Naming versus Locating Entities) :

Örneğin bir sunucu servisinin başka bir makineye taşındığını varsayalım, hatta

yeni taşındığı makine farklı bir alan’da (domain) olsun. Büyük olasılıkla ismi

değişmeyecektir (taşınsa da değişmeyecektir, örneğin ftp.alan.com taşınıp

ftp.alan

2.com makinesine de konsa ismi yanı kalacaktır). Çünkü uygulamalar ve

4

ftp.alan

2.com makinesine de konsa ismi yanı kalacaktır). Çünkü uygulamalar ve

kullanıcıların, onunla ilişkilendirilmiş sembolik bağlantıları

(symbolic links)

mevcuttur. Burada isim bir tanımlayıcı olarak kullanılmıştır. Fakat bağlantılar

(ona bağlantı yapacak olanlar için) geçersiz hale gelecektir. Bu soruna iki temel

çözüm bulunmaktadır.

Birinci çözüm, yeni makinenin adresini DNS veritabanına ftp.alan.com için

kaydetmektir. Alternatif ikinci çözüm ise adres yerine yeni makinenin

(ftp.alan ftp.alan

2.com) ismini kaydetmektir, etkin olarak bu ftp.alan.com ) ismini kaydetmektir, etkin olarak bu ftp.alan.com bir sembolik bir sembolik

bağlantıya dönüşür. İki çözümünde ciddi eksiklikleri mevcuttur. Yeni makinenin

adresini kaydettiğimizi düşünelim, açıkça bakma (lookup) işlemleri bu

yaklaşımda etki etmez. Bununla beraber, ftp.alan.com bu sefer buradan başka

bir makineye tekrar taşınırsa, DNS veritabanındaki girdisi (alan.com) oldukça iyi

güncellenecektir. Güncelleme yerel bir işlemden uzun sürmez, yüz milisaniyeler

seviyesinde gerçekleşir. Bu yaklaşım, diğer bir deyişle yönetici katmandaki

düğümler üzerinde yapılan işlemlerin etkin olduğu varsayımını ihlal eder.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Varlıkları yerleştirmeye karşılık isimlendirme

(Naming versus Locating Entities) :

Sembolik bağlantı kullanımının ana eksiği, bakma (lookup) işleminin daha az

etkin hale gelmesidir. Her bir bakma (lookup) iki adıma bölünür:

1

-) Yeni makinenin isminin bulunması,

5

2-) Bu isimle ilişkilendirilmiş adrese bakılması (lookup).

Bununla beraber, ftp.alan.com tekrar taşınıp ftp.alan

3.com olunca, yerel bir

işlemle ftp.alan

2.com ismi sembolik bağlantı ile ftp.alan

3.com ismine

güncellenerek çevrilir; DNS veritabanında (alan.com için) girdi olduğu gibi

bırakılır. Bunun eksiği bakma (lookup) işlemine bir adım daha eklenmesidir.

Yüksek seviyeli mobil varlıklar için durum kötüleşir; bir varlık her seferinde

taşındıkça, yerel olmayan güncelleme işlemleri yapılması ve bakma (lookup)

işlemlerine bir adım daha eklenmesi gerekmektedir. Bir diğer ciddi problem ise

ftp.alan.com ‘un değişime izin vermemesidir. Sonuçta, ömür boyu değişmez

isimlerin seçilmesi’nin (varlığı temsil eden isim) beklenmesi isim seçiminde

önemli bir noktadır.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Varlıkları yerleştirmeye karşılık isimlendirme

(Naming versus Locating Entities) :

Bununla beraber, aynı isim başka bir varlık için kullanılmamalıdır. Pratikte, böyle

isimlerin seçimi çok uzun ömürlü varlıklar için özellikle zor (ve WWW ‘de

isimlendirmede gösterilmesi zor) olur. Özellikle, bir çok varlık farklı isimler ile

bilinir ve tüm bunlar geçerli kalmak zorundadır, daima aynı varlığı işaret etmeli

6

ve hareketlilik (mobility) ile uyumlu olmalıdırlar.

Bu nedenlerden, geleneksel isimlendirme servisleri (DNS gibi) mobil varlıklar için

uygun değildir, farklı çözümler gerekir. Çünkü geleneksel isimlendirme servisleri

insan-dostu isimler ve adresler arasında doğrudan bir haritalama (eşleştirme)

yaparlar. Bu yüzden bir isim veya bir adresin her sefer değişmesinde

haritalama’nında değişmesi gerekir.

Varlıkların yerleştirimi’nin (tanımlayıcılar kullanılarak) isimlendirmeden

ayrılmasına iyi bir çözüm Şekil 4.17 (b) ‘de görülmektedir. Fakat tanımlayıcılar

insan-dostu sunumu desteklemez, onlar sadece makineler tarafından işlenmek

için optimize edilmişlerdir. Tanımlayıcılar ihtiyaç olunduğu sürece yerel olarak

depolanırlar, çünkü asla farklı bir varlığı işaret etmeyecekleri veya

değişmeyecekleri bilinir.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Varlıkları yerleştirmeye karşılık isimlendirme

(Naming versus Locating Entities) :

Fig. 4.17

7

Şekil 4.17 ‘de, a) Doğrudan, tek-seviyeli isim ve adresler arası haritalama.

b) İki-seviyeli, tanımlayıcıları kullanan haritalama.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Varlıkları yerleştirmeye karşılık isimlendirme

(Naming versus Locating Entities) :

Sonuçta, bir tanımlayıcı tekrar istenildiğinde isim servisinde bakma (lookup)

yapılmadan yerel olarak elde edilebilir (retrieve). Bir varlığın yerleşimi ayrı bir

yerleşim servisi

(location service) tarafından idare edilir. Bu servis

tanımlayıcıyı girdi olarak alır, tanımlanmış varlığın adresini geri döndürür.

8

Birden çok kopyası varsa (varlığın) birden çok adres geri döndürülür.

Basit çözümler (Simple solutions) :

Yerel alan dışında da varlıkların yerleşimi söz konusudur. Etkin broadcast

işlemlerinin (yayınlama) yapıldığı ağ’a sahip bir dağıtık sistem göz önüne alacak

olursak, böyle bir ortam (environment) ‘da bir varlığı yerleştirmek basittir. Buna

örnek olarak, yerel-olan kablosuz ağı örnek alınabilir. Burada bir mesaj varlığın

tanımlayıcısını içerir ve bu mesaj her makineye broadcast edilir, sonuçta her

makine bu varlığın sahibi olup olmadığını kontrol etmek için bu mesaja istekte

bulunur. Sadece bu varlığın bir erişim noktasına sahip makine geriye cevap

gönderir, bu cevapta bu erişim noktasının adresi bulunur. Bu prensip Internet

Adres Çözümleme Protokolü’nün (Address Resolution Protocol, ARP) sadece IP

adresi verilen makine için veri-bağı (data-link) adresini bulmasında kullanılır.

(Yani IP

MAC ID). Bunun tam terside Reverse ARP olur. (Yani MAC ID

IP).

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Basit çözümler (Simple solutions) :

Ağ büyüdükçe broadcast yapmak etkin olmamaya (verimsiz) başlar. Multicasting

ile sadece özel (gerekli) gruplara noktadan-noktaya ağlarda varlık yerleşim tespiti

yapılabilr. Internet ağ-seviyesi multicasting grupları desteklemektedir.

Bir multicast adresi bir den çok varlık için genel bir yerleşim servisi olarak

9

kullanılabilir. Her bir çalışanın bir mobil bilgisayarının olduğu şirket örneğinde,

her bir yerel ağ’a bağlanınca dinamik IP adresleri alırlar ve özel bir multicast

grubuna dahil olurlar. A makinesi ile B makinesi arası iletişim bu grup içerisinde

olur. Bir multicast adresi kullanmanın bir diğer yolu da onu bir kopyalanmış

(replika edilmiş) varlıkla ilişkilendirmektir. Böylece en yakın kopyayı (replika)

yerleştirirken (yerini belirlerken) multicasting kullanılmış olur.

İşaretçilerin iletilmesi (Forwarding pointers) :

Mobil varlıkların yerleştirilmesinde (yerinin belirlenmesi) için popüler bir

yaklaşım da, işaretçilerin iletilmesindir (forwarding pointers). Prensip basittir: bir

varlık A ‘dan B’ye taşınırsa, yeni B yerleşimi için arkasında bir referans bırakır.

Bunun ana avantajı basitliğidir, oldukça kısa sürede varlık yerleştirilir. Geleneksel

isimlendirme servislerinde örneğin bir istemci şu anki adresi işaretçilerin iletimi

ile oluşan zinciri takip ederek bakma (lookup) ile bulabilir.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Basit çözümler (Simple solutions) :

İşaretçilerin iletilmesi (Forwarding pointers) :

Bunun önemli eksikleri bulunur. İlk önce, zincir uzadıkça bir varlığı yerleştirmek

oldukça maliyetli olmaya başlar, çünkü özel bir ölçüt bulunmamaktadır. İkinci

olarak zincirdeki tüm orta (intermediate) yerleşimler ihtiyaç duyulduğu sürece

10

iletilen işaretçilerin zincirinin bir parçcası olarak bulundurulur/bakımı yapılır.

Üçüncü olarak kırık bağlantılar (broken links) için yara alabilirlik (vulnerability)

mevcuttur.

Stub kelimesi koçan anlamına gelmektedir. Proxy kelimesi vekil anlamına gelir.

Böylece iletilen işaretçilerin kaybolması durumunda, varlık artık erişemez duruma

gelir. İletilen işaretçilerin çalışma mantığını anlamak için dağıtık nesnelere

bakmak gerekir. SSP zinciri ((stub, scion ) skeleton) pairs anlamındadır)

yaklaşımında her bir iletilen işaretçi bir (proxy, skeleton) çifti olarak bulunur.

[SSP ‘de bir proxy, bir stub ve bir skeleton, bir scion olarak adlandırılır ve (stub,

scion) çifti SSP ‘yi açıklar]. Bir skeleton (yani sunucu-taraflı koçan) geçerli

nesneye yerel bir referansı içerir veya yerel referansı bir proxy için (yani istemcitaraflı koçan) (bir nesne için) içerir. Uzak referanslar için girdi elemanları olarak

davranan skeleton’lar ve çıkış elemanları olarak proxy’ler bulunur.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Basit çözümler (Simple solutions) :

İşaretçilerin iletilmesi (Forwarding pointers) :

Fig. 4.18

11

Şekil 4.18 ’de, (proxy, skeleton) çifti kullanarak işaretçi iletimi prensibi.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Basit çözümler (Simple solutions) :

İşaretçilerin iletilmesi (Forwarding pointers) :

Bir nesne A adres uzayından B adres uzayına taşınırsa, arkasında bir proxy bırakır

(A ‘daki yerleşimde) ve B ‘deki yerine (işaret edilen yere) bir iskelet (skeleton)

kurar. Burada yaşanılan göç tamamen istemciye saydamdır. İstemcinin nesne ile

12

ilgili gördüğü tek şey proxy’dir.

Invokasyon (çağrı) nereden yapılıyorsa oradaki proxy’nin tanımlaması yapılır, bu

invokasyon yeri (nereden olduğu) istemciden gizlidir. Proxy tanımı, istemcinin

transport-seviyesi adresini içerir, yerel olarak yaratılmış bir numara ile proxy

tanımlanır. Bir invokasyon, şu anki yerleşimdeki bir nesneye eriştiğinde bir cevap

invokasyonun başlatıldığı yerdeki proxy’e geri gönderilir.

Cevabın sonuna (piggybacked) şu an ki yerleşim eklenir ve proxy, nesnenin şu

anki yerleşimdeki biri olan ilgili iskeletine (skeleton) göre ayarlanır. Eğer bir

iskelet (skeleton) daha fazla süre herhangi bir proxy tarafından işaret edilmiyorsa,

yok edilebilir (silinebilir). Bir (proxy, skeleton) zincirindeki bir süreç çökerse veya

ulaşılamaz olursa problem ortaya çıkar.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Basit çözümler (Simple solutions) :

İşaretçilerin iletilmesi (Forwarding pointers) :

Fig. 4.19.

13

Şekil 4.19 ‘da, bir proxy ’deki bir kısayol’un depolanması ile iletilen bir işaretçinin

tekrar yönlendirilmesi.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Ev-tabanlı Yaklaşımlar (Home-based Approaches) :

Büyük-ölçekli ağlarda mobil varlıkların desteklenmesinde bir ev yerleşimi

(home location) yaklaşım bulunmaktadır, bu yerleşim bir varlığın şu anki

yerleşiminin takip edilebilmesini sağlar. Ağ veya süreç hatalarına karşı özel

teknikler ile koruma sağlanılabilir. Pratikte ev yerleşimi varlık nerede yaratıldıysa

14

orasının seçilmesidir. Ev-tabanlı yaklaşımlarda bir geri dönüşlü

(fall-back)

mekanizma iletilen işaretçiler tabanlı yerleşim servisleri için bulunur. Örneğin

mobil makineler sabit birer IP adresi kullanırlar, bu IP adrese yapılacak tüm

iletişimler ilk olarak mobil makinenin (host) ev etmeni ‘ne (home agent)

yönlendirilir.

Bu ev etmeni mobil IP ‘nin de içinde bulunduğu yerel alan ağında bulunur. Bir

mobil makine başka bir ağ’a taşındığında güvenilir-adres

(care-of-address)

isimli geçici olarak erişilebilecek (iletişim için) ev etmeninde kayıtlı bir adres

bulunur. Bir mobil makine için bir paketi ev etmeni aldığında, eğer makine şu

anki ağ içindeyse (alıcı makinenin yerleşimine bakma (lookup) işlemi ile bakılır)

basitçe ev etmeni paketi iletir, eğer ağ içinde değilse veri IP paketi haline getirilip

makinenin şu anki bulunduğu (yeni) yerleşimine tünelleme ile (care-of-address

‘ine) gönderilir.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Ev-tabanlı Yaklaşımlar (Home-based Approaches) :

Aynı zamanda göndericiye alıcının şu anki (yeni) yerleşim de bildirilir. IP adres

mobil makineler için tanımlayıcı olarak kullanılır; bu arada bu yaklaşımın eksiği,

mobil bir varlıkla iletişimde ilk adım olarak ev ile iletişime geçilmesi gerekliliğinin

büyük

-ölçekli ağlarda varlığın tamamen farklı yerde olmasından dolayı iletişim

15

gecikmelerine yol açmasıdır.

Mobil telefonlarda uygulanan bir çözüm ise iki-taraflı şema (two-tiered scheme)

kullanmasıdır. Bir mobil varlık için bir bağlantı kurulduğunda, bir istemci ilk önce

yerel bir kaydı kontrol eder, böylece mobil varlığın yerel olarak elde edilebilir olup

olmadığını kontrol eder; eğer değilse varlığın ev yerleşimine bağlanılarak şu anki

yerleşimi bulunabilir.

Ev tabanlı yaklaşımın bir diğer önemli eksiği sabit ev yerleşimi kullanımıdır, yani

ev yerleşiminin daima var olduğu varsayılır, fakat bazen bu şekilde varlığa erişim

imkansız hale gelebilir. Buna çözüm olarak ev yerleşimi geleneksel isimlendirme

servisine kayıt edilerek, istemcinin ilk önce buraya bakması (lookup) sağlanılabilir

(kayda bakıp ev yerleşimi canlı mı değil mi anlaşılabilir).

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

İki-taraflı ev-tabanlı yaklaşım, varlıkları yerleştirmek için çoklu katmanlara

genelleştirilebilir. Global yerleşim servisi tabanlı yaklaşımlar mevcuttur. Bunlar

genel-amaçlı yerleşim servisi olarak, bir çok hiyerarşik (sıradüzensel) yerleşim

servisinin sunumu şeklinde hazırlanmışlartır ve Kişisel İletişim Sistemleri

16

(Personal Communication Systems) olarak isimlendirilirler.

Genel mekanizma:

Hiyerarşik şema’da, bir ağ alanlar (domain’ler) topluluğuna bölünmüştür, bu

çokça benzer bir şekilde DNS hiyerarşik organizasyonuna benzemektedir. En

üstte tek bir alan (domain) tüm ağı kaplamaktadır (spans kelimesi bölmek veya

kapsama anlamına gelmektedir). Her bir alan daha küçük birden fazla alt

alan’lara (subdomain) bölünmüştür. Bir alt-seviye alan (domain), yaprak (kol)

alan (leaf-domain) olarak adlandırılır. Bunlar genel de yerel alan ağları

(bilgisayar) veya mobil telefon ağları hücreleri olmaktadırlar. DNS’ye benzer

olarak (ve diğer hiyerarşik sistemlere), her bir D alanı (domain’i) ilgili bir

“ dir(D) “ dizin düğümü’ne (directory node) sahiptir, böylece bu domaindeki

varlıkları takip edebiliri. Dizin node eğer yüksek-seviyede bir domain’e aitse buna

kök (dizin) düğümü (root (directory) node) denilmektedir. Bu düğüm tüm

varlıklar hakkında bilgiye sahiptir.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Fig. 4.21

17

Şekil 4.21 ‘de, Alanlar içindeki yerel bir servisin hiyerarşik organizasyonu, her

birinin ilişkili bir dizin düğümü bulunduğu gösterilmektedir.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Bir D alanında (domain’inde) şu an yerleşik bulunan her bir varlık dizin düğümü

olan “ dir(D) ” deki bir yerleşim kaydı (location record) tarafından temsil edilir.

Her bir üst

-seviye alan ‘ın dizin düğümü alt

-seviye alan’daki varlıkların kaydını

18

Her bir üst

-seviye alan ‘ın dizin düğümü alt

-seviye alan’daki varlıkların kaydını

içerir. Bunu sadece ilgili dizin düğüm’lerinin işaretçilerini içererek yaparlar

(işaretçi iletimi).

Sonuçta, kök düğümü her bir varlık için bir yerleşim kaydı’na (kayıt girdisi) sahip

olur, burada her bir yerleşim kaydı’nın bir sonraki alt-seviye alt alanı’nın

(subdomain) dizin düğümü için bir işaretçi (pointer) depolayarak (burada şu an

yerleşik olan ilişkili varlığın kaydının bulunduğu alt alan (subdomain)

kastedilmektedir) tutulduğu bilinmelidir.

Bir varlık birden fazla adrese sahip bulunabilir, örneğin birebir kopya (replika)

edilmiş olabilir. Hiyerarşik yapıda bakma (lookup) işlemi, bir istemcinin E

varlığını yerleştirmek (yerini bulmak) istemesiyle ilgili olarak, istemcinin

bulunduğu D alanındaki (yaprak alan) dizin düğümü’ne bir bakma (lookup) isteği

göndermesi ile başlar (alttan yukarı doğru).

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Eğer dizin düğümünde E varlığı’nın kaydı yoksa bu varlık bu alan’da yoktur ve

istek dizin düğmü tarafından bir üstüne (parent’ine) yani ebeveynine gönderilir.

Ebeveyn alan (parent domain), çocuk alan’dan (child domain), (yani ilk isteğin

19

yapıldığı bu örnekteki istemcinin içinde bulunduğu alan’dan bahsediliyor), daha

büyük bir alan’dır.

Eğer burada da kayda rastlanmaz ise bir üste daha gönderilir. İstek, kaydın (M

dizin düğümünde) bulunduğu yere varınca M alan’ında bu E varlığının var olduğu

anlaşılır. Buradaki “ dir(M) ” E varlığının hangi alt alan’ında olduğunu işaretçi

içeriği ile kaydetmiştir ve istek bu alt alan’ın dizin düğümüne gönderilir. Böylece

E varlığının şu an bulunduğu en uç noktaya kadar işaretçi iletimi ile istek

iletilerek E varlığının bulunduğu yere varılır.

Böylece tüm yol’dan sonra varılan adres (bakma (lookup neticesinde) istemciye

geri döndürülür. (Bu işlemde (geri dönüş) hiyerarşik olarak ilerler yani alttan üste

sonra üstten alta doğru olur).

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Hiyerarşik yerleşim servislerinde en önemli gözlem, bakma (lookup) işleminin

yerelliği ortaya çıkartmasıdır. Prensipte, varlık istenilen istemcinin merkezinde

olduğu bir çemberin (dereceli olarak büyüyen) içinde aranmaktadır. Arama alanı,

20

her seferinde bir üst seviyedeki dizin düğümü’ne bakma (lookup) isteğinin

iletilmesi ile genişlemektedir. En kötü durumda, istek kök düğüm’e ulaşıncaya

kadar arama devam eder. Çünkü kök düğüm her bir varlık için bir yerleşim

kaydı’na (location record) sahiptir ve istek basitçe yaprak (leaf) düğüm’lerden

birine aşağıya doğru işaretçilerin oluşturduğu yol sayesinde iletilebilir.

Güncelleme işlemleri, ilgili D alanındaki yaprak düğüm “ dir(D) “ de araya

yerleştirme (insertion) işlemi olarak başlatılır ve araya yerleştirme isteği ebeveyn

(parent) düğüme iletilir. Bu şekilde alan’ın dizin düğümü M ‘e iletilir. M zaten E

varlığının yerleşim bilgisine (işaretçisi sayesinde) sahiptir. E varlığının ilgili bir

sonraki alt-seviye düğümü ile ilgili bilgi isteğine karşılık aşağı doğru ayarlamalar

yapılarak yaprak düğüm ilgili yaprak alan ile ilişkili varlığın adresi için bir kayıt

tutar (oluşturur).

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Bir adresin araya yerleştirilmesi (insert) için bir yukarıdan-aşağıya

(top-down)

bakış açısıyla bir işaretçi zinciri oluşturulmasını (kurulmasını, install) ve bunun E

varlığı için bir yerleşim kaydı içeren düşük

-seviyeli dizin düğümü’nden başlayan

21

bir yapıda olmasını gerektirir. Ebeveyn düğüme (parent node) aray ekleme

isteğinin geçirilmesinden (gönderilmesinden) önce bir yerleşim kaydının

oluşturulması da bir alternatiftir. Diğer bir deyişle, işaretçi zinciri

(chain of

pointers) alttan-yukarı

(bottom-up) doğru oluşturulur.

Bunun avantajı, bir adresin bakma (lookup) işlemi için olabildiğince kısa sürede

hazır hale gelmesidir. Sonuçta, eğer bir ebeveyn düğüm geçici olarak erişilemez

durumdaysa, adres alan içinde geçerli (şu anki) düğüm tarafından temsil edildiği

için bakma (lookup) işlemi hala devam edebilir.

Bir silme işlemi, araya yerleştirme (insert) işlemine benzer ilgili düğümün ilgili

dizin düğüm’deki yerleşim kaydı silinir buna göre işaretçiler güncellenir. Eğer bir

E varlığı için bir yerleşim kaydı ebeveyn düğüm’de boş (empty) hale gelirse, bu

kayıt silinerek bir silme süreci, hala boş olmayan bir yerleşim kaydı kalıncaya

kadar işaretçi silinmesi ile devam eder veya kök düğüm’e erişilinceye kadar

devam eder. (devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Fig. 4.22

22

Şekil 4.22 ‘de, farklı yaprak alanlarda iki adrese sahip bir varlığın bilgisinin

depolanmasına ait örnek gösterilmektedir.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma:

Fig. 4.23

23

Şekil 4.23 ’de, hiyerarşik olarak organize olmuş bir yerleşim servisindeki bir

yerleşime bakılması (lookup) işlemi gösterilmektedir.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Genel mekanizma: Fig. 4.24

24

Şekil 4.24 ‘de gösterilenler:

a) E varlığı hakkında bilgi sahibi olan ilk düğüme bir araya yerleştirme isteğinin

iletilmesi.

b) Oluşturulan yaprak düğüm için iletilen işaretçilerin bir zinciri.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

İşaretçi önbellekleri:

Önbellekleme, eğer önbelleklenen veri nadiren değişikliğe uğruyorsa etkin olur.

Örneğin mobil bir E varlığı için etkinlik oldukça düşüktür. Genelde, eğer D; E

mobil varlığının düzenli olarak hareket ettiği bir küçük alan’da ise, E ‘nin

25

“ dir(D) ” deki şu anki yerleşimi çin bir bakma (lookup) işlemini başka herhangi

bir düğüm için yapmak yerine sadece E için yapılabilir.

Bu yaklaşıma işaretçi önbellekleme

(pointer caching) denilir. Burada

karşılaşılan bir sorun ise bir önbellek girdisi’nin (cache entry) geçersiz

(invalidate) olduğu durumdur.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

İşaretçi önbellekleri:

Fig. 4.25

26

Şekil 4.25 ‘te, bir varlığın zamanını oldukça fazla geçirdiği bir alt-seviye alanına ait

dizin düğümü’ne olan bir referansı önbellekleme gösterilmektedir.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

İşaretçi önbellekleri:

Fig. 4.26

27

Şekil 4.26 ‘da, bir adres mevcut iken yerel olmayan bir adres geri döndürdüğü için

geçersiz olması gereken bir önbellek girdisini gösteren şematik verilmektedir.

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Ölçeklenebilirlik konuları:

Hiyerarşik yerleşim servisleri için ana problemlerden biri kök düğümün her bir

varlık için bir yerleşim kaydını depolaması ve her bir varlık için istekleri

işlemesidir. Depolamanın kendisi büyük bir problem değildir. Her bir kayıt

28

oldukça küçüktür. Gerçek problem, kök’ün hiçbir özel ölçüt olmadan bir çok

bakma (lookup) işlemi ve güncelleme (update) isteğiyle karşılaşılır ve bu bir

darboğaz (bottleneck) oluşumuna neden olur. Buna çözüm, kök düğüm ve diğer

yüksek-seviyeli dizin düğümleri alt düğümleri (subnode’lara) bölmektir. Her bir

alt düğüm (subnode) kendi ilgili yerleşim servislerini varlıklar için destekler.

Bunun yanında, bu alt düğüme ayrıştırma da yeterli olmamaktadır. Fazla sayıda

alt düğüm’ün fiziksel olarak (örneğin 100 tane alt düğüm) nerede yerleştiği ve

yerleşim servisleri tarafından kapsandığı en önemli sorudur. Bir ihtimal, alt

düğümleri bir merkezi yaklaşımla yerleştirmektir. Böylece alt düğümler

birbirlerine daha yakın hale gelirler. Örneğin bir küme (cluster) içerisinde

bulunurlar. Etkin olarak bir paralel bilgisayar uygulaması (gerçekleştirimi) olarak

bir COW veya MPP için bir kök düğüm bulunur.

(devam)

Hareketli varlıkların yerleştirilmesi (Locating Mobile Entities)

Hiyerarşik Yaklaşımlar (Hierarchical Approaches) :

Ölçeklenebilirlik konuları:

Ayrıca tek makine yapısında diğer işlem gücü yeterli gelmezse ağ sayesinde

arttırılır. Daha iyi bir yaklaşım olarak alt düğümler ağ üzerinden tek biçimli

(uniform) olarak dağıtılırlar. Bununla beraber, eğer iş düzgün yapılmazsa

29

ölçeklenebilirlik problemlerine neden olur. Buna çözüm olarak, alt düğümlerin

olabildiğince kök düğüme yakın yerleşimde bulunmaları gerekliliği verilebilr.

Böylece belirli kök-seviyeli isteklerin daha etkin yerine getirilebilmesi sağlanılır.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Bir varlık için bir referansın bulunması durumunda o varlığın yok edilememesi

(remove) kabul edilemez. Varlıkların yok edilmesinin dağıtık sistemlerde

yönetimi sıklıkla zordur. Özellikle, sistemde bulunan bir varlığa (herahngi bir

yerde) ait referans bilinmediğinde, bir referans ile daha sonra o varlığa erişmek

sıklıkla karşılaşılan bir durumdur, bu durumda varlığın yok edilmesi/silinmesi

sonraki erişimler için (erişim denemeleri için) hata (error) ile sonuçlanır.

30

Bu problemleri azaltmak (referanssız varlıkların yok edilmesi problemleri), dağıtık

sistemin bir varlığa daha fazla ihtiyaç yoksa onun yok edilmesini sağlayan

otomatik imkanları sağlamasıyla mümkün olur. Bu imkanlar topluluk olarak,

Dağıtık çöp toplayıcıları

(Distributed garbage collectors) olarak bilinirler.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referanssız Nesneler Problemi (The Problem of Unreferenced Objects) :

Uzaktaki nesneler göz önüne alınarak, dağıtık nesneler için çöp toplama (garbage

collection) olayına bakılabilir. Bir uzaktaki nesne (remote object) için buna ait

tüm durum (state) bir nesne sunucusunda bulunduğunu ve istemcilerin sadece

bir vekil’e (proxy) sahip olduğunu söyleyebiliriz. Bir uzak nesne (remote object)

31

genelde bir (proxy, skeleton) çifti olarak işaret edilerek (refer) gerçekleştirilebilir.

İstemci-taraflı proxy’ler; proxy’ler ile beraber çöp toplama işini yapmak için

gerekli yetkiye sahip yönetim yapma kısmındaki iskeletiyle (skeleton’u ile), yani

onunla ilişkili iskelet anlamında, daha doğru bir ifade ile, onun nesnesi ile

bağlantı kurabilmek için gerekli tüm bilgiyi içermektedir.

Diğer bir deyişle çöp toplama işi ’nin istemci ve geçerli nesnelerden gizlenilmesi

gerekir. Bir nesne’nin kendisi bir uzak referansı başka bir nesne için tutar, bunun

anlamı bu uzak referanstaki vekil (proxy) için yerel bir işaretçi (pointer)

olmasıdır. Buna benzer olarak, bir uzak referans, kendi ilgili vekili’nin başka bir

sürece kopyalanmasıyla başka bir sürece gönderilebilir/geçirilebilir. Aşağıdaki

anlatımlarda bir nesne’nin sadece ve sadece bir uzak referansı olduğunda

erişilebildiğini varsayalım. Bir nesne için hiçbir uzak referans’ın olmadığında,

sistemden silinmesi (yok edilmesi) gerekir. Diğer yandan, bir nesne için uzak bir

referansa sahip olmak bu nesne’nin daima erişilebilir olduğu anlamına gelmez.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referanssız Nesneler Problemi (The Problem of Unreferenced Objects) :

Çeşitli nedenlerle (iki nesnenin olması da mümkündür), her biri bir diğeri için bir

referansı depolayan ve (fakat) aksine tümü tarafından (diğer nesneler) refere

(işaret) edilmemiş olan böyle iki nesne olması mümkündür. Böyle bir durum en

genel olarak sadece birbirlerini işaret eden (referring only to each other) iki

32

nesne’nin bir çevrimi (cycle) şeklindedir. Böyle nesneler tespit edilmeli ve

silinmeli/yok edilmelidir.

Genelde bu model bir çizge (graf) ile gösterilir, burada her bir düğüm bir nesneyi

göstermektedir. Ayrımı yapılmış bir nesneler alt kümesine kök küme (root set)

denilir ve kendilerini işaret etmeleri (referenced themselves) istenmez. Bir nesne

kök küme (root set) içindeyse, tipik olarak sistem çapında bir servisi, bir

kullanıcıyı vb. göstermektedir.

Şekil 4.28 ‘de, bir referans çizgesi görülmektedir. Tüm beyaz düğümler nesneleri

temsil ederler, bunlar doğrudan olmayan veya dolaylı (indirekt) olarak, kök küme

‘deki nesneler tarafından işaret edilmişlerdir (referenced).

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referanssız Nesneler Problemi (The Problem of Unreferenced Objects) :

Fig. 4.28

33

Şekil 4.28 ’de, birbirlerinin referanslarını içeren nesnelerin temsil edildiği çizge

örneği gösterilmektedir.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referanssız Nesneler Problemi (The Problem of Unreferenced Objects) :

Tek-işlemcili (uniprocessor) sistemde, referanssız nesnenin tespiti ve yok

edilmesi/silinmesi dağıtık bir sisteme göre kolaydır, çünkü dağıtık çöp toplama ağ

bağlantısı/iletişimi gerektirir. İletişim büyük oranda çözümlerin etkinliğini ve

ölçeklenebilirliğini belirler. Bazı iyi bilinen çözümlere bakacak olursak, bu

34

çözümler problemleri kısmen çözmektedir.

Referans Sayma (Reference Counting) :

Tek-işlemcili sistemlerde, popüler bir yöntem ile, bir nesne basitçe bu nesnenin

referansları sayılarak silinebilir. Bir nesne için bir referansın yaratıldığı her

seferde, bu nesne için bir referans sayacı artırılır. Benzer olarak bir referans

silindiğinde/yok edildiğinde, referans sayacı azaltılır. Referans sayacı sıfır’a

ulaştığında, nesne silinebilir.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Basit Referans Sayma (Simple Reference Counting) :

Dağıtık sistemlerde, basitçe referans sayma belirli sayıda probleme yol açar.

Bunların bir kısmı iletişimin güvenli olmamasından kaynaklanır. Genelleştirme

kaybolmaksızın, bir nesnenin kendi referans sayacını kendi ilişkili iskeletinde

35

depolar. Bu iskelet nesne sunucusu tarafından oluşturulur/bakımı yapılır ve

nesne’den sorumludur. Bu durum aşağıdaki Şekil 4.29 ‘dan görülmektedir.

Bir

P süreci, bir

O uzak nesnesi (remote object) için bir referans oluşturduğunda,

O nesnesi için bir p vekilini (proxy) onun adres uzayında kurar (installs).

Referans sayacını artırmak için vekil bir

m mesajını tekrar iletilir. Çift kez

yollanmış mesajları tespit etmek için özel ölçütler yoksa, iskelet kendi referans

sayacını yanlış olarak tekrar artırır. Pratikte böyle duplike (çift kopya) mesajların

tespiti görece basittir.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Basit Referans Sayma (Simple Reference Counting) :

Fig. 4.29

36

Şekil 4.29 ‘da, güvenli olmayan iletişim varlığında düzgün (doğru) bir referans

sayma işleminin bakımı/idaresi ile ilgili problemi gösteren şematik verilmektedir.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Basit Referans Sayma (Simple Reference Counting) :

Benzer olarak, uzak bir referansın silinmiş olması, problemlere yol açabilir. Bu

durumda, bir vekil bir mesajı referans sayacının azaltılması için gönderebilir.

Eğer onay (ACK mesajı) tekrar kaybolursa, o zaman bir mesajın tekrar iletimi bir

37

diğerininkine de yol açabilir (doğru olmayan sayaç azaltımı). Sonuçta, dağıtık

referans sayımındaki duplike

(çift kopya) mesajların tespiti ve geldiklerinde

onların göz ardı edilmesi en temel konudur.

Başka bir çözülmesi gereken problem ise, başka bir sürece bir uzak referansı

kopyaladığımızda oluşur. Eğer süreç

P

1, bir referansı süreç

P2 ‘ye geçiriyorsa,

nesne daha doğrusu onun iskeleti (skeleton) yaratılan yeni referanstan haberdar

olmayabilir. Sonuçta, eğer süreç

P

1, kendi referansını silmeye karar verirse,

referans sayacı sıfıra düşer ve

O nesnesi

P süreci ona daha bağlantı yapmadan

2 süreci ona daha bağlantı yapmadan

silinebilir. Bu problem Şekil 4.30 ‘da şematik olarak görülmektedir.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Basit Referans Sayma (Simple Reference Counting) :

38

Şekil 4.30 ‘da görülenler:

a) Bir referansın başka bir sürece kopyalanması ve daha sonra sayaç artırılması.

b) Bu duruma bir çözüm.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Basit Referans Sayma (Simple Reference Counting) :

Nesne’nin iskeletini

P

1 ‘in bilgilendirmesi bir çözümdür, bunu süreç

P2 ’ye bir

referans geçirerek (göndererek) yapar. Ayrıca, bir süreç asla bir referansın varlığı

hakkında bilgiye sahip olduğu iskelet tarafından onaylanmadan (acknowledge)

39

önce bir referansın silinmesine izin vermez. Bu çözüm bir önceki Şekil 4.30 ‘daki

(b) şıkkında görülmektedir.

Onay (acknowledgement)

O nesnesi tarafından

P2 sürecine,

P2 için O nesnesinin

referansı kayıt ettiğinin kontrolü için gönderilir, daha sonra

P2 ‘nin O nesnesi için

olan referansını silebilmesine de izin verir.

P2 olabildiğince uzun süre

O nesnesinin kendi referansını bildiğinden emin olmadığında,

P2 referans

artırımı için

O nesnesine istek için izin vermez.

Ayrıca, güvenli iletişimde bir referansın geçirilmesi için şimdi üç mesaj

gerektiğini göz önüne almak gerekir. Bu büyük-ölçekli dağıtık sistemlerde

başarım (performance) problemlerine neden olur.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

Basit dağıtık referans sayımı, artırım ve azaltma arasında (referans sayacı) yaşana

bir rekabettir. Bu koşullar sadece ve sadece azaltım işlemleri olursa önlenebilir.

Bu çözüm her bir nesnenin sabit bir toplam ağırlığa sahip olduğu

40

ağırlıklandırılmış referans sayımı

(weighted reference counting) yöntemi

ile olur. Bir nesne oluşturulduğunda, toplam ağırlığı ilişkili iskeleti’nde (burada

S

olarak belirtilmiş) kısmi bir ağırlık ile beraber (ilk değer olarak toplam ağırlığa

atanan değer) depolanır. Buna ait detay Şekil 4.31 (a) şıkkında görülmektedir.

Yeni bir uzak referans (p, s) oluşturulduğunda, kısmi ağırlığın yarısı, yeni p

vekiline atanmış nesnenin iskeleti’nde depolanır. Buna ait detay Şekil 4.31 (b)

şıkkında görülmektedir.

Geri kalan yarı ise

S iskeleti’nde tutulur. Bir uzak referans duplike (çift kopya)

edildiğinde, örneğin,

P

1 sürecinden

P2 sürecine geçirildiğinde,

P

1 ‘in vekil’indeki

kısmi ağırlığın yarısı, kopyalanmış

P2 vekil’ine atanır, bu sırada diğer yarısı ise P1

‘deki vekil ‘de kalır. Buna ait detay Şekil 4.31 (c) şıkkında görülmektedir.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

Bir referans yok edildiğinde nesne’nin iskelet’ine bir azaltım mesajı gönderilir,

bunun sonrasında toplam ağırlıktan silinen referansın kısmi ağırlığı çıkartılır.

Toplam ağırlık sıfıra eriştiğinde, daha fazla uzak referans (remote reference)

41

kalmaz ve böylece nesne güvenle silinebilir. Burada, bu durumda mesajların

kaybolmadığı ve bir kez den fazla kez teslim edilmediğinin varsayıldığını göz

önüne almamız gerekir.

Ağırlıklandırılmış referans sayımında ana problem, sınırlı sayıda referansın

oluşturulabileceğidir. Nesne iskeleti’nin kısmi ağırlığı ve oldukça kısa sürede onun

uzak referansları, sıfıra düşerse (drops to zero), daha fazla referans oluşturulmaz

ve duplike (birebir kopya ile çifte çıkarma) edilmez. Bu probleme çözüm, dolaylı

(indirekt) kullanımdır.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

Fig. 4.31

42

Şekil 4. 31 ‘de görülenler:

a) Ağırlıklandırılmış referans sayımında, ağırlıkların ilk kez atanması.

b) Yeni bir referans oluşturulmasında, ağırlık atanması.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

Fig. 4.31

43

Şekil 4. 31 ‘de görülenler:

c) Bir referans kopyalandığında, ağırlık atanması.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

Süreç

P

1 ‘in bir referansı

P2 sürecine geçirmek (göndermek) istediğini varsayalım,

fakat kendi vekil’inin kısmi ağırlığı

1 ‘e ulaşmış olsun. Bu durumda,

P

1 bir

iskeleti’ne benzerdir. Bir vekil daha sonra

P2 ’ye (gönderilir), ona iskeleti’nin

s′

s′

44

iskeleti’ne benzerdir. Bir vekil daha sonra

P2 ’ye (gönderilir), ona iskeleti’nin

yarı kısmi ağırlığı ile atanmış haliyle gönderilir. Diğer yarı (ağırlığın diğer yarısı)

diğer vekil’lere dağıtılmak üzere iskeletinde kalır. Eğer ‘ye ait toplam

ağırlık

1 ‘e ayarlanırsa, bu yaklaşım

P2 sürecinden

P

1 sürecine iletilen bir işaretçi

oluşturulması ile aynıdır.

Eğer, dönüşte

P2 kendi referansını geçirmek (göndermek) isterse, başka bir

iletilen işaretçi oluşturmalıdır. Buradaki en ciddi problem uzun zincir’in (iletilen

işaretçi zinciri) başarım kaybına neden olmasıdır. Zincirler hataya karşı hassastır.

Dolaylı yolu kullanmaya bir alternatifte oluşturum [nesil] referans sayımı

s′

s′ s′

Dolaylı yolu kullanmaya bir alternatifte oluşturum [nesil] referans sayımı

(generation reference counting) olarak bilinir. Her bir vekil kaç kez

kopyalandığını bir sayaç ile tutarak depolar, bu bir nesil numarasına ek’tir. Yeni

bir (p, s) referansı oluşturulduğunda, ilgili

p vekili’nin nesil numarası sıfıra

ayarlanır. Çünkü

p ‘den henüz hiçbir kopya oluşturulmamış ve kopya sayacı şu an

sıfıra ayarlıdır. Nesil referans sayımı, güvenli iletişimi gerektirir, fakat bir kopya

oluşturulmasında iskelet’le bağlantıya ihtiyaç duymadan referans

duplikasyonu’nu idare edebilir. (devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

45

Fig. 4.32

Şekil 4.32 ‘de, bir referans kısmi ağırlığı

1 ‘e ulaştığında bir dolaylı yolun

oluşturulması şematik olarak görülmektedir.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Sayma (Reference Counting) :

Gelişmiş Referans Sayma (Advanced Reference Counting) :

46

Fig. 4.33

Şekil 4.33 ‘de, nesil referans sayımında uzak bir referansın oluşturulması ve

kopyalanması şematik olarak görülmektedir.

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Listeleme (Reference Listing) :

Referans yönetiminde farklı bir yaklaşımda, iskelet’in ona referansı olan vekil’leri

kayıt altında tuttuğu yaklaşımdır. Diğer bir deyişle referans saymak yerine, bir

iskelet’in, kendini referans gösteren tüm vekiller için açık bir liste’nin

bakımını/idaresini yapmasıdır.

47

Böyle bir referans listesi’nin bazı özellikleri bulunur. Bir vekil’in referans listesine

eklenilmesi, vekil hali hazırda listede bulunuyorsa bir etki etmez. Benzer olarak,

olmayan bir vekil’in listeden silinmesi’ninde bir etkisi olmaz. Vekil ekleme ve

çıkarma işlemi idempotent (denk güçlü, aynı güce sahip) işlemlerdir.

Ulaşılamayan Varlıkların Tanımlanması (Identifying Unreachable

Entities) :

Dağıtık sistemdeki tüm varlıkların taranması istenildiğinde bir yöntem vardır.

Genelde, kök küme’den (root set) erişilebilen varlıkların kontrol edilmesiyle bu

yapılır ve tüm diğerleri silinir. Böyle yöntemler genelde tarama-tabanlı çöp

toplama

(tracing-based garbage collection) olarak adlandırılır. Bu yöntem için

ölçeklenebilirlik problemleri mevcuttur, çünkü tüm varlıkları dağıtık sistem içinde

taramayı gerektirir.

(devam)

Referanssız Varlıkların Yok edilmesi (Removing Unreferenced Entities)

Referans Listeleme (Reference Listing) :

Saf (naif) Tarama Yöntemleri (Naive Tracing in Distributed Systems) :

Dağıtık tarama-tabanlı çöp toplamayı anlamak için ilk önce tek-işlemcili

(uniprocessor) sistemlerde taramanın nasıl çalıştığının bilinmesi gerekir.

Tek işlemcili sistemlerde en basit tarama yaklaşımı işaretle

-ve

-temizle

(mark

- 48 Tek işlemcili sistemlerde en basit tarama yaklaşımı işaretle

-ve

-temizle

(mark

- and-sweep) tarzı toplayıcılardır.

İşaretleme fazı boyunca, kök küme’deki varlıklardan oluşturulan referans

zinciri takip edilerek varlıklar taranır.

Her bir varlığa işaretlenmiş olarak bu yolla ulaşılabilir, örneğin varlıkların ayrı bir

tabloya kaydı alınabilir. Temizleme fazı, işaretlenmiş varlıkların yerleştirilmesi

için (yerlerinin belirlenmesi için) sonuna kadar tüketilen belleği içerir. Böyle

varlıklar çöp olarak kabul edilir ve yok edilmeleri gerekir.

(devam)

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

49

Ders’in sonu…

50

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 12

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 12: Senkronizasyon, Saat senkronizasyonu,

Mantıksal saatler, Genel durum.

Senkronizasyon

• Bu konuya bakıldığında, birden çok sürecin paylaşımlı bir kaynağa eşzamanlı

erişememesi önemli bir konudur (printer gibi paylaşımlı bir kaynağa), fakat

bunun yerine geçici olarak birbirlerini dışlayan (mutually exclusive) erişme izin

verecek şekilde işbirliği yapmaları daha önemlidir. Örneğin, çoklu süreçler

bazen olayların sırlamasında anlaşılması için gerekli olabilir, yani P sürecinden

3

gelen

m

1 mesajı,

Q sürecinden gelen

m

2 mesajından önce veya sonra

gönderilebilir.

• Dağıtık sistemlerde senkronizasyon, tek işlemcili veya çoklu işlemcili

sistemlerdeki senkronizasyondan daha zordur. Bir çok durumda bir süreç

grubundaki bir süreç koordinatör olarak belirlenir, bu seçim algoritmaları

(election algorithms) ile yapılır ve önemli bir durumdur.

• Senkronizasyon, Senkronizasyon, karşılıklı birbirini dışlama karşılıklı birbirini dışlama’nın (mutual exclusion mutual exclusion) dağıtık ) dağıtık

sistemler ve dağıtık işler (transactions) için önemli konularıdır. Dağıtık karşılıklı

birbirini dışlama, çoklu süreçlerin eşzamanlı erişmiinden paylaşımlı kaynakları

korumaya izin verir. Dağıtık işlemler benzer şeyler yapar, fakat ileri (gelişmiş)

eşzamanlı kontrol mekanizması sayesinde optimize erişim sağlarlar.

(devam)

Senkronizasyon

Fig. 5.1

4

• Şekil 5.1 ‘de görüldüğü üzer, her bir makine kendine ait saatine sahip

olduğunda, başka bir olaydan sonra oluşan bir olay daha önceki bir

zamana yinede atanmış olabilmektedir.

Senkronizasyon

Saat Senkronizasyonu (Clock Synchronization) :

Bir merkezi sistemde, zaman belirsiz değildir. Bir süreç zamanı bilmek isterse, bir

sistem çağrısı yaparak zamanı çekirdek ‘e (kernel) sorar. Eğer A süreci zamanı

sorarsa ve çok az bir süre sonra B süreci zamanı sorarsa, B süreci A ‘nın

zamanından daha büyük (veya ona yakın veya eşit) ir değer alır. Fakat daha düşük

5

bir zaman değeri almaz. Dağıtık bir sistemde zaman konusunda anlaşma söz

konusu değildir.

Bir dağıtık sistemde zaman hakkında global bir anlaşma bulunmaz. Örneğin

yazılan bir program derlenince, sonuçta çıkan çalıştırılabilir binary (çalıştırılabilir) program nesne dosyalarını hem eski hem de yeni kaynaklardan oluşacak bir

karışım şeklinde içerir. Bu programın çökmesi mümkündür ve programcı hatanın

neden kaynaklandığını kod’a bakarak anlayamaz. Buradaki en önemli soru şudur:

“Bir dağıtık sistemdeki tüm saatleri senkronize etmek mümkün müdür?”

(devam)

Senkronizasyon

Saat Senkronizasyonu (Clock Synchronization) :

Fiziksel Saatler (Physical Clocks) :

Neredeyse tüm bilgisayarlarda zaman tutmak için bir devre vardır. “Saat”

kelimesinin anlmaı bu bakış açısından biraz farklıdır. Timer (zamanlayıcı) daha

uygun bir kelime olmaktadır. Bir bilgisayar timer ’ı, özel bir teçhizat ile, quartz

6

kristali ile oluşturulmuştur. Bu kristal baskı (tension) altında tutulduğunda,

belirli frekansta (Cessium atomu) salınım yapar. Kristalin her salınımı bir sayacı

(counter) ve bir tutma kayıtçısı’nı (holding register) değerini azaltır.

Bir sayaç sıfırlandığında, bir kesinti (interrupt) oluşturulur ve tutma

kayıtçısı’ndan bu sayaç tekrar doldurulur. Böylece bir timer saniyede 60 kesinti

oluşturacak şekilde programlanabilir (saniyede 60 kesinti). Her bir kesintiye saat

tıklaması

(clock tick) denilir.

Not: “Register” kelimesine bazı kaynak kitaplarda “kayıtçı” bazılarında ise

“yazmaç” olarak verilmektedir.

(devam)

Senkronizasyon

Saat Senkronizasyonu (Clock Synchronization) :

Fiziksel Saatler (Physical Clocks) :

Çoklu işlemciler çıkar çıkmaz, her birinin kendi saati olmuştur ve iş değişmiştir.

Bir kristalin salınımı kararlı (stabil) olabilir, fakat kristalin farklı bilgisayarların

hepsinde tam aynı frekansta çalışması garanti edilemez. Pratikte

n tane

7

bilgisayarı olan bir sistemin tüm

n tane kristali tamamen farklı oranlarda

(frekanslarda) çalışır, (yazılım olarak) saatler okunduğunda farklı değerler

vererek, senkronize olmaktan dereceli olarak çıkarlar. Bu saat değerlerindeki

farklılığa saat eğriliği

(clock skew) denilir. Bu saat eğriliği neticesinde, bir

dosya, süreç veya mesajla ilgili zamanın doğru olmasını programlar bekler ve

bunların oluşturulduğu makineden bağımsız olması da beklenir.

Bazı sistemlerde (yani gerçek

-zamanlı sistemler) gerçek saat zamanı önemlidir.

Böyle sistemlere harici fiziksel saat gerekmektedir. Bu durumda iki sorun ortaya

çıkar:

1-) Bu saatleri gerçek dünya saatleri ile nasıl senkronize edeceğiz?

2-) Bu saatleri birbirleri ile nasıl senkronize edeceğiz?

(devam)

Senkronizasyon

Saat Senkronizasyonu (Clock Synchronization) :

Fiziksel Saatler (Physical Clocks) :

Güneşin gökyüzünde en üst noktaya çıktığı duruma, güneş transit’i denir. Bu olay

öğlen vakti olur. İki ardışık güneş transit’i arası zamana güneş günü (solar day)

denilir. 24 saat sürer ve her saat 3600 solar saniye sürer. Bu tam olarak 1/86400

8

denilir. 24 saat sürer ve her saat 3600 solar saniye sürer. Bu tam olarak 1/86400

güneş günü yapar. Daha önceki yıllar ile günümüzdeki günler arası averaj bir ölçü

alınarak 86400 ‘e bölündüğünde ortalama solar saniye (mean solar second) elde

edilir. Dünya çapında 50 laboratuvarın Cessium133 saatlerinden alınan sonuçları

Paris’teki BIH

(Bureau International de l’Heure) kurumuna bildirmeleri ile bu

değerlerin ortalaması alınarak Uluslar arası Atomik Zaman

(International

Atomic Time, TAI) elde edilir.

BIH, kaçak saniyelerin (leap seconds) oluşturduğu problemi (TAI ile solar saniye

arası 800 msec ‘lik farktan dolayı oluşan) çözmüştür. Buna çözüm ise Evrensel

Koordinasyon Zamanı

(Universal Coordinated Time, UTC) olarak bilinir.

Buna uyum için kayıp saniye oluşunca elektrik enerjisi üreten firmalar 50 Hz’ i

51 Hz ‘e, 60 Hz ‘i 61 Hz ‘e eşitleyerek bu kayıp saniyeyi telafi etmeye çalışır.

(devam)

Senkronizasyon

Fig. 5.2

9

Şekil 5.2 ‘de ortalama solar gün’ün hesaplanması görülmektedir.

Fig. 5.3

Senkronizasyon

10

Şekil 5.3 ‘de, TAI ve UTC saniyelerinin karşılaştırılması

görülmektedir.

Senkronizasyon

Saat Senkronizasyonu (Clock Synchronization) :

Fiziksel Saatler (Physical Clocks) :

Hassas ölçümlü saat isteyen insanlar için, Standart Zaman Ulusal Enstitüsü

(National Institute of Standart Time, NIST) bir kısadalga radyo istasyonu’ndan

(WWV istasyonu) her bir UTC saniyesi başında kısa bir sinyal (pulse) yayınlar. Bu

11

duruma göre (hava şartları) ve arası değişkenlik (yayınlama

gösterebilir. Uydu ile de bu sisteme benzer olarak ile UTC işlemleri

Yerdurağan Ortam İşlemsel Uydusu

(Geostationary Environment

Operational Satellite, GEOS) sayesinde gerçekleşir.

±

1msec

±10msec

±

0.5msec

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Eğer bir makinenin WWV (kısadalga radyo yayını ile UTC saniye bilgisi almak

için) alıcısı varsa diğer tüm makineler buna göre senkronize edilebilirler. Eğer

hiçbir makinenin WWV alıcısı yoksa her makine kendi zamanını tutacaktır ve asıl

amaç mümkün oldukça bu makineleri bir arada tutabilmektir.

12

Tüm algoritmalar aynı temeldeki sistem modeline sahiptir. Her bir makine, bir

saniyede

H kez kesintiye sebep olan bir timer’a sahiptir. Eğer timer durursa,

kesinti (interrupt) handler ’ı, tıklama (tick) sayısını (kesintilere ait olan) takip

eden yazılımsal saate

1 değer ekleri çünkü geçmişteki zaman üzerinde

anlaşılmıştır. Bu saatin değerine C diyelim. Daha özel olarak, UTC zamanı

t iken,

p makinesindeki saatin değeri;

C

(

t

)

olur. Mükemmel bir dünyada, şeklinde tüm

p ‘ler ve tüm

t ‘lere sahip

olunur. Diğer bir deyişle,

ideal olarak bulunur.

(devam)

C

(

t

)

p

C

t

t

p

(

)

=

=

1

dt

dC

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Gerçek timer ’lar bir saniyede H kez kesintiye uğramazlar. Teorik olarak H=60 ile

bir timer 216000 tıklama (tick) saat başına üretir. Modern timer chip’leri için

göreli hata (relative error) 10-5 olmaktadır.

13

eşitliği belirtilirse, burada değeri, üretici tarafından belirtilen maksimum

kaydırma oranı

(maximum drift rate) olarak bilinir.

1

−

ρ

≤

≤

1

+

ρ dt

dC

ρ

(devam)

Senkronizasyon

Fig. 5.4

14

• Şekil 5.4 ‘de, farklı oranlarda tıklama yapan saatler olduğunda fiziksel saat

ve UTC saati arasındaki ilişki gösterilmektedir.

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Eğer iki saat de UTC ’den kayıyorlarsa (uzaklaşıyorlarsa) farklı yönlere doğru ve

senkronize olduktan sonra arada kadar zaman geçmişse, bu iki saat

birbirinden kadar uzaklaşmıştır. İşletim sistemi tasarımcıları aradaki farkın

den daha fazla olmamasını garanti edeceklerse (yazılımsal olarak) en azından

∆

t

2

ρ∆

t

δ

15

den daha fazla olmamasını garanti edeceklerse (yazılımsal olarak) en azından

her saniyede bir bu saatlerin senkronize edilmesi gerekir. Senkronizasyon

algoritmaları bu senkronizasyonun nasıl yapılacağı konusunda birbirlerinden

ayrılırlar.

Cristian Algoritması (Cristian’s Algorithm):

Bu algoritma başlarken, zaman sunucusu (time server) olarak adlandırılan ve

WWV alıcısına sahip bir makinenin sayesinde diğer makinelerinde ona göre

senkronize olduklarını belirtmek gerekir. Periyodik olarak, her saniyeden

δ

δ

2

ρ

senkronize olduklarını belirtmek gerekir. Periyodik olarak, her δ

2 saniyeden ρ

uzun olmayacak aralıkta, her bir makine bir mesajı zaman sunucusu‘na

gönderecek şu anki zamanı sorar. Time server makinesi olabildiğince hızlı bir

şekilde kendi şu anki zamanı, CUTC ‘yi içeren mesajla beraber geri gönderir.

Şekil 5.5 ‘te, T

0 ve T1 aynı saatle ölçülmüştür.

(devam)

δ

2

ρ

Cristian's Algorithm

Senkronizasyon

Fig. 5.5

16

Şekil 5.5 ’te , şu anki zamanın time server’dan (zaman sunucusu) alınması

gösterilmektedir.

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Cristian’s Algorithm:

İlk yaklaşımda, gönderici cevabı aldığında, sadece kendi saatini CUTC ‘ye ayarlar.

Bununla beraber, bu algoritma’nın iki problemi bulunur; biri büyük, biri küçük

olmak üzere iki problemi vardır. Büyük problem zamanın asla geriye doğru

17

gitmeyeceğidir. Eğer göndericinin saati hızlıysa (fast clock), CUTC göndericinin şu

anki

C değerinden daha küçük olabilir. CUTC ‘nin sadece alınmasıyla ciddi

problemler oluşabilir, böyle bir nesne dosyasının kaynak kod’dan daha önce

derlenmesi problemi oluşması kaçınılmazdır. Böyle bir değişim dereceli olarak

yapılmalıdır.

Şöyle bir yol takip edilir:

Bir timer’ın saniyede 100 kesinti yaptığını varsayalım, normale olarak her bir

kesinti zamana 10 msec ekler. Yavaşladığında, kesinti rutini sadece

9 msec kadar

her bir seferde düzeltme yapılana kadar ekler. Benzer olarak, saat dereceli olarak

ilerletilebilir ve 11 msec her bir kesinti olduğunda, hepsini birden atlamak yerine

(tüm ara mesafeyi) eklenebilir.

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Cristian’s Algorithm:

Küçük problem ise, göndericiye geri gönderilen zaman sunucusu‘nun (time server)

sıfır olmayan zaman cevabı (yani hiç fark olmasın sunucu ile istemci saatleri

arasında) bulunduğunda oluşur. Burada problem, ağ gecikmesinden kaynaklı

18

olabilir (çünkü gecikme saat tabanlı değil başka nedenlerdendir, yani sunucu ile

istemci ’nin saatleri aynı değeri gösteriyor), veya ağ yükünden kaynaklanıyor

olabilir.

Cristian algoritması bunu ölçmeyi denemektedir. En basit şekilde gönderici, isteğin

gönderimi ile (zaman sunucusuna) ve cevabın ulaşması arasındaki süreyi doğru

olarak kaydetmesi yeterlidir. Burada

T0 başlangıç zamanı ile,

T1 bitiş zamanı aynı

saatle ölçülür, böylece eğer göndericinin saati önemli bir süre UTC‘den uzak

(bağımsız) kaldığında görece doğru bir süre aralığı elde edilmiş olur.

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Cristian’s Algorithm:

Diğer bilgilerin yokluğunda, mesaj yayılım süresinin en iyi ölçümü

olarak verilir. Cevap geldiğinde, mesajdaki değer bu miktarla artırılarak bir

sunucu’nun şu anki zamanı’nın hesabını (ölçümünü) verir. Eğer teorik minimum

(

)

2

T1 −

T0

19

sunucu’nun şu anki zamanı’nın hesabını (ölçümünü) verir. Eğer teorik minimum

yayılım zamanı biliniyorsa, zamanın diğer özellikleri hesaplanarak ölçülebilir. Bu

ölçümün daha da iyileştirilmesi, zaman sunucusu’nun kesintiyi idare etmesi ve

gelen mesajı işlemesinin ne kadar süre aldığının yaklaşık olarak bilinmesi ile

mümkün olmaktadır. Kesintiyi idare etme süresi’ne (Interrupt Handling

Time , I) dersek,

T0 ile

T1 arası sürede mesaj yayılımı ile geçen süre olur.

Tek yönlü yayılım zamanının en iyi ölçümü bunun yarısını almaktadır. Burada

A‘dan B ‘ye giden mesajların farklı yol izledikleri durumlar göz önüne

T

−

T

−

I

1 0

A‘dan B ‘ye giden mesajların farklı yol izledikleri durumlar göz önüne

alınmamıştır.

Doğruluğu artırmak için Cristian, bir tek ölçüm yerine seri ölçümler önermiştir.

Bu sayede ‘ı aşan herhangi bir ölçüm (yani bu aralık için eşik değerini aşan

ölçümler) ağ tıkanıklığı’nın kurbanı olarak göz ardı edilir ve güvensiz bulunurlar.

(devam)

T1 −

T0

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Cristian’s Algorithm:

Geriye kalan denemeler ve ölçümlerden elde edilen değerlerin ortalaması alınarak

daha iyi bir değer elde edilir. Alternatif olarak daha hızlı olarak geri gelen mesaj

daha az trafikten geçtiği kabul edilerek daha doğru varsayılır ve saf yayılım

20

zamanını gösterdiği belirtilir. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Berkeley Algoritması:

Cristian algoritmasında zaman sunucusu pasif’tir. Diğer makineler periyodik

olarak ona zamanı sormaktadırlar. Tümü kuyruklarına karşı sorumluluklarıyla bu

işi yapmaktadır. Berkeley UNIX ‘te, tamamen karşıt bir yaklaşım ele alınır. Burada

zaman sunucusu (aslında bir time deamon) aktif’tir, her bir makine zamandan

zamana ondaki zamanın ne olduğunun sorulması ile kontrol altında tutulur

(polling every machine from time to time to ask what time it is there). Cevaba

dayalı olarak, ortalama bir zamanı hesaplar ve diğer tüm makinelere bunu söyler,

böylece makineler ya saatlerini ilerletir yada bu değere uydurmak için saatlerini

yavaşlatırlar. Bu yöntem WWV alıcısına sahip olan hiçbir makine

bulunmadığındaki sistem için uygundur. Time daemon‘ın zamanı operatör

tarafından periyodik olarak elle (manuel) ayarlanmalıdır.

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Berkeley Algoritması:

Saat 3:00 ‘da, time daemon diğer makinelere kendi zamanını söyleyip, onların

zaman değerlerini sorar (Bu anda time daemon 3:00, bir makine 2:50, diğeri 3:25

değerinde olsun). Daha sonra her makine kendi değeri ile cevap verir. Bu değerler

21

ile ortalama bir değer hesaplayan time daemon, her bir makineye kendisini (bu

üye makinelerin) ayarlayacağı (saatlerini) söyler. Yani senkronizasyon sağlanılmış

olur (Örnekteki birinci makine +15, ikincisi de -20 yapıyor).

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Berkeley Algoritması: Fig. 5.6

22

Şekil 5.6 ‘da gösterilenler;

a) Time daemon, diğer tüm makinelere onların saat değerlerini sorar.

b) Makineler cevaplandırırlar.

c) Time daemon, herkese saatlerini nasıl ayarlamaları gerektiğini söyler.

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Ortalama alma algoritmaları (Averaging Algorithms):

Yukarıda bahsedilen her iki yöntemde oldukça merkezi ve genelde dezavantajlıdır.

Merkezi olmayan algoritmaları göz önüne alarak, merkezi olmayan (decentralized)

saat senkronizasyon algoritmalarının bir sınıfı, zamanı sabit

-uzunluklu

23

resenkronizasyon aralıklarına bölerek çalışır. Burada

i inci aralık ile başlar

ve olana kadar sürer, buradaki

T0 geçmişte üzerinde anlaşmaya

varılmış an’dır (moment) ve

R bir sistem parametresidir. Her bir aralığın

başlangıcında, her bir makine kendi saatine göre şu anki zamanı yayımlar. Çünkü

farklı makinelerdeki saatler tam olarak aynı hızda çalışmazlar, bu yayım tam

(özellikle) eş zamanlı olmayacaktır.

Bir makine kendi saatini yayımladıktan (broadcast) sonra, diğer tüm yayımları

T0

+ iR

T0

+

(

i

+ )1

R

Bir makine kendi saatini yayımladıktan (broadcast) sonra, diğer tüm yayımları

(bu yayımlar

S aralığı boyunca bu makineye ulaşmaktadırlar) bir yerel timer için

toplamaya başlar. Tüm yayımlar ulaştığında, bir algoritma onlar için yeni bir

zamanı hesaplamak için çalışır. En basit algoritma sadece tüm diğer makinelerden

alınan değerlerin ortalamasını alır. Önemsiz bir varyasyon (bu konunun bir

varyasyonu) en yüksek

m ve en düşük

m değerlerinin göz ardı edilmesi ilk etapta,

daha sonra kalanların ortalamasının alınmasına dayanır.

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Ortalama alma algoritmaları (Averaging Algorithms):

Bu sayede anlamsız

m hatalı saatlerine karşı kendini korumayı böyle uç değerleri

göz ardı ederek sağlanmasıdır. Her bir mesajı düzeltmeye uğraşan başkja bir

varyasyon kaynaktan yayılım zamanının hesabına bu mesajlara ekler. Bu hesap,

24

ağın bilinen topolojisi ile oluşturulabilir veya deneme mesajlarının

yansılanmasının (echoed) ne kadar bir süre aldığının ölçülmesi ile oluşturulur.

Internet’te en çok kullanılan algoritmalardan biri de Ağ Zaman Protokolü

(Network Time Protocol, NTP) olmaktadır. NTP,

1 ilâ 50 msec arası doğrulukta

(dünya çapında) başarılır (yani mesaj süre aralığı max. 50 msec). Bunu gelişmiş bir

saat senkronizasyon sistemi ile başarmaktadır.

(devam)

Senkronizasyon

Saat Senkronizasyon Algoritması (Clock Synchronization Algorithms) :

Çoklu Harici Zaman Kaynakları (Multiple External Time Sources):

UTC ile oldukça fazla doğrulukta senkronizasyon gerektiren sistemlerde, sistemi

birden çok WWV alıcısıyla donatmak mümkündür (Sadece WWV değil GEOS veya

diğer UTC alıcılarıyla da donatılmış olabilir). Bununla beraber, zaman

25

kaynaklarındaki uyumsuz doğru olamama nedeniyle kendiliğinden sinyal yolunda

oldukça değişikliğe uğrar, en iyi işletim sistemi UTC ‘nin yanlış olduğu durumda

bir zaman aralık’ı (time range) oluşturur.

Genelde çeşitli zaman kaynakları farklı aralıklar (ranges) üretir, genel bir

anlaşmaya uymaya çalışan eklenmiş makineler buna ihtiyaç duyar. Bu anlaşmaya

ulaşmak için, bir UTC kaynağına sahip her bir işlemci, kendi aralığını periyodik

olarak yayımlar, örnek olarak her bir UTC dakikası başlangıcında özellikle

yayımlanır. Hiçbir işlemci zaman paketlerini anlık olarak alamaz. Daha kötüsü,

iletim ile kabul arası gecikme, kablo mesafesi ve paketin dolaştığı yönlendirici

(router) sayısına bağlıdır. Bunlar her biri için (UTC kaynağı ve işlemci çifti)

farklıdır. Ayrıca Ethernet üzerinden gönderimde çarpışma (paket bazlı çarpışma)

söz konusu olduğunda yaşanan gecikmelerde bulunur, veya bir işlemci aldığı bir

önceki paket ile meşgul ise bu sırada belirli süre (milisaniyeler mertebesinde)

zaman paketlerine bakılmaz.

(devam)

Senkronizasyon

Senkronize Saatlerin Kullanımı (Use of Synchronized Clocks) :

Yeni teknolojiler sayesinde milyonlarca saati birkaç UTC milisaniyesi içinde

senkronize etmek mümkündür. Yeni algoritmalar senkronize saatleri kullanır hale

gelmiştir. En önemli sorun, bir sunucuya en çok bir kez (at-most-once) mesaj

teslimatının nasıl sağlanacağı eğer bu sırada çökme meydana gelirse ne

26

olacağıdır.Geleneksel yaklaşımda, her bir mesaj için bir tekil mesaj numarası

taşınması ve her bir sunucunun, mesajlara ait tüm numaraları depolaması

kararlaştırılmıştır. Bu depolama her yeni mesajın (retransmisyondan gelen) tespiti

ile olur.

Asıl soru, mesaj numaralarının (sayılarının) daha ne kadar süre kaydedileceğidir?

Zaman kullanılarak algoritma şöyle değiştirilebilir: Her bir mesaj bir bağlantı

tanımlaycısı (gönderici tarafından seçilen) taşır ve bir zaman damgası (timestamp)

taşımaktadır. Her bir bağlantı için, sunucu en güncel gözlemlenmiş zaman

damgası’nı bir tabloya kaydeder. Eğer bir bağlantı için gelen herhangi bir mesaj,

zaman damgası’ndan daha düşük (bu bağlantı için kaydedilmiş zaman

damgası’ndan düşük) değerli ise, mesaj duplike (çift kez yollanmış) olarak kabul

edilir ve red edilir.

(devam)

Senkronizasyon

Senkronize Saatlerin Kullanımı (Use of Synchronized Clocks) :

Eski zaman damgalarını silebilmek için, her bir sunucu sürekli global bir

değişkenin bakımını yapar:

G

=

Şu anki zaman - Max.Ömür Zamanı - Max. Saat

Eğriliğ

i

27

(İngilizcesi: )

Burada, Max Life Time, bir mesajın canlı olduğu maksimum zaman ve Max. Clock

Skew ise en kötü durum olarak bakılan saatin UTC ‘den ne kadar uzakta olduğu

(ileri veya geri kalma miktarı) olur.

G değerinden daha eski zaman damgası,

güvenle tablodan kaldırılır, çünkü tüm eski zaman damga’lı mesajlar hali hazırda

ölüdür. Eğer gelen bir mesaj bilinmeyen bir bağlantı tanımlayıcıya sahip ise,

G‘den

daha güncel bir zaman damgası’na sahip ise kabul edilir, eğer daha eski bir zaman

G

= Current ime - Max.Life Time - Max.Clock Skew

daha güncel bir zaman damgası’na sahip ise kabul edilir, eğer daha eski bir zaman

damgasına sahip ise duplike olduğuna karar verilip red edilir. Bunun etkisi, G tüm

eski mesajların mesaj numaraları (sayıları) toplamı olmasıdır. Her ‘de, şu anki

zaman disk’e yazılır.

(devam)

∆

t

Senkronizasyon

Senkronize Saatlerin Kullanımı (Use of Synchronized Clocks) :

Bir sunucu çöktüğünde tekrar boot ederek açılır ve disk’e depolanan zamandan

G‘yi tekrar yükler ve güncelleme periyodu ile onu artırır. Herhangi bir gelen mesaj

G’den daha eski bir zaman damgası’na sahipse duplike olarak belirlenerek red

edilir. Sonuçta, çökmeden evvel kabul edilmiş tüm mesajlar red edilir.

28

Bazı yeni mesajlar doğru olmasa da red edilir, fakat tüm koşullar altında algoritma

en çok bir kez (at-most-once) semantik’ini (anlamsallık’ını) sağlar. Bu algoritmaya

ek olarak, önbellek tutarlılığı başarılarak saatlerin nasıl senkronize edileceği,

“vakit doldu” biletlerinin dağıtık sistem yetkilendirilmesinde nasıl kullanılacağı ve

atomik işlerin (transactions) nasıl idare edileceği tanımlanmıştır (yeni geliştirilen

ileri algoritmalarda).

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Bir çok amaçla tüm makinelerin aynı zaman üzerinde anlaşmaları yeterli olur.

Radyodan her saat başı anons edilen gerçek saat ile bunların aynı olmasına temel

de gerek yoktur. Örnek olarak “make” komutunda tüm makineler saat 10:00 ‘u

gösterirken aslında 10:02 olabilir. Bazı algoritma sınıfları, saatlerin içsel

29

tutarlılıkları sağlamaya çalışır (yani gerçek zamana yakın olup olmamalarının

önemi yoktur). Böyle algoritmalar, mantıksal saatler olarak bilinir (gerçek

zamanla değil kendi içlerinde (birbirleriyle) tutarlıdırlar). Lamport’un klasikleşmiş

makalesinde mutlak olması gerekmeyen saat senkronizasyonunun mümkün

olduğu gösterilmiştir.

Eğer iki süreç etkileşmiyorsa, bunların saatlerinin senkronize edilmesine gerek

yoktur, çünkü senkronizasyon eksikliği gözlemlenmez ve probleme yol açmaz.

Genel olarak tüm süreçlerin tam olarak zamanın şu an ne olduğuna anlaşmasının

önemli olmadığı, fakat olayların oluş sıralarında anlaşmaya varmalarının daha

önemli olduğu ortaya çıkar.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Mantıksal saatleri senkronize etmek için Lamport, happens-before isimli bir

ilişki tanımlamıştır. Burada;

a

→

b ifadesi “a happens before b” yani “

a olayı

b olayı‘ndan önce

30

ifadesi “a happens before b” yani “

a olayı

b olayı‘ndan önce

gerçekleşmiştir” olarak okunur. Buna göre

a olayı oluştuktan sonra

b olayı

oluşmakta olduğu kabul edilir. Bu ilişki iki durumda doğrudan gözlemlenir.

1

-) Eğer

a ve

b olayları aynı süreçte ise ve

a olayı

b ‘den önce olduysa, o zaman

doğrudur.

2-) Eğer

a, bir süreç tarafından gönderilmiş bir mesajın olayıysa ve

b, başka bir

süreç tarafından alınan mesajın olayıysa, o zaman doğrudur. Bir mesaj

gönderilmeden önce alınamayacağı için veya gönderildiği sıradaki olay, sonlu

bir süre (sıfır değerinde zaman aralığına sahip olmayan varışla) alır. Yani sıfır

a

→

b

a

→

b

a

→

b

bir süre (sıfır değerinde zaman aralığına sahip olmayan varışla) alır. Yani sıfır

süre geçmez belirli bir süre geçer.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Happens-before bir transitif ilişkidir (geçişli), eğer ve ise, o zaman

olur. Eğer x ve y gibi iki olay, farklı süreçlerde olduysa ve bunlar mesaj

alışverişinde bulunmuyorsa, (ayrıca üçüncü parti yoluyla dolaylı olarak da

a

→

b

b

→

c

a

→

c

31

alışverişinde bulunmuyorsa, (ayrıca üçüncü parti yoluyla dolaylı olarak da

bulunmuyorsa, o zaman doğru değildir, fakat de değildir. Bu

olaylara eşzamanlı (concurrent) denilir. Bu durumda hangi olayın hangisinden

önce veya sonra oluştuğu söylenemez veya söylenmesi gerekmez.

Her olayın süre ölçümü bir zaman değeri atanarak yapılır. Örneğin a olayı için

zaman değeri atanır. Ve bunu tüm süreçler kabul eder. Saat zamanı

C daima

ileri doğru giden artan ve asla geri gitmeyen (azalan) bir değer olarak bilinir.

Zaman düzeltmeleri için pozitif bir değer ataması yapılır, (negatif olacak şekilde)

x

→

y

y

→

x

C

(

a

)

değer çıkarılmaz.

Aşağıdaki şekilde her bir zaman süresi tıklama (tick) sayısı olarak verilmiş üç

farklı süreçte gerçekleşen (her bir sürecin kendi saati bulunmaktadır) olaylar

görülmektedir. Senkronize olmak için ilgili olay (bir süreçten diğerine mesajın

ulaşması) oluştuğunda geride kaldığı tespit edilen süreç yerine yeni sürecin saatine

+

1 değer eklenerek yola devam edilmektedir.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Küçük bir ekleme ile bu algoritma isterleri global saat için karşılayabilecek hale

gelir. Her iki olay arası ekleme, saatin en az bir kez tıklama (tick) atmasını

gerektirir. Eğer bir süreç iki mesajı arka arkaya çok hızlı gönderip alıyorsa, bu iki

32

olay arasında (en azından) saatini bir tıklama (tick) kadar ileri alması gerekir.

Bazı durumlarda, ek bir ister gerekebilir: hiçbir iki olayın tam olarak aynı zamanda

olmaması durumu. Bu amacı başarabilmek için, belirli sayıda süreci düşük-sırada

(öncelikte) zaman sonunda oluşan olaylar için ekleyebiliriz, yani süreç

1 ve süreç

2‘deki olaylar 40 ıncı saniye‘de olmuşsa öncekisi 40.1 ‘inci saniye’de sonrakisi 40.2

inci saniye’de olmuş gibi alınabilir. Bu yöntemin kullanımı ile tüm olayların zaman

atanması (dağıtık bir sistemde) mümkün olur.

(devam)

Senkronizasyon Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Fig. 5.7

33

Şekil 5.7 ‘de,

a) Üç süreç’in her birinin kendi saati var. Saatler farklı oranlarda (hızlarda)

çalışıyor.

b) Lamport algoritması saatleri düzeltmiş olduğu durum görülüyor.

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Dağıtık sistemde aşağıdaki koşullarda tüm olaylara zaman atanması mümkündür.

1-) Eğer a olayı b ‘den önce oluyor ve ikisi aynı süreç içindeyse

C

(

a

)

<

C

(

b

) olur.

34

olur.

2-) Eğer a ve b bir mesajın gönderimi ve alımını temsil ediyorsa, buna göre;

olur.

3-) Tüm ayrı(özel) a ve b olayları için,

olur.

Algoritma, bize tüm olayların (sistemde) toplam bir sıralamasını sağlar.

C

(

a

)

<

C

(

b

)

C

(

a

)

<

C

(

b

)

C

(

a

)

≠

C

(

b

)

Algoritma, bize tüm olayların (sistemde) toplam bir sıralamasını sağlar.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Toplam-sıralı Çoğa gönderim (Totally-ordered Multicasting) :

Lamport zaman damga’larının bir uygulaması olarak, bir veritabanı’nın birkaç

siteye kopyalanmış olması (replika) edildiği durumu göz önüne alalım. Örneğin,

35

sorgu başarımını artırmak için bir banka bir hesap veritabanını iki farklı şehre

kopyalamış olsun. Bir sorgu her zaman en yakın kopyaya iletilir. Hızlı cevabın

(sorgu cevabı) ücreti kısmen yüksek güncelleme maliyetleri ile ödenir, çünkü her

bir güncelleme işlemi her bir kopyada çalıştırılmalıdır.

Örnek ile bakacak olursak:

Bir A şehrindeki müşteri hesabına 100$ eklemek istesin. Şu an hesabında da 1000$

bulunsun. Aynı zamanda B şehrindeki banka çalışanı da bu müşterinin hesabını

%

1 oranında artırsın. İki güncellemede veritabanı’nın her iki kopyasında

yapılmalıdır. Bununla beraber, altta yatan ağ yüzünden oluşan gecikmeler ’den

(iletişimde) dolayı aşağıdaki Şekil 5.8 ‘de gösterilen sırada güncellemeler yerine

varır.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Toplam-sıralı Çoğa gönderim (Totally-ordered Multicasting) :

Fig. 5.8

36

Şekil 5.8 ‘de, bir kopya (replika) veritabanı’nın güncellenmesi ve tutarsız bir

durumda bırakılması şeması.

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Toplam-sıralı Çoğa gönderim (Totally-ordered Multicasting) :

A şehrinde müşterinin güncelleme işlemi, B şehrinden gelen artıştan önce

gerçekleştirilmiştir. Aksine, B şehrindeki kopya hesap kopyası (replikası) ilk önce

37

%

1 artılır ve daha sonra 100$ eklenilir. Sonuçta, A şehrindeki veritabanında

toplam miktar 1111$ olurken, B şehrindeki miktar 1110$ olur. Burada karşılaşılan

problem, iki güncelleme işleminin her bir kopyada aynı sırada

gerçekleştirilmesinin gerekliliğidir.

Hangi iş emrinin önce geldiğine göre oluşan fark, tutarlılık bakış açısıyla

önemlidir. Genelde böyle durumlar, toplam-sıralı çoğa gönderim

(totallyordered multicast) yapılmasını gerektirir. Bu, her bir alıcı için aynı sırada tüm

mesajların teslim edildiği bir çoğa gönderim işlemidir. Lamport zaman damga’ları

toplam-sıralı çoğa gönderim’i tamamen dağıtık biçimde gerçekleştirimi için

kullanılabilir. Bir grup sürecin birbirleriyle mesajları çoğa gönderim (multicast)

yoluyla değiş-tokuş ettiğini varsayalım. Her bir mesaj daima şu anki (mantıksal)

zamana (göndericisinin mantıksal zamanına) ait zaman damgası’nı taşımaktadır.

Bir mesaj çoğa gönderim (multicast) edildiğinde, ana fikir olarak göndericiye de

gönderilir.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Toplam-sıralı Çoğa gönderim (Totally-ordered Multicasting) :

Ek olarak, aynı göndericiden gelen mesajlar, gönderildikleri sırada alınırlar ve

hiçbir mesaj kaybolmaz. Bir süreç, bir mesajı aldığında, onu yerel bir kuyruğa

38

koyar ve zaman damgası’na göre sıralar. Alıcı bir onayı (acknowledgement) diğer

süreçlere çoğa gönderim yolu ile yayar.

Şunu unutmamak gerekir ki, eğer Lamport’un algoritması takip edilerek yerel

saatler ayarlanmışsa, alınan mesajın zaman damgası onaylanan mesajınkinden

daha düşük değerlidir. Bu yaklaşımın en ilginç tarafı, tüm süreçler sonuçta yerel

kuyruklarında aynı kopyaya sahip olacaklardır.

Her bir mesaj, tüm süreçlere çoğa gönderim yapılır, buna onay da dahildir ve tüm

süreçler tarafından alınıldığı varsayılır. Her mesajın gönderildiği sıralama’da

alındığını hatırlayalım. Her bir süreç alınan bir mesajı kendi yerel kuyruğuna

koyar, bunu mesajın zaman damgası’na göre yapar.

(devam)

Senkronizasyon

Mantıksal Saatler (Logical Clocks) :

Lamport Zaman damgaları (Lamport Timestamps):

Toplam-sıralı Çoğa gönderim (Totally-ordered Multicasting) :

Lamport’un saati iki mesajın aynı zaman damgası’na sahip olmadığına emin

olmaktadır, fakat zaman damgaları tutarlı bir global olay sırasını

39

yansıtmaktadırlar. Diğer her bir süreç tarafından onaylanmış ve kuyruğun başında

bulunan bir mesaj olduğunda çalışan uygulama için bir süreç kuyruklanmış bir

mesajı teslim eder. Bu noktada mesaj kuyruktan silinir ve uygulamanın kontrolüne

geçer, ilgili onay basitçe silinir. Her bir sürecin kuyruğun aynı kopyasına sahip

olması yüzünden tüm mesajlar aynı sırada her yerde teslim edilirler. Diğer bir

deyişle, toplam-sıralı çoğa gönderim elde edilmiş olur.

(devam)

Senkronizasyon

Genel Durum (Global State) :

Bir dağıtık sistemin genel durumu

(global state), her bir sürecin yerel

durum’larından (local states) oluşmaktadır. Buna şu an dolaşımda bulunan

mesajlar da dahildir. Bu mesajlar gönderilmiş fakat yerine ulaşmamıştır. Bir

sürecin yerel durumu (local state), onun ne ile ilgilendiğine bağlıdır. Dağıtık

40

veritabanı sistemi de, sadece veritabanı’nın parçası biçimindeki böyle kayıtlardan

oluşabilir ve hesaplama için dışlanmış geçici kayıtlar kullanılır. Örnek olarak

tarama-tabanlı çöp toplama’da, vekil’lerin işaretlerini gösteren değişkenler’den

(iskelet (skeleton) ve nesnelerinde de işaretleri olabilir) oluşabilir ve bir sürecin

adres uzayında bulunabilir.

Bir dağıtık sistemin genel durumu’unu bilmek bazı nedenlerden dolayı kullanışlı

olabilir. Örneğin, yerel hesaplamanın durduğu bilindiğinde ve dolaşan hiçbir

mesaj olmadığında, sistem hiçbir sürecin yapılamayacağı bir duruma girer. Böyle

bir genel durumu analiz edebilmek için, ölümcül kilitlenme

(deadlock) olması

sonucuna varılır veya bir dağıtık hesaplama düzgünce sonlandırılmıştır sonucuna

varılır. Böyle bir analizin nasıl yapıldığına bakılacak olursa, bir dağıtık sistemin

genel durumu’nun en basit ve dosdoğru yoldan kaydı’na dağıtık anlık görüntü

(distributed snapshot) denilir.

(devam)

Senkronizasyon

Genel Durum (Global State) :

Dağıtık bir anlık görüntü (distributed snapshot), dağıtık sistemin bulunduğu

durumu yansıtır. En önemli özelliği, böyle bir anlık görüntü’nün tutarlı bir genel

durumu yansıtmasıdır. Özellikle, başka bir

Q sürecinde bir mesajı alan P sürecini

kaydedersek, aslında bu mesajı gönderen

Q sürecini kaydetmek zorunda oluruz.

Diğer taraftan, anlık görüntü alınan (mesajların) fakat asla gönderilmeyen

41

mesajların kaydını içerebilir, biz bunu istememekteyiz. Bunun tersi bir duruma da

(

Q bir mesajı gönderir ve

P henüz bu mesajı almamıştır) izin verilebilir.

Bir genel durum (global state) grafik olarak bir kesim (cut) diye adlandırılan

biçimde gösterilebilir. Tutarlı bir kesim, birden fazla sürecin her biri için en son

olayın kaydını gösterir. Bu durumda kolayca tüm kayıtlı mesaj alıcılarının ilgili bir

kayıtlı gönderim olayına sahip olduğu kanıtlanabilir. Tutarsız bir kesim de ise,

anlık görüntü hiç bir ilgili gönderim olayı içermez.

Algoritmayı daha basitçe açılayabilmek için (dağıtık bir anlık görüntü alınması

algoritması), dağıtık bir sistemin süreçler topluluğu olarak temsil edildiği ve bu

süreçlerin birbirleriyle çok yönlü noktadan noktaya iletişim kanalları ile bağlı

olduğunu varsayalım. Herhangi bir süreç algoritmayı başlatabilir. Başlatan süreç

(P süreci olsun), kendi yerel durumunu kaydederek başlar, daha sonra gidiş

kanallarının (kendisine ait) her biri boyunca bir işaretlemeci (marker) gönderir.

(devam)

Senkronizasyon

Genel Durum (Global State) :

Bu sayede, alıcının genel durumu kaydında katılımı zorunlu olduğu ortaya çıkar.

Bir

Q süreci, bir geliş kanalından (

C kanalı olsun) bir işaretlemeci (marker)

aldığında, kendi yerel durumunu kaydedip kaydetmemesine bağlı bir hareket

sergiler. Daha henüz bu yerel kaydı bitirmemişse (already done), ilk önce kendi

yerel durumunu kaydeder ve kendinin gidiş kanallarının her birinden bir

42

işaretlemeci (marker) yollar.

Eğer

Q kendi durumunu hali hazırda kaydetmişse, C kanalındaki işaretlemeci

(marker),

Q ‘nun kanalın durumunu kaydetmek zorunda olduğu bir belirteç

(indicator) durumundadır. Bu durum (state),

Q tarafından işaretlemeci (marker)

alınmadan önce, kendi yerel durumunun kaydedildiği en son zamandan beri

alınan mesaj dizisi biçimindedir.

(devam)

Senkronizasyon

Genel Durum (Global State) :

43

Fig. 5.9

Şekil 5.9 ‘da görülenler;

a) Tutarlı kesim(consistent cut)

b) Tutarsız kesim (incosistent cut)

Burada

m harfli olanlar

mesajları,

p harfli olanlar

süreçleri göstermektedir.

Senkronizasyon

Genel Durum (Global State) :

44

Fig 5.10

Şekil 5.10 ‘da görülenler;

a) Dağıtık bir anlık görüntü için süreç ve kanalların bir organizasyonu

Senkronizasyon

Genel Durum (Global State) :

45

Fig 5.10

Şekil 5.10 ‘da görülenler;

b) Q süreci ilk kez bir işaretlemeci (marker) alır ve kendi yerel durumunu kaydeder.

c) Q , gelen tüm mesajları kaydeder.

d) Q , geliş kanalları (kendi kanalları) için bir işaretlemeci (marker) alır ve geliş

kanalının durum kaydını bitirir.

Senkronizasyon

Genel Durum (Global State) :

Bir süreç, algoritmada kendine ait parçayı bitirdiği, kendi geliş kanallarının her

birinden bir işaretlemeci (marker) olduğunda söylenebilir. Bunların her birini

işlemiştir. Bir noktada, yerel durumunu kaydeder ve olabildiğince her bir geliş

kanalı için durumu kaydeder, bunlar toplanılabilir ve gönderilebilir, örneğin anlık

görüntü’yü başlatan süreç gibi. Bu ikincisi, daha sonra şu anki durumu analiz eder.

46

Bu sırada dağıtık sistem normal olarak çalışmaya devam eder.

Örnek: Sonlandırma Tespiti (Termination Detection) :

Dağıtık bir hesaplamanın sonlandırılmasının tespitinde anlık görüntü alınması bir

uygulama’dır. Eğer bir

Q süreci bir anlık görüntü’yü ilk kez isteyen bir işaretlemeci

(marker) aldığında, işaretlemeci ’yi (marker) yollayan süreci kendinden önceki

(parent) yani ata olarak varsayar.

Q anlık görüntü’de kendine ait parçayı tamamladığında kendine öncekine

(atasına) bir DONE mesajı yollar. Özyineleme ile, bir dağıtık anlık görüntü’yü

başlatan bir DONE mesajı aldığında (tüm sonrakilerinden (child) yani

çocukları’ndan) bu anlık görüntü’nün tamamen alındığını (bitirildiğini) anlar.

(devam)

Senkronizasyon

Genel Durum (Global State) :

Örnek: Sonlandırma Tespiti (Termination Detection) :

Bununla beraber, bir anlık görüntü mesajların hala dolaştığı bir genel durum’u

gösterebilir. Özellikle, bir süreç kaydının, kendi yerel durumunu kaydettiği nokta

ile kanaldan işaretlemeci ’yi (marker) aldığı nokta arasında kendi geliş

47

kanallarının birinden aldığı mesajlara sahip olduğunu varsayalım. Buna dağıtık

hesaplamanın tamamlanmış olduğunu katmazsak, bu mesajlar anlık görüntü’nün

parçası olmayan diğer mesajları oluşturmuş olabilir.

Bir anlık görüntü ‘de tüm kanalların boş olması istenir. Algoritmaya basit bir

değişiklik uygulanmıştır, bu aşağıda görülebilir. Bir

Q süreci anlık görüntü’de

kendi parçasını bitirdiğinde, kendinden öncekisine bir DONE mesajı geri

döndürür veya bir CONTINUE mesajı geri döndürür. Bir DONE mesajı sadece

aşağıdaki iki koşul karşılandığında geri döndürülür.

1.) Tüm

Q sürecinin sonrakileri (çocukları) bir DONE mesajı döndürür.

2.)

Q, kendi durumunu kaydettiği nokta ile kendi geliş kanallarının her birinden

işaretlemeci (marker) aldığı nokta arasında herhangi bir mesaj almamıştır.

Diğer tüm durumlarda

Q bir CONTINUE mesajını kendinden öncekine (atasına)

gönderir.

(devam)

Senkronizasyon

Genel Durum (Global State) :

Örnek: Sonlandırma Tespiti (Termination Detection) :

Sonuçta, anlık görüntü’nün orijinal başlatıcısı (

P süreci olsun) ya bir CONTINUE

mesajı alır veya sadece kendinden sonrakide (çocuğu) bir DONE mesajı alır.

Sadece DONE mesajı alındığında, hiçbir düzgün (sıradan) mesajın dolanım’da

48

olmadığı bilinir ve bu hesaplamanın bitirilmesidir (terminated). Aksi halde,

P

süreci başka bir anlık görüntü ’yü başlatır ve sadece DONE mesajları gelene kadar

(geri dönene kadar) buna devam eder. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

49

Ders’in sonu…

50

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 13

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 13: Seçim algoritmaları (Election Algorithms),

Karşılıklı dışlama (Mutual Exclusion),

Dağıtık işler (Distributed Transactions).

Seçim algoritmaları (Election Algorithms)

• Bir çok dağıtık algoritma, bir sürecin koordinatör olarak davranmasını veya özel

bir rol gerçekleştirmesini gerektirir. Genelde, hangi sürecin bu özel sorumluluğu

aldığının önemi yoktur, fakat bir tanesi bunu yapmalıdır. Seçim algoritmaları

koordinatör seçiminde kullanılır. Koordinatör kelimesi böyle özel bir süreci

tanımlamada kullanılan jenerik bir isimdir.

3

• Eğer tüm süreçler tamamen aynıysa, içlerinden birini özel olarak seçmenin

imkanı yoktur. Genelde, seçim algoritması en yüksek süreç numarasına sahip

süreci bulmaya ve onu koordinatör olarak atamaya çalışır. Bunun yanı sıra, her

bir sürecin tüm süreçlerin süreç numaralarını da bildiğini varsayarız.

• Süreçler, hangi sürecin şu an çalıştığını (canlı olduğunu) veya çalışmadığını (ölü

olduğunu) bilmemektedirler. Seçim algoritması’nın asıl amacı, bir seçim

başladığında tüm süreçlerin yeni koordinatör’ün kim olacağı konusunda

anlaşmalarını içerir.

(devam)

Seçim algoritmaları (Election Algorithms)

Bully algoritması (The Bully Algorithm) :

Bully kelimesi İngiliz dilinde anlam olarak kabadayı veya zorba anlamına

gelmektedir.

Bully algoritması’na göre; herhangi bir süreç “koordinatör’ün artık isteklere cevap

4

vermediğini” bildirir ise, bir seçim başlatılır. Bir seçimi elinde bulunduran P

süreci için:

1.)

P bir seçim mesajını yüksek numaralı tüm süreçlere gönderir.

2.) Eğer hiç cevap gelmezse

P seçimi kazanarak koordinatör olur.

3.) Daha yüksek biri cevaplarsa, koordinatörlüğü o alır. P ‘nin görevi bitmiş olur

(done).

Herhangi bir anda, bir süreç bir ELECTION mesajını daha düşük numaralı iş

arkadaşlarından birisinden alabilir. Böyle bir mesaj yerine vardığında, alıcı bir OK

mesajını gönderene, geri gönderir, bu onun canlı olduğunu ve idareyi ele alacağını

gösterir. Alıcı daha sonra bir seçimi tutar, hali hazırda bir tanesini tutmaksızın

bunu yapar. Sonuçta tüm süreçler biri haricinde bundan vazgeçer, bu “biri” yeni

kazanandır. Daha sonra tüm süreçlere bir mesaj ile zaferini ilan ederek yeni

koordinatörlük görevine başlar.

(devam)

Seçim algoritmaları (Election Algorithms)

Bully algoritması (The Bully Algorithm) :

Eğer yüksek numaralı bir süreç önceden çalışmazken çalışır hale gelirse böylece

bir seçimi elinde tutar. Şu an çalışan en yüksek numaralı süreçten daha yüksek

numaralıysa koordinatörlüğü kazanır ve yeni koordinatör olur. Bu tarz, en

güçlünün kazanmasından dolayı “bully” yani kabadayı/zorba algoritması

5

denilmiştir.

En son seçimin sonunda kazana süreç diğerlerine kendisinin koordinatör

olduğunu belirten COORDINATOR mesajını gönderir.

(devam)

Seçim algoritmaları (Election Algorithms)

Bully algoritması (The Bully Algorithm) :

Fig. 5.11

6

Şekil 5.11 ‘de Bully algoritması için gösterilenler;

a) Süreç

4 bir seçimi tutar. b) Süreç 5 ve Süreç

6 cevap verir ve Süreç

4’e

durmasını söylerler. c) sonra Süreç 5 ve Süreç

6 ‘nın her biri bir seçimi tutar.

Seçim algoritmaları (Election Algorithms)

Bully algoritması (The Bully Algorithm) :

Fig. 5.11

7

Şekil 5.11 ‘de Bully algoritması için gösterilenler;

d) Süreç

6, Süreç

5 ‘e durmasını söyler. e) Süreç

6 koordinatörlüğü (seçimi) kazanır

ve herkese bunu söyler (COORDINATOR mesajı ile).

Seçim algoritmaları (Election Algorithms)

Halka Algoritması (A Ring Algorithm) :

Bir Halka (Ring) kullanmaya dayalı bir seçim algoritması da bulunur. Diğer halka

algoritmalarının aksine bu bir jeton (token) bulundurmaz. Burada her bir sürecin

kendinden sonraki süreci bildiğini, fiziksel ve mantıksal olarak sıralı olduklarını

varsayalım.

8

Herhangi bir sürecin koordinatör’ün çalışmadığını bildirmesiyle, kendi süreç

numarasını içeren bir ELECTION mesajı oluşturur ve kendinden sonra gelene

mesajı gönderir.

Eğer kendinden sonraki çalışmıyorsa, bu sonraki süreci atlar ve bir sonrakine

(halkanın bir diğer üyesi) veya daha sonrakine geçer, bir çalışan süreç bulana

kadar ilerler. Her bir adımda, gönderici kendi süreç numarasını mesajdaki listeye

ekler, böylece kendini etkin biçimde bir koordinatör olarak seçilebilecek bir aday

haline getirir. Sonuçta, her şeyi başlatan sürece mesaj geri döner.

(devam)

Seçim algoritmaları (Election Algorithms)

Halka Algoritması (A Ring Algorithm) :

Süreç (başlatan) bu olayın kendi süreç numarasını içeren bir mesaj aldığında

farkına varır. Bu noktada, mesaj tipi COORDINATOR olarak değişir ve tekrar tüm

halkayı dolaşarak koordinatörün kim olduğunu herkese bildirir (yani en yüksek

numaraya sahip liste üyesini).

9

Bu mesaj bir kez dolaştıktan sonra, silinecek herkes kendi işine geri döner.

Sonuçta eğer iki süreç mesajı COORDINATOR tipi mesaja çevirip gönderirse

(aynı üyeler ve aynı sırada), her iki mesajda yok edilir, sadece biraz bantgenişliği

harcanır, zararlı olmaz.

(devam)

Seçim algoritmaları (Election Algorithms)

Halka Algoritması (A Ring Algorithm) :

Fig. 5.12

10

Şekil 5.12 ‘de, Halka (Ring) kullanan seçim algoritması şematik oalrak

gösterilmektedir.

Karşılıklı dışlama (Mutual Exclusion)

• Birden çok süreç içeren sistemler sıklıkla basitçe kritik bölgeler kullanılarak

programlanabilirler. Bir süreç paylaşımlı veri yapılarını okuduğu veya

güncellediğinde, ilk önce karşılıklı birbirini dışlamayı başarmak için kritik bir

bölgeye girer ve aynı zamanda (anda) paylaşımlı veri yapılarını başka hiçbir

sürecin kullanmadığından emin olur. Tek işlemcili sistemlerde kritik bölgeler

11

semafor’lar, monitör’ler ve benzeri yapılar ile korunur.

Merkezi Algoritma (A Centralized Algorithm) :

En dosdoğru şekilde, karşılıklı dışlamayı dağıtık bir sistemde başarmanın yolu, bu

işin tek işlemcili sistemde nasıl yapıldığını simüle etmektir (benzetmektir). Bir

süreç koordinatör olarak seçilir (makinede çalışan en yüksek ağ adresine sahip

biri). Bir süreç kritik bir bölgeye girmek istediği sırada, koordinatöre girmek

istediği kritik bölgeyi belirten ve izin almak için sormak amacıyla bir istek

mesajını gönderir. Başka hiçbir süreç bu kritik bölgede değilse, koordinatör izni

onaylayan bir cevabı isteyene geri yollar. Cevap yerine ulaştığında, istek yapan

süreç kritik bölgeye girer.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Merkezi Algoritma (A Centralized Algorithm) :

Koordinatör eğer bir kritik bölgede farklı bir sürecin hali hazırda bulunduğunu

biliyorsa, istek yapan sürece izin vermez. Sisteme bağımlı olan, red etmek de

kesin bir yöntemdir. Alternatif olarak “permission denied” (izin isteği red edildi)

biçiminde cevap geri döndürülür.

12

Red edilen süreç bloklanır (bekler) ve bu süre içerisinde (beklerken) koordinatör

bu istek yapan süreçten gelen istekleri kuyruğa alır. Şu an kritik bölgede olan

süreç buradan çıkınca, koordinatöre bir mesaj atarak kritik bölgeden çıktığını

belirtir (dışlayan erişimi serbest bırakır).

Koordinatör kuyruktaki ilk maddeyi alır ve (istek yapan sürecin geciktirilmiş

isteği) bu isteği yapana onay mesajı yollar. Eğer istek yapan hala bloklu ise,

bloklanması biterek kritik bölgeye girer. İzin isteği reddi (permission denied)

almışsa, bu süreç gelen trafik için bakınır veya daha sonra bloklanır. Her şekilde,

onayı gördüğünde kritik bölgeye girer.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Merkezi Algoritma (A Centralized Algorithm) :

Bu şekilde, bu algoritma koordinatör’ün bir seferde bir kritik bölgede bir sürecin

bulunmasını sağlamasıyla karşılıklı dışlayan erişimi garantilediği görülür.

İsteklerin alındığı sıralama ile onaylanması açıkça anlaşılır. Hiçbir süreç sonsuza

kadar beklemez (no starvation = sonsuza kadar bekleme yok anlamına gelir). Bu

13

şema kolayca gerçekleştirilir, sadece üç mesaj her kritik bölge kullanımında

gereklidir (istek, onay, serbest bırakma) (bunların İngiliz dilindeki

karşılıkları: request, grant, release olarak verilmektedir).

Sadece kritik bölge yönetiminden ziyade genel kaynak ayrımında da bu şema

kullanılabilir. Bu merkezi yaklaşımın bazı eksikleri vardır. Buradaki koordinatör

tek hata noktasıdır, o çökerse, tüm sistem de çöker. Eğer süreçler normal olarak

bir istek yapıldıktan sonra bloklanır ise, “permission denied” dan dolayı ölü bir

koordinatörü ayırt edemez, çünkü her iki durumda da hiç mesaj geri gelmez.

Ayrıca büyük sistemlerde, tek bir koordinatör bir başarım (performance)

darboğazı oluşturur.

(devam)

Merkezi Algoritma (A Centralized Algorithm) :

Karşılıklı dışlama (Mutual Exclusion)

Fig. 5.13

14

Şekil 5.13 ‘te görülenler; a) Süreç

1, koordinatörden kritik bir bölgeye girmek için izin ister. İzin

verilir (Permission granted). b) Süreç 2, daha sonra aynı kritik bölgeye girmek için koorindatörden

izin ister. Koordinatör cevap vermez. c) Süreç 1 kritik bölgeden çıktığında, çıktığını koordinatöre bildirir,

koordinatör bunun üzerine Süreç 2 ‘ye cevap verir.

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Tek bir hata noktasına sahip olmak (merkezi yaklaşımdaki gibi) sıklıkla kabul

edilemez bulunur. Sistemdeki tüm olayları sıralamak dağıtık algoritma için

başlangıç fikri olmuştur. Bunu başarmanın bir yolu, dağıtık karşılıklı dışlayan

erişim için zaman damga’ları sağlanmasıdır. Algoritma şöyle çalışır:

15

Bir süreç kritik bir bölgeye girmek istediğinde, girmek istediği bölgenin ismini

içeren bir mesaj oluşturur, bu mesajda bu sürecin süreç numarası ve şu anki

zaman değeri de bulunur (yani

3 adet bilgi içeriyor). Daha sonra, tüm diğer

süreçlere bu mesajı gönderir (konsept olarak kendisine de gönderir).

Mesajın gönderimi güvenli varsayılır, her bir mesaj onaylanmıştır

(acknowledged). Eğer güvenli iletişim grubu varsa, kişisel mesajların yerine

kullanılabilir. Bir süreç başka bir süreçten bir istek mesajı alırsa mesajda

belirtilen isimdeki kritik bölgeye göre onun durumuna bağlı hareket edilir.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Ayrım yapılması (anlaşılması gereken)

3 durum bulunur:

1.) Eğer alıcı kritik bölgede değilse, bu kritik bölgeye girmek istemiyorsa, geriye bir

OK mesajını göndericiye gönderir.

2.) Eğer alıcı hali hazırda kritik bölgedeyse, cevap vermez. Bunun yerine, isteği

16

kuyruklar.

3.) Eğer alıcı kritik bölgeye girmek istiyorsa ve henüz yapmadıysa, herkese

gönderdiği bir mesajda içerilen biri ile gelen mesajdaki zaman damgası’nı

karşılaştırır. En düşük olan kazanır. Eğer gelen mesaj daha düşük ise, alıcı

geriye bir OK mesajı yollar. Eğer kendi sahip olduğu mesaj daha düşük bir

zaman damgası’na sahipse, alıcı gelen isteği kuyruklar ve hiçbir şey göndermez

(Kuyruklama = alınan mesajı kendi kuyruğuna (alıcının kuyruğu)

yerleştirmesi).

• İsteğin gönderiminden sonra, kritik bir bölgeye giriş izni için sorulmasında, bir

süreç geri durup verilen izinleri herkesin almasına kadar bekler. En kısa sürede

tüm izinler alınınca, bu süreç kritik bölgeye girer. Kritik bölgeden çıkarken,

kuyruğundaki tüm süreçlere OK mesajı yollar ve tümünü kuyruktan siler.

(devam)

Dağıtık Algoritma (A Distributed Algorithm) :

Karşılıklı dışlama (Mutual Exclusion)

Fig. 5.14

17

Şekil 5.14 ‘te görülenler; a) Aynı anda aynı kritik bölgeye girmek isteyen iki süreç. b) Süreç

0, en düşük zaman damgasına sahiptir, böylece o kazanır. c) Süreç

0 ‘ın işi bitince (done), bir OK mesajı gönderir, böylece Süreç

2

kritik bölgeye girer.

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Merkezi algoritmada, karşılıklı dışlama ölümcül kilitlenme

(deadlock) ve

sonsuza kadar bekleme (starvation) olmadan garanti edilmektedir. Her bir girdi

için gerekli mesaj sayısı şimdi 2(n-1) tanedir, burada n, sistemdeki toplam süreç

sayısıdır.

18

Bunun en iyi yanı, tek bir hata noktasının olmayışıdır. Ne yazık ki, tek bir hata

noktasının,

n tane hata noktasıyla yer değiştirmiş olmasıdır. Eğer herhangi bir

süreç çökerse, isteklere cevap vermede çöker. Bu sessizlik, yanlış olarak izin reddi

(denial of permission) şeklinde yorumlanılır, böylece daha sonraki, tüm kritik

bölgelere girmeye çalışan tüm süreçlerin denemeleri de bloklanılır.

Çünkü

n süreçten birinin çökme (failing) (ya da hata yapma) olasılığı tek bir

koordinatörün çökmesinden (failing) en azından

n kez büyüktür. Bu algoritmayı

değiştirip en kötü n kez’den daha iyi hale getirmek mümkündür, boot etmek için

daha fazla ağ trafiği gerektirir.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Algoritmayı bazı püf noktalar ile yamalamak olasıdır. Bir istek geldiğinde, alıcı

daima bir cevap gönderir, ya onaylamak (granting) ya da izni red etmek için

(denying permission) bunu yapar. Bir istek ya da bir cevap kaybolduğunda,

göndericinin zamanı dolar ve denemeye, ya bir cevap geri gelene veya gönderici,

19

hedefin öldüğünün sonucuna varıncaya kadar devam eder.

Bir istek red edildikten sonra, gönderici bir sonraki (sıradaki) OK mesajını

beklemek için bloklanmak zorundadır. Başka bir problem, her bir sürecin grup

üyelik listesini kendinin tuttuğu bir grup iletişimi’nin bu algoritma ile

yapılmasıdır. Yöntem en iyi küçük boyutlu süreç grupları ile (grup üyelikleri asla

değişmeyen süreçler ile) yapılmasıdır.

Sonuçta, merkezi algoritma’nın problemlerinden birisi de, tüm istekleri

yönetmenin bir darboğaz oluşturmasıdır. Dağıtık algoritmada, tüm süreçler kritik

bölgelere giriş ile ilgili tüm kararlar da içerilirler. Eğer bir süreç yükü idare

edemiyorsa, herkesi tam olarak aynı şeyi paralel olarak yapması için çalıştırması

ile daha yardımcı olur.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Algoritma modifiye edilerek, bir sürecin diğer süreçlerin küçük boyutlu (basit)

çoğunluğunun izinlerini alarak kritik bir bölgeye girmesi sağlanabilir, böylece tüm

süreçlerin (hepsinin) iznini almasına gerek kalmaz.

20

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Dağıtık sistemde karşılıklı dışlama için tamamen farklı bir yaklaşım için bu

algoritma denenmiştir. Bir veriyolu

(bus) ağımızın olduğunu ve kalıcı olmayan

süreç sıralamamız olduğunu varsayalım. Yazılımda, bir mantıksal halka’da her bir

süreç halka’da bir pozisyona atanmıştır.

21

Halka pozisyonları ağ adreslerinin nümerik sırasında veya diğer bazı anlamlarda

ayrılmış olabilir. Sıralamanın neye göre yapıldığının önemi yoktur. Tüm

sıralamalarda, her bir süreç kimin kendinden sonra bu hat üzerinde olduğunu

bilir.

Halka

(Ring) başlatıldığında, Süreç

0, verilmiş jeton (token) olur. Jeton halka

boyunca dolanır. Süreç

k ‘dan süreç k+1 ‘e geçer (Ring büyüklüğü’nün

matematiksel modülosu), burada noktadan noktaya mesajlaşma vardır. Bir süreç

komşusu’ndan jetonu ele geçirince, kritik bir bölgeye girmeyi deneyip

denemeyeceğine bakar. Eğer öyleyse (girebiliyorsa), süreç bölgeye girer, gerekli

olan işini yapar ve bölgeyi terk eder. Bu süreç çıktıktan sonra, halka boyunca

jetonu gönderir. Aynı jeton kullanılarak ikinci bir kritik bölgeye girmeye izin

verilmez.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Eğer bir süreç , komşusu’ndan jetonu alır ve bir kritik bölgeye girmekle

ilgilenmiyorsa, sadece jetonu sonrakine iletir. Sonuçta, hiçbir süreç herhangi bir

kritik bölgeye girmekle ilgilenmiyorsa jeton halka boyunca yüksek hızla

dolanımına devam eder.

22

Bu algoritmanın doğruluğu açıkça görülür. Herhangi bir anda jetona sahip sadece

bir süreç olur ve sadece bir süreç (bu süreç) gerçekte kritik bölgede bulunabilir.

Çünkü jeton süreçler arsında iyi tanımlanmış bir sırada dolanır ve sonsuza kadar

bekleme (starvation) gerçekleşmez. Bir kez bir süreç bir kritik bölgeye girmek

isterse, en kötü ihtimalle diğer her sürecin bir kritik bölgeye girmesini veya

çıkmasını bekler.

Eğer jeton kaybolursa tekrar oluşturulması gerekir. Kaybın tespiti çok zordur,

keza jetonun ağ’da arka arkaya görünmesi arasındaki zaman miktarı sınırsız

(unbounded) olur. Bundan dolayı, bir saattir gözlemlenemeyen jeton için bu

durumun anlamı, onun kayıp olduğu veya birisinin hala onu kullandığıdır.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

Süreç çöktüğünde kurtarmak görece basittir. Bir süreç jetonu alınca onay

(acknowledge) alması ile bir ölü süreç komşusu ona bir jeton vermeye çalıştığında

tespit edilebilir. Bu noktada ölü süreç gruptan silinir. Ölü süreçten sonraki sürece

jeton devredilerek, jeton dolanımı devam eder. Bu yüzden, üye herkesin halka

23

konfigürasyonunu bilerek, ona bakması gerekir.

(devam)

Karşılıklı dışlama (Mutual Exclusion)

Dağıtık Algoritma (A Distributed Algorithm) :

24

Şekil 5.15 ’te görülenler;

a) Bir ağ üzerinde sıralamasız süreç grubu. b)Yazılımda oluşturulmuş mantıksal halka.

Dağıtık işler (Distributed Transactions)

Karşılıklı dışlama ile ilgili bir konuda transaction ‘dır. Bunu Türkçe olarak iş

veya işlem olarak ifade edebiliriz. Karşılıklı dışlama algoritması paylaşımlı bir

kaynağa (dosya, printer gibi) bir seferde en fazla bir süreç tarafından

erişildiğinden emin olunmasını sağlar.

25

İşlemler (transactions), genelde eşzamanlı birkaç sürecin eşzamanlı erişim

karşısında paylaşımlı bir kaynağın korunmasını sağlarlar. Özellikle, işlemler

(transactions) paylaşımlı verinin korunmasında kullanılırlar. Bununla beraber,

işlemler daha fazlasını da yapabilir.

Özellikle, bir sürecin çoklu verilere tek bir atomik işlem olarak erişebilmesine ve

onları değiştirebilmesine olanak sunarlar. Eğer süreç, işlem işlerken işi yarıda

bırakırsa, her şey işlem başlamadan önceki duruma geri döndürülür.

(devam)

Dağıtık işler (Distributed Transactions)

İşlem Modeli (The Transaction Model) :

Bir süreç bir işlem’e bir veya daha çok süreç ile başlamak istediğini duyurur.

Bunlar çeşitli seçenekler, varlık oluşturulması ve silinmesi, bir süre bir işleme’nin

(operation) yapılmasını görüşürler. Daha sonra işlemi başlatan (initiator) tüm

diğerlerinin işi sonuna kadar yapacaklarını commit etmelerini (işi üstelenerek

26

onaylama) istediğini onlara duyurur.

Eğer tümü anlaşırlarsa, sonuçlar kalıcı hale getirilir. Eğer bir veya daha fazla

süreç red ederse (veya anlaşmadan önce çökerlerse) durum işlem başlamadan

önceki durum neyse tam olarak ona döner, buna tüm dosyalardaki yan etkiler,

veritabanları vs. dahildir. Bu hepsi-veya-hiçbiri özelliği, programcının işini

kolaylaştırır.

Eski bilgisayarlarda (1960’lı yıllar), gün sonunda veriler veri teyplerine

depolanıyordu. Bunun güzel tarafı, herhangi bir nedenle çalışmada hata oluşursa

(sorun olursa), tüm teypler geri sarılarak, iş tekrar hiçbir zarar olmadan baştan

başlatılabiliyor olmasıydı. Temel olarak bu sistem, işlem’lerin hepsi-veya-hiçbiri

özelliğine benzemektedir.

(devam)

Dağıtık işler (Distributed Transactions)

İşlem Modeli (The Transaction Model) :

Bunun modern bankacılıktaki uygulaması online veritabanlarının

güncellenmesidir. Bir müşteri, modem’e sahip bir PC kullanarak bir hesaptaki

parasını, diğer hesaba aktarmak istediğinde işleme/çalışma (operation) iki

adımda gerçekleşir:

27

1-) Hesap no

1 ‘den a miktarında para çekilir.

2-) Hesap no

2 ‘ye a miktarında para yatırılır.

Bu sırada, eğer birinci adımda telefon hattı koparsa, bu para yerine

ulaşamayacağı için kaybolmuş olur. Bu problemi çözebilecek, bu iki işleme/

çalışma (operation) tek bir grup haline getirebilecek bir işlem (transaction)

oluşturulabilir. Böylece bu iki iş adımı ya beraberce tamamlanacaktır veya

beraberce tamamlanamayacaktır. Buradaki ana konu eğer işlem (transaction)

tamamlanamazsa ilk duruma geri dönüleceğidir. Veritabanını, aynı teyplerdeki

gibi geri sarmak, işlem’in (transaction ‘ın) önerdiği bir yetenektir.

İşlemler (transaction’lar) kullanılarak programlama yapmak, altta yatan dağıtık

sistem veya çalışma zamanı sistemi (dile ait) tarafından desteklenmelidir

(ilkellerle).

(devam)

Dağıtık işler (Distributed Transactions)

İşlem Modeli (The Transaction Model) :

Tipik işlem ilkelleri

(transaction primitives) örnekleri aşağıdaki tablo’da

verilmektedir.

İlkel (Primitive

) Tanım (Description

)

Fig. 5.18

28

İlkel (Primitive

) Tanım (Description

)

BEGIN\_TRANSACTION Bir i

şlem’e ba

şlanılması (Make the start of a

transaction

)

END\_TRANSACTION

İşlem’in bitirilmesi ve i

şin üstlenilerek onaylama’nın

denenmesi (Terminate the transaction and try to

commit

)

ABORT\_TRANSACTION İşlem’in öldürülmesi ve eski de

ğerlerin geri al (Kill the

transaction and restore the old values

) ABORT\_TRANSACTION

READ

Verilerin bir dosyadan, bir tablodan veya ba

şka bir

yerden okunması (Read data from a file, a table, or

otherwise

)

WRITE Verilerin bir dosyada, bir tabloya veya ba

şka bir yere

yazılması (Write data to a file, a table, or otherwise

)

Dağıtık işler (Distributed Transactions)

İşlem Modeli (The Transaction Model) :

BEGIN\_TRANSACTION ve END\_TRANSACTION bir işlem’in (transaction ‘ın)

kapsamını (scope) sınırlar. İşlem gövdesini bu ikisi arasındaki işleme’ler (operations)

biçimlendirir. Yine bu işlemlerin ya tümü ya da hiçbiri çalıştırılmaz. Bunlar, sistem

çağrıları, kütüphane prosedürleri veya gerçekleştirime bağlı olarak ayraçlarla ayrılmış

ifadeler (programlama diline göre) olabilmektedirler.

29

Aşağıdaki örnekte, bir havayolu ile aktarmalı olarak WP = White Plains ‘den Nairobi

şehrine ve oradan da Malindi isimli şehre gidilmek istenilsin. (JFK = John F. Kennedy

havaalanı).

BEGIN\_TRANSACTION

reserve WP -> JFK;

reserve JFK -> Nairobi;

reserve Nairobi

-> Malindi;

BEGIN\_TRANSACTION

reserve WP -> JFK;

reserve JFK -> Nairobi;

reserve Nairobi

-> Malindi full =>

A ) şıkkında; başarılı olunmuş üç uçuş da commit edilebilir. Böylece işlem sonlandırılır.

B) şıkkında; son uçuş commit edilemez (zaten dolu), bu yüzden işlem abort edilir

(Başarısız olunmuştur).

reserve Nairobi

-> Malindi;

END\_TRANSACTION

(a)

reserve Nairobi

-> Malindi full =>

ABORT\_TRANSACTION

(b)

Dağıtık işler (Distributed Transactions)

İşlem Modeli (The Transaction Model) :

İşlem’lerin hepsi-veya-hiçbiri özelliği, ACID kelimesinin baş harfleri ile

belirtilen

4 adet özellikle belirtilir: (İşlem’ler aşağıdaki gibi olmalıdır) :

1

-)

Atomic (Atomik)= Dış dünya için, işlem’ler (transactions) bölünemez

30

1

-)

Atomic (Atomik)= Dış dünya için, işlem’ler (transactions) bölünemez

olmalıdır.

2-)

Consistent (Tutarlı) = İşlem’ler (transactions) sistem sabitleri’ni

(invariants) ihlal etmemelidir.

3-)

Isolated (İzole) = Eşzamanlı işlem’ler birbirini etkilemezler.

4-)

Durable (Dayanıklı) = Bir işlem bir kez commit edildiyse (onaylandıysa),

değişiklikler kalıcı olur.

(devam)

Dağıtık işler (Distributed Transactions)

İşlem Modeli (The Transaction Model) :

Atomik özelliği, işlem‘in ya tamamen tamamlanmasını veya hiç yapılmamasını

belirtir, yani eğer oluyorsa tek bir bölünmez bütün olarak anlık hareketle olur ya

da hiç olmaz. Bir iş seyrindeki bir işlem (transaction) için diğer süreçler

(transaction’da içerilsin veya içerilmesinler) hiçbir orta seviye durumu (ara

31

durum) göremezler (yani olayı bir bütün gibi görürler).

Tutarlı (Consistent) terimi ile buradaki kasıt, örneğin banka içinde para

miktarının sabit kalması gerekliliği gibi bir kural varsa, buna göre iç transferlerde

para miktarı değişimi işlem dışından görülebilir olmamalıdır.

İzole (Isolated) terimi ile buradaki kasıt, en son sonuç’un tüm işlem’lerin

(transaction’ların) seri olarak (sıralı) çalıştığının (sisteme bağlı bir sırada)

görülmesidir. Dayanıklılık (Durable) terimi ise, bir kez bir işlem commit edildiyse

(onaylandıysa) yoluna dosdoğru devam edeceği ve ne olursa olsun, sonuçların

değişmeden kalacağını belirtir.

(devam)

Dağıtık işler (Distributed Transactions)

İşlemlerin Sınıflandırılması (Classification of Transactions) :

ACID özelliklerini karşılayan işlem tipleri bir düz işlem

(flat transaciton) olarak

isimlendirilir. Bunlar en basit tipli işlemlerdir ve sıklıkla kullanılırlar. Bunların

bazı belirli kısıtları vardır. Bunlar haricinde iki önemli işlem sınıfı bulunur: iç içe

(nested) işlem’ler ve dağıtık (distributed) işlem’lerdir.

32

Düz işlemlerin bazı kısıtlamaları :

Düz (Flat) işlem’lerin (transactions) ana sınır’ı (kısıt’ı) kısmi sonuçlara,

onaylanma veya abort (iptal) edilmede de izin vermemeleridir. Atomik özelliği’nin

güçlülüğü, aslında bu düz işlem’ler için zayıflıktır. Biraz önceki uçuş örneğinde tek

tek bilet almak yerine tek bilet (aktarmalı) almak daha ucuz olabilir.

Sonuçta, işlem’lerin kısmen onaylanabilmesi de istenebilir. Düz işlem’ler (Flat

transactions) buna izin vermez. Başka bir örnek olarak da bir web sitesindeki bir

sayfanın başka bir yere kopyalandığını varsayalım. Bu durumda bu sayfaya olan

tüm hiper bağlantılar (hyperlinks) tek bir atomik işlemle güncellenmelidir. Aksi

halde geçici olarak bu sayfaya ulaşmak mümkün olmaz. Bu teoride düz işlem’in

(flat transaction) kullanıldığı görülür.

(devam)

Dağıtık işler (Distributed Transactions)

İşlemlerin Sınıflandırılması (Classification of Transactions) :

Düz işlemlerin bazı kısıtlamaları :

Buradaki problem, böyle bir işlem’in saatlerce sürebilecek olmasıdır. Bu sayfayı

gösteren tüm referanslar Internet’e yayılmış olabilir. Her bir güncellemeyi ayrı bir

33

işlem ile yapmak iyi değildir. Olası bir çözüm ise, güncellemeleri onaylamak

(commit) ve fakat bağlantıyı (link) yenilenip güncellenmemiş olanlarda eski

sayfayı saklamaktır.

İç içe işlemler (Nested Transactions) :

Önceki anlatılan sınırlamaları halletmek için iç içe işlemler kullanılabilir. İç içe bir

işlem belirli sayıda alt işlem’lerden oluşur. En üst-seviyedeki işlem kendi

çocuklarını alır ve bu çocuklar birbirleriyle paralel olarak çalışabilir (farklı

makinelerde de başarım kazancı için çalıştırılabilir). Bu çocukların her biri bir

veya daha fazla alt işlem’leri çalıştırabilir veya kendi çocuklarına sahip olabilir. Alt

işlem’ler zorluğu artırır, yani önemli bir problemdir. Bir işlem’in birkaç alt işlem’i

paralel olarak başlattığını varsayalım ve bunlardan biri onaylansın, bunun

sonuçları ata (parent) işlem’ine görünür hale gelir. Daha sonra ileri hesaplamada,

ata (parent) abort olur (iptal edilir). Tüm sistem en üst-seviyedeki işlem’in

başlamasından önceki duruma gelir.

(devam)

Dağıtık işler (Distributed Transactions)

İşlemlerin Sınıflandırılması (Classification of Transactions) :

İç içe işlemler (Nested Transactions) :

Sonuçta, alt işlemlerin sonuçları yinede yapılmadan onaylanmalıdır. Bu sadece en

üst-seviye işlem’lere yukarıda uygulanan sürekliliği gösterir. İşlemlerin keyfi

34

olarak derin biçimde iç içe oluşturulması yüzünden, önemli miktarda yönetim her

şeyin doğru yürütülmesi için gereklidir.

Anlamsallık (semantik) açıktır. Herhangi bir işlem veya alt işlem başladığında,

tüm verinin kavramsal olarak özel bir kopyası tüm sistemde, başlatılan bu işlem’i

istediği gibi yönetmesi için ona verilir. Eğer abort (iptal) olursa, kendi özel evreni

yok olur, sanki hiç var olmamaış gibi olur. Eğer kabul edilirse, kendi özel evreni

ata’sının (parenti’nin) evreni’nin yerini alır.

Böylece eğer bir alt işlem onaylanır ve yeni bir alt işlem daha sonra başlatılırsa,

ikincisi birinci alt işlem’in ürettiği sonuçları görür. Benzer olarak, eğer bir

çevreleyen (daha üst seviye) işlem abort (iptal) olursa, onun altındaki tüm alt

işlemler de abort (iptal) olur.

(devam)

Dağıtık işler (Distributed Transactions)

İşlemlerin Sınıflandırılması (Classification of Transactions) :

İç içe işlemler (Nested Transactions) :

İç içe işlem’ler dağıtık sistemlerde önemlidir, bu yolla bir işlem birden çok

makine arasında doğal olarak dağıtılabilir. Bununla beraber, iç içe işlem’ler

genelde orijinal işlem’in işinin mantıksal bir bütünüdür. Örneğin, işlem’in üç

35

farklı uçuş için ayrılması gerektiği örneğe bakalım, burada üç alt işlem’e

mantıksal olarak bölünebilir.

Bu alt işlem’lerin her biri ayrı ayrı ve diğer ikisinden bağımsız olarak yönetilebilir.

Bununla beraber, bir iç içe işlem’in mantıksal bir bölümü alt işlem’ler içine

oluşturulmasının tüm dağıtımın dikkat etmesi gereksizdir. Yani daha küçük alt

parçalara bölmek mantıksız olabilir. Uçuş örneğindeki koltuk rezervasyon

işlem’ini daha alt işlem’lere bölmek anlamsızdır.

Bu durumda, karşılaşılan olay, bir (düz) alt işlem’in birden çok makine arasında

dağıtılmış veri üzerinde işlem yapmasıdır. Böyle işlem’ler dağıtık işlem’ler olarak

bilinirler.

(devam)

Dağıtık işler (Distributed Transactions)

İşlemlerin Sınıflandırılması (Classification of Transactions) :

İç içe işlemler (Nested Transactions) :

Fig. 5.20

36

Şekil 5.20 ‘de görülenler;

a) Bir iç içe işlem. b) Bir dağıtık işlem.

Dağıtık işler (Distributed Transactions)

İşlemlerin Sınıflandırılması (Classification of Transactions) :

İç içe işlemler (Nested Transactions) :

İç içe işlem’ler ile dağıtık işlem’ler arası fark zor (nadir) olur fakat önemlidir. İç

içe bir işlem hiyerarşik bir yapıda alt işlem’lere mantıksal olarak bölünmüştür.

Aksine, dağıtık bir işlem mantıksal olarak düz, bölümlenemez işlem olur. Bu işlem

37

dağıtık veri üzerinde çalışır.

Dağıtık işlem’lerin ana problemi, ayrı dağıtık algoritmaların verinin kilitlenmesi

idaresi için gerekmesi ve tüm işlem’in onaylanması

(committing) gerekmesidir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

38

Ders’in sonu…

39

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com

Dağıtık Sistemler

Çanakkale

Onsekiz

Mart

Üniversitesi

Bilgisayar Mühendisliği Bölümü

Dağıtık Sistemler

Bahar yarıyılı

Ders notları

HAFTA 14

Dr. Bahadır KARASULU

bahadirkarasulu@comu.edu.tr

2

Hafta 14: Dağıtık işler – Kısım 2.

Dağıtık işler (Distributed Transactions)

Gerçekleştirim (Implementation) :

İşlem’ler kulağa büyük bir fikir gibi geliyorlar, fakat nasıl gerçekleştiriliyorlar? Bu

soru önemlidir. İşlem’ler atomik olmazsa, değişiklikler yok olmakta ve sorun

teşkil edebilmektedir.

Özel çalışma alanı (Private Workspace) :

3

Kavram olarak, bir süreç bir işlem’i başlattığında, işlem’in eriştiği tüm dosyaları

içern özel bir çalışma alanı oluşturulur (ona verilir). Bu işlem onaylanma

(commit) veya abort (iptal) edilene kadar buna ait tüm okuma ve yazmalar

doğrudan dosya sistemine gitmek yerine özel çalışma alanına gider. Bu gözlem

doğrudan ilke gerçekleştirim yöntemini verir: Bir süreç bir işlem’i başlattığında

anında ona özel bir çalışma alanı verilir.

Bu teknikteki problem, bir özel çalışma alanına her şeyi kopyalamanın

maliyetinin engelleyici oluşudur, fakat çeşitli optimizasyonlar onu fizible (uygun)

hale getirir. İlk optimizasyon, bir sürecin bir dosyayı okuduğu fakat onu

değiştirmediğinde özel bir kopyaya ihtiyaç olmamasıdır. Bu sadece gerçek dosyayı

kullanır (transaction başladığından beri değişmeksizin). Sonuçta, bir süreç bir

transaction’ı başlattığında, kendi atası’nın (parenti’nin) çalışma alanına geriye

doğru bir işaretçi haricinde boş olan bir özel çalışma alanı oluşturulması yeterli

olur. (Yani tek bir işaretçi içeren boş bir alan oluşturulur.

(devam)

Dağıtık işler (Distributed Transactions)

Gerçekleştirim (Implementation) :

Özel çalışma alanı (Private Workspace) :

Burada, işlem eğer en süt-seviye işlem ise ata’nın (parent’ın) çalışma alanının

dosya sistemi olduğunu söyleriz. Eğer süreç bir dosyayı okumak için açarsa, geriye

doğru olan işaretçi’ler dosya atası’nın (parenti’nin) çalışma alanında (veya sonraki

ata’nın) bulununcaya kadar takip edilir. Bir dosya yazmak için açılırsa, aynen

4

okumak için açıldığında izlenilen yolla takip edilir, sadece şimdi ilk önce özel

çalışma alanına kopyalanır (okuma yönteminden farklı bir adım olarak).

Bununla beraber, ikinci bir optimizasyon daha fazla kopyalama işlemini de yok

eder. Tüm dosyayı kopyalamak yerine, sadece dosya’nın index’i özel çalışma

alanına kopyalanır. Her bir dosya ile ilişkili veri bloğu olan index, disk bloklarını

nerede olduğunu söyler. UNIX’te disk adresleri orijinal disk bloklarını içerir.

Bununla beraber, bir dosya bloğu ilk önce modifiye edildiğinde, bir bloğun

kopyası oluşturulur ve kopyanın adresi index’e eklenilir (insert). Daha sonra bu

blok orijinal’i etkilemeden güncellenebilir. Blokları eklemek de (append) böyle

yapılabilir. Yeni bloklar bazen gölge bloklar

(shadow blocks) olarak

adlandırılırlar. İlgili işlem’i çalıştıran süreç değişikliğe uğramış dosyayı

görebilirken, tüm diğer süreçler orijinal dosyayı göremeye devam ederler.

(devam)

Dağıtık işler (Distributed Transactions)

Gerçekleştirim (Implementation) :

Özel çalışma alanı (Private Workspace) :

Daha karmaşık işlem’de, sadece bir tane yerine daha büyük (çok) sayıda dosya

özel çalışma alanında içerilebilir. Eğer işlem abort edilirse, özel çalışma alanı

silinir ve işaret ettiği tüm özel bloklar boşlar listesi’ne (free list) geri koyulur. Eğer

işlem onaylanırsa, özel indisler ata’nın (parent’ın) çalışma alanına atomik olarak

5

taşınır. Boşlar listesi’ne konulanlar, artık ulaşılamayan bloklardır.

Bu şema dağıtık işlem’ler için de çalışır. Bu durumda, bir süreç, bir dosya içeren

her bir makine de başlatılıyorsa, bu dosya işlem’in bir parçası olarak erişilebilir

olur. Her bir süreç ‘in kendi özel çalışma alanı bulunur. Eğer işlem abort edilirse,

tüm süreçler kendi özel çalışma alanlarını göz ardı ederler.

Diğer yandan, işlem onaylandığında (commit), güncellemeler yerel olarak

yayımlanılır (yayılır), bu işlem’in tamamen tamamlandığını (bitirildiğini) gösterir.

(devam)

Dağıtık işler (Distributed Transactions)

Gerçekleştirim (Implementation) :

Özel çalışma alanı (Private Workspace) :

Fig. 5.21

6

Şekil 5.21 ‘de görülenler;

a) Bir üç-bloklu dosya için dosya index’i ve disk blokları.

b) Bir işlem’in Blok

0 ‘ı değiştirdiği ve Blok 3 ‘ü eklediğinden sonraki durum.

c) Onaylamadan (committing) sonraki durum.

Dağıtık işler (Distributed Transactions)

Gerçekleştirim (Implementation) :

Önceden-yazma Kaydı (Write-ahead Log) :

Başka bir yerel yöntem de Önceden-yazma Kaydı

(Write-ahead Log)

yöntemidir. Bu yöntemle, dosyalar yerlerinde değişikliğe uğrarlar, fakat herhangi

bir blok değiştirilmeden önce hangi işlem’in değişiklik yaptığını söyleyen bir kayıt

log ’a (kayıt dosyasına) yazılır, buradaki değişiklikten kasıt, dosya ve blok’un

7

değişmiş olması ve eski ve yeni değerlerin ne olduğudur. Log (kayıt) başarıyla

yazıldıktan sonra ancak değişim dosya üzerinde uygulanır.

Eğer işlem başarılı ve onaylanmışsa, bir onay kaydı log dosyasına yazılır, fakat veri

yapıları halihazırda güncellenmiş olarak değişikliğe uğramazlar. Eğer işlem abort

(iptal) olursa, log orijinal duruma geri dönmekte kullanılır. Sondan başlayarak

geri gitmek için, her bir log kaydı okunarak bu kayıtta tanımlı değişimler geri

alınır (değişiklikler yapılmamış hale getirilir). Bu harekete geriye dönme

(rollback) denilir.

Bu şema, dağıtık işlem’ler için de çalışır. Bu durumda, her bir makine kendi yerel

dosya sistemi için kendi değişim log kaydını tutar. Geriye dönmek

(rolling

back) işlemi her bir makine’de ayrı ayrı yapılan geri dönme işlemi ile yapılır.

Böylece her makinede orijinal dosyaya dönüş olur.

(devam)

Dağıtık işler (Distributed Transactions)

Gerçekleştirim (Implementation) :

Önceden-yazma Kaydı (Write-ahead Log) :

x = 0; Log Log Log

Fig. 5.22

8

x = 0;

y = 0;

BEGIN\_TRANSACTION;

x = x + 1;

y = y + 2

x = y \* y;

END\_TRANSACTION;

(a)

Log

[x = 0 / 1]

(b)

Log

[x = 0 / 1]

[y = 0/2]

(c)

Log

[x = 0 / 1]

[y = 0/2]

[x = 1/4]

(d)

Şekil 5.22 ‘de görülenler;

a) Bir işlem, b) ve c) ve d) şıkları, her bir ifade çalıştırılmadan önceki log kaydı.

(a) (b) (c) (d)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

İşlem’lerin atomik özelliğini başarmak (ve dayanıklılığını (durability)), hata

varlığında önemli bir konudur. İzolasyon ve tutarlılık özellikleri temelde,

eşzamanlı işlem’lerin çalıştırılmalarının düzgünce kontrol edilmesiyle idare

edilirler.

9

Bu eşzamanlılık kontrolü birkaç işlem’in simültane olarak çalıştırılmasına izin

verir, fakat yönetilen veri öğeleri topluluğu bu yolla tutarlı bir durumda kalır. Bu

tutarlılık var olan işlem’in veri öğesine özel bir sırada erişmesi ile başarılır. Bu sıra

sayesinde, final sonucu eğer tüm işlem’ler sıralı (seri) olarak çalıştırılırsa aynı

olur.

Eşzamanlılık kontrolü en iyi olarak üç farklı yöneticinin katmanlı bir yapıda

organize olduğu biçimde anlaşılır. En alt katman, bir veri yöneticisi’nden (data

manager) oluşur, bu veri üzerindeki geçerli okuma ve yazma işlemini

gerçekleştirir. Veri yöneticisi hangi işlem’in bir yazma veya okuma yaptığı ile

ilgilenmez. Doğal olarak, işlem’ler hakkında hiçbir şey bilmez.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Orta katman bir planlayıcı

(scheduler) içerir ve düzgünce kontrol edilmiş

eşzamanlılık için ana sorumluluğu taşır. Bu planlayıcı, hangi işlem’in hangi

zamanda bir okuma veya yazma işlemini veri yöneticisine geçireceğine karar verir

(izin verir). Bunu, izolasyon ve işlem’lerin tutarlılığını bir araya getiren bir yolla

tekil (individual) okuma ve yazma işlemlerinin planlanması sayesinde yapar.

10

En yüksek katman işlem yöneticisi’ni (transaction manager) içerir, bu işlem’lerin

atomik olma özelliğinin garanti edilmesinden birinci dereceden sorumludur. Bu

yönetici, işlem ilkellerini planlayıcı (scheduler) için planlayıcıya yollanan istekler

(scheduler request) haline çevirerek işletir. Yukarıdaki şekilde bulunan model,

aşağıdaki şekilde dağıtık duruma uyarlanmıştır. Her bir site kendi planlayıcısına

sahip ve veri yöneticisine sahiptir.

Bunlar beraberce yerel verinin tutarlı kaldığından emin olunmasından

sorumludurlar. Her bir işlem, tek bir işlem yöneticisi tarafından idare edilir.

Sonraki, tek sitelerin planlayıcıları ile iletişim kurar. Eşzamanlılık kontrol

algoritmasına bağlı olarak bir planlayıcı uzak veri yöneticileri ile iletişime

geçebilir.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Fig. 5.23

11

Şekil 5.23 ‘te, işlem’lerin idare edilmesinde yöneticilerin genel organizasyonu

şematik olarak gösterilmektedir.

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Fig. 5.24

12

Şekil 5.24 ‘te, dağıtık işlem’leri idare etmek için genel yönetici organizasyonu

şematik olarak gösterilmektedir.

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Serileştirilebilme (Serializability) :

Eşzamanlılık kontrolü’nün ana amacı, hata izole haldeyken aynı zamanda birden

çok işlem’in simültane olarak çalıştırılabilmesinin sağlanmasıdır. Bunun anlamı,

final sonucu, eğer işlem’ler belirli bir sırada biri diğerinden sonra gelecek şekilde

çalıştırılırsa sonucun aynı olacağıdır. Aşağıdaki tabloda (a) ile (c) arası üç adet

13

çalıştırılırsa sonucun aynı olacağıdır. Aşağıdaki tabloda (a) ile (c) arası üç adet

işlem, ayrı süreçler tarafından simültane olarak çalıştırılmaktadır.

BEGIN\_TRANSACTION

x = 0;

x = x + 1;

END\_TRANSACTION

(a)

BEGIN\_TRANSACTION

x = 0;

x = x + 2;

END\_TRANSACTION

(b)

BEGIN\_TRANSACTION

x = 0;

x = x + 3;

END\_TRANSACTION

(c)

Fig. 5.25

Schedule 1 x = 0; x = x + 1; x = 0; x = x + 2; x = 0; x = x + 3 Legal

Schedule 2 x = 0; x = 0; x = x + 1; x = x + 2; x = 0; x = x + 3; Legal

Schedule 3 x = 0; x = 0; x = x + 1; x = 0; x = x + 2; x = x + 3; Illegal

(d)

Şekil 5.25 ‘te görülenler; a) – c)

T1, T

2, ve T

3

şeklindeki üç adet i

şlem.

d) Polası planlar (possible schedules).

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Eğer bu yukarıdaki tabloda bulunan üç işlem seri olarak (sıralı) çalıştırılısa x’in

final değeri 1, 2 veya

3 olabilir. Bu en son çalışana bağlıdır (x , paylaşımlı bir

değişken, bir dosya veya benzeri bir varlık olabilir). Tablodaki (d) şıkkında çeşitli

sıralamalar görürüz, bunlar planlar

(schedules) olarak adlandırılır. Bunlar

birbirilerinin içinde (interleaved) olabilirler. Tabloda verilen; Schedule

1 gerçekte

14

birbirilerinin içinde (interleaved) olabilirler. Tabloda verilen; Schedule

1 gerçekte

seridir. Diğer bir deyişle, işlem’ler kesinlikle seridir (sıralıdır), böylece tanımdaki

serileştirilebilme koşulunu sağlar. Schedule

2 serileştirilmiş değildir, fakat hala

legal’dir, çünkü x için bir değer olan sonuç kesinlikle seri olan işlem’ler

çalıştırılarak elde edilebilmiştir. Schedule

3 ise illegal’dir, çünkü x ’e

5 değeri verir,

bu bazı şeylerin işlemlerin seri sırada üretilmediği (sonuç olarak) anlamındadır.

Birbirinin içinde olarak (interleaved) düzgünce tekil işlemlerin yapılabilmesi

sisteme bağlıdır.

Sistemin işlemin herhangi bir sırasını seçebilmesine olanak tanımakla, cevabın

doğru alınabilmesi sağlanılır, böylece kendi karşılıklı dışlamalarını yok ederek

programlama daha kolay hale gelir. Planları ve eşzamanlılık kontrolünü anlamak

için, tam olarak neyin hesaplanıldığını bilmek gerekmez. Diğer bir deyişle, x ‘in

diğerinin

2 veya

3 ile artmasının önemi yoktur. Daha önemli olan konu x ‘in

değerinin değişmiş olmasıdır. Sonuçta, işlem’ler bir seri okuma ve yazma işlemi

olarak (özel bir veri üzerinde gerçekleştirilen) gösterilebilir.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Yukarıdaki örnekteki üç işlem T1, T

2, T

3 şöyle gösterilebilir:

Eşzamanlılık kontrolü arakasındaki fikir doğru plan için çakışma işlemleri’dir

(conflicting operations). Eğer aynı veri öğesi üzerinde iki işlem çalışıyorsa,

write

(

T , x); read

(

T , x); write

(

T , x

) i i i

15

(conflicting operations). Eğer aynı veri öğesi üzerinde iki işlem çalışıyorsa,

çakışırlar ve eğer en az bir tanesi yazma işlemi ise, bu çakışma gerçekleşir. Bir

okuma-yazma çakışması’nda, tam olarak bir tanesi yazma işlemidir.

Aksi halde, uğraşılan şey bir yazma-yazma çakışması olur. Çakışan işlem’lerin

aynı işlem’den olup olmamasının bir önemi yoktur, farklı işlemlerden de

olabilirler. Önemli olan iki okuma işleminin asla çakışmamasıdır. Eşzamanlılık

kontrolü algoritmaları genelde okuma ve yazma işlemlerinin senkronize olmasına

bakılarak sınıflandırılırlar. Senkronizasyon, ya karşılıklı dışlayan mekanizmaların

paylaşımlı veri üzerinde oluşması sayesinde ya da zaman damga’larının kullanımı

ile sıralama işlemlerinde yer alırlar.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

Daha ileri bir ayrıştırma, kötümser

(pessimistic) ve iyimser

(optimistic)

eşzamanlı kontrol arasında yapılır. Kötümser yaklaşım temel olarak Murphy

yasalarına benzer, yani bir şey kötüye gidiyorsa, mutlaka kötü sonuçlanır. İyimser

yaklaşımda ise hiçbir şeyin kötüye gitmeyeceği varsayılır.

16

İki kademeli kilitlenme (Two-phase locking,

2PL) :

En eski ve en yaygın kullanılan eşzamanlılık kontrolü algoritması kilitleme

(locking) olarak bilinir. En basit biçimde, bir süreç okumak veya yazmak için bir

işlem’in parçası olarak bir veri öğesine ihtiyaç duyar, bu veri öğesi için

planlayıcı’dan (scheduler) bir kilit’i onaylamasını (grant) ister. Benzer olarak, bir

veri öğesi, daha fazla kullanılmayacaksa (gereksiz ise), planlayıcı kilit’i serbest

bırakmak ister.

Planlayıcı’nın görevi, böyle bir yolla sadece geçerli planların (schedules)

sonuçlarında kilitleri onaylar veya serbest bırakır (grant or release).

Diğer bir deyişle, sadece serileştirilebilen planları sağlayan bir algoritmayı

uygulamak ister. Böyle bir algoritma iki kademeli kilitleme’dir.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

İki kademeli kilitlenme (Two-phase locking) :

İki kademeli kilitleme’dir (

2 Phase Locking, 2PL), planlayıcı ilk önce büyüme

fazı

(growing phase) boyunca gereken tüm kilitleri ele geçirir. Daha sonra

küçülme fazı

(shrinking phase) boyunca bunları serbest bırakır. Buna göre

aşağıda verilen üç kurala uyulmalıdır:

17

1.) Planlayıcı oper(T, x) işlemini işlem yöneticisinden aldığında, halihazırda

onaylanmış (granted) bir kilit için bu işlemin herhangi bir diğer işlemle çakışıp

çakışmadığını test eder. Eğer bir çakışma varsa, oper (T, x) işlemi bekletilir

(delayed) yani böylece T işlemi’de bekler). Eğer hiçbir çakışma yoksa, planlayıcı,

x veri öğesi için bir kilit’i onaylar ve işlemi veri yöneticisine gönderir.

2.) Planlayıcı, veri yöneticinin hangi kilit ayarlanmışsa onun için olan işlemi

gerçekleştirimi kabul edene kadar, x veri öğesi için bir kilitli asla serbest

bırakmaz.

3.) Bir kez planlayıcı, T işlemi adına serbest bırakırsa, T adına başka bir kilit’i asla

onaylamaz, bu durumda hangi veri öğesi için T ‘nin bir kilit istediği’nin bir önemi

yoktur. T tarafından yapılan herhangi bir deneme (başka bir kilit’i ele geçirmek

için) bir programlama hatasıdır ve T abort (iptal) edilir.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

İki kademeli kilitlenme (Two-phase locking) :

Tüm işlemler eğer iki kademeli kilitlemeyi kullanırsa, tüm planlar birbirlerinin

içersinde olacak biçimde (interleaved) hale getirilip serileştirilebilirler. Bu yüzden

iki kademeli kilitleme geniş çapta kullanılmaktadır.

18

Bir çok sistemde küçülme fazı

(shrinking phase), işlem çalışması bitirilene

kadar, ya onaylanan ya da abort edilenme kadar yer almaz, kilit’in serbest kalması

ile olur. Bu kurala (politikaya) kesin iki kademeli kilitleme

(strict two-phase

locking) denilir.

Bu kural aynı zamanda kademeli abort’ları (iptal’leri) yok eder (cascaded

aborts), yani onaylanmış bir işlem’i yapılmamış hale getirir çünkü görmemesi

gereken bir veri öğesini görmüştür.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

İki kademeli kilitlenme (Two-phase locking) :

Birkaç tane iki kademeli kilitleme şeması bulunmaktadır:

▫ Merkezi (centralized)

2PL : Tek bir site kilit onaylama ve serbest

bırakmadan sorumludur.

▫ Birincil (Primary)

2PL : Her bir veri öğesi bir birincil kopyaya atanır. Kilit

19

▫ Birincil (Primary)

2PL : Her bir veri öğesi bir birincil kopyaya atanır. Kilit

yöneticisi bu kopyalama makinesinde kilitlerin onaylanması ve serbest

bırakılmasından sorumludur.

▫ Dağıtık (Distributed)

2PL : Verinin birden çok makineye dağıtıldığı

varsayılır. Birincil

2PL ‘nin ve merkezi

2PL ‘nin aksine her bir makinedeki

planlayıcı bu kilitlerin onaylanmış veya serbest bırakılmış olması ile ilgilenmez,

fakat işlem yerel veri yöneticisine gönderilir.

(devam)

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

İki kademeli kilitlenme (Two-phase locking) :

Fig. 5.26

20

Şekil 5.26 ‘da, iki kademeli kilitleme şematik olarak gösterilmektedir.

Dağıtık işler (Distributed Transactions)

Eşzamanlılık Kontrolü (Concurrency Control) :

İki kademeli kilitlenme (Two-phase locking) :

Fig. 5.27

21

Şekil 5.27 ‘de, kesin iki kademeli kilitleme şematik olarak gösterilmektedir.

Referans kitap / Kaynakça

[Kaynakça 1]

Tannenbaum, A., “Distributed System: Design and

Principles”, Prentice Hall.

22

Ders’in sonu…

23

Dr. Bahadır KARASULU karasulu.bahadir@gmail.com