DAĞITIK PROGRAMLA

Dağıtık programlama, dağıtık, açık, ölçeklenir, saydam ve hataları giderebilen bir programlama modelidir. Uzak yordam çağrıları (RPC), işletim sistemi komutlarını bir ağ bağlantısı üzerinde dağıtmak için kullanılır. Temel olarak uygulamanın verisinin ayrı bir mantıkta, uygulamada iş yapacak kodların ayrı bir mantıkta, uygulamanın arayüzünün ise daha başka bir mantıkta sunulması demektir.

Birden fazla bilgisayar ve/veya sunucunun birbirleri arasında iletişim kurması ve birbirleri ile replike şekilde bir ağ bütünü olarak çalışmasını sağlar. Ağda bulunan donanımlar kullanıcıya tek bir bilgisayar gibi davranır ve en iyi performansı sağlamayı amaçlar.

Dağıtık programlamanının önemli bir amacı , kaynakları paylaşmaya duyulan gerekliliktir. Bu kaynaklar donanımsal bileşenler (harddisk, yazıcı) olabileceği gibi, dosyalar, veri tabanı, gibi servislerdir. Dağıtık programlamanının temelinde , "bir bilgisayar bir işlemi 5 saniyede gerçekleştiriyorsa, aynı işlem 5 bilgisayarda 1 saniyede yapılabilir" kuralı yatmaktadır.

Bütün işler tek bir kod, hatta tek bir bilgisayar tarafından yürütülmez. Her bir katmanı ayrı bir bilgisayar tarafından koşturulur.

- İzole değildir.
- Bulut' un temelini oluşturur
- Senkron ve replike'dir
- Yedekli çalışır
- Birçok bağımsız bilgisayarın varlığı belirgin değildir
- Ölçeklenebilirlik artar.

İstemci ile sunucunun birbirinden ayrıldığı dağıtık yazılımlara birçok sebepten dolayı ihtiyaç duyarız. Bunlardan birkaçı aşağıda sıralanmıştır:

1. Bazı karmaşık programlar dağıtık bilgisayarlar üzerinde çalışır.
2. Birçok ucuz bilgisayar üzerinde bulunan işlemcileri kullanarak çoklu işlemci (CPU) ile çalışmanın birçok üstünlüğü vardır.
3. Dağıtık programlama aynı zamanda ölçeklenebilirliği de arttırır.
4. Bazı yazılımlar vardır ki sadece belli donanımlara sahip bilgisayarlarda çalışırlar.
5. Güvenlik gerekçesi ile bazı servisler ayrı bilgisayarlarda olmak zorundadır.
6. Servislerin sadece kaynak olarak kullanıldığı durumlar.

Örnekleme yapacak olursak;

Dağıtık programlamayla birden fazla CPU(işlemci) olmasına rağmen kullanıcı bunu tek işlemci olarak görür. Kullanıcı çalıştırdığı programları nerede çalıştırdığını ve dosyaların nerede yerleşmiş olduğunu bilmez. Bu tür işlemlerin hepsi sistem tarafından otomatik yapılır.

Google, youtube, yahoo, myspace facebook, soundcloud, yandex gibi firmalar dağıtık programlama teknolojisi ile çalışan örneklerdir.

Dağıtık sistemlerin altında yatan temel prensipler şunlardır;

- işlemler (processes)
- iletişim (communication)
- isimlendirme (naming)
- senkronizasyon (synchronization)
- tutarlılık (consistency)
- hata payı (fault tolerance)
- güvenlik (security)

Dağıtık programlama günümüzde geliştirilen modern uygulamaların üç temel özelliğinden biridir. Diğer modern dağıtık uygulamalar;

- **Katmanlı Mimari**(layered architecture): İstemci/sunucu ya da n katmanlı model. Katman sayısı arttıkça karmaşık azaltır (reduce complexity).

Bilgisayarların donanım olarak hızlanması ve gelişmesiyle birlikte katman sayısı da artmıştır;

1. Monolitik Uygulamalar: Tek bir çalıştırılabilir dosyadan oluşan uygulamalar.

Monolithic (tek parça) uygulamalarda bir veya birden fazla fonksiyonalite bir aradadır.

Böyle bir ortamda ise uygulamaların çalışabilmesi için IBM tarafından geliştirilmiş server teknolojisi olan mainframe sunucular kullanılmaktadır.

Monolithic uygulamaların geliştirilmesi uzun bir süre almaktadır, bakım ve uygulamada

yapılacak düzeltme, yeni özellik ekleme gibi işlemler zordur ve çok efor gerektirmektedir.

Günümüzde hakim olan yazılım geliştirme mimarisi ve mantığına aykırı olarak monolithic

uygulamalarda yazılmış olan bir işlevsellik veya bir özellik taşıyan bileşen aynı programlama

dilinde yazılmış olsa bile başka bir uygulamada kullanılamaması ve yazılım geliştiricilerin

yeni uygulama için aynı kodu tekrar yazmak zorunda kalması veya uygulamanın bir kısmını

değiştirilmesinin,kaldırılmasının istenmesi durumunda bile tüm uygulamanın kodu üzerinde

değişiklik yapılması gerekmektedir

2. İstemci/Sunucu Uygulamaları: Bu mimaride uygulama 2 katmanlıdır.İşlem yükü istemci ve sunucu arasında paylaştırılmış olup; istemci tarafı sunum ve iş mantıklarını işletirken, sunucu tarafı veriye erişim ile ilgilenmektedir. Veri katmanı bir bilgisayarda diğer katman olan istemci katmanı ise kullanıcı bilgisayarlarında bulunur.

Yazılım bileşenlerin tekrar kullanılamaması sorunu vardır. Yani Visual Basic'te yazılmış bir bileşeni C temelli diller desteklemiyor, kullanamıyor yada C temelli bir dille yazılmış bir bileşen Visual Basic de kullanılamıyor.

3. Üç Katmanlı Mimari: Nesneye dayalı yazılım geliştirme ile bilikte nesneler arasındaki iletişimin bir başka katman olarak tasarlandığı uygulamalardır. İki katmanlı mimariden farklı olarak kullanıcı arayüzü ile iş bileşenleri birbirinden ayrılarak, kullanıcılara istemci yükleme işleminin çok kolaylaştığı uygulamalardır.
4. Çok Katmanlı Mimari: Günümüz yazılım mimarisi olup mümkün olduğunca katmanların birbirinden bağımsız tasarlandığı uygulamalardır. Günümüzde bu tür uygulamalarda katmanların arayüzleri web servis olarak tanımlanır ve istemci katmanlar bu servisleri kullanarak işlevlerini yürütür. Web

servislerinin etkin kullanıldığı bu tür uygulamalara **SOA** (Service Oriented Architecture) uygulaması adı verilir.

- **Bileşen Kullanımı**(component using): Bileşen kullanımı yazılımda tekrar kullanılabilirliği (reusing) arttırır.

Bileşen Modeli (Component Model) hem monolithic hemde istemci/sunucu uygulamaların problemlerine çözüm bulmuştur. Artık yazılım geliştiriciler geliştirdikleri uygulamaları bileşenler ile yazarak bu bileşenleri ileride başka projelerde tekrar kullanabiliyorlar.

Yazılım yeniden kullanımı farklı seviyelerde ve büyüklüklerde olabilir. Kod seviyesinde yeniden kullanım; nesne ve fonksiyonun yeniden kullanımı; basit bir fonksiyonu yerine getiren ufak yazılım bileşenleridir. Örnek olarak matematiksel bir işlem yapan fonksiyon veya nesne sınıfı verilebilir. Bu yaklaşımda bu fonksiyonlar standart kütüphanelerin içinde yer alabilir ve bu kütüphanelerin yeniden kullanımı ile sağlanabilir. Bilesen seviyesinde; bir uygulamanın bileşenleri, alt sistemler olabileceği gibi basit nesneler de olabilir, başka uygulamalarda yeniden kullanılabilir. Tüm uygulamanın yeniden kullanımı; sadece kod

seviyesinde değil tüm uygulama sisteminin başka teknolojiler veya çalışma ortamlarına uydurulması ya da uygulamalara göre farklılaştırabilmesi ile sağlanabilmektedir. Diğerlerine göre daha büyük oranda yeniden kullanım söz konusudur, bunun için yeniden kullanım sistematik olarak ele alınmalıdır. Ortak bir imariden türeyen uygulama aileleri örnek verilebilir.

Yazılım geliştirmenin başlangıcından günümüze kadar özellikle Nesneye Dayalı Programlamanın ortaya çıkmasıyla birlikte yeniden kullanım için birçok farklı yöntem geliştirilmiştir. Bir Yazılım Ürün Hattı mimarisi geliştirirken bu mevcut yöntemlerden daydalanılmaktadır. Bu yöntemlerden bazılarını şu şekilde sıralayabiliriz;

- Yazılım Mimarileri (SOA, DSSA, vb), bir yazılım sisteminin yapısını, davranışını ve önemli özelliklerini

göstermek için yüksek seviyeli soyutlama sağlar. Genelde yazılım mimarileri bir bileşen kümesi ve bu bileşenlerin entegrasyonu ve konfigürasyonundan oluşur.

- Tasarım Kalıpları
- Alana Özgü Diller ve Kod Üreteçleri

- Uygulama Çatılarını (Frameworks), yazılım sistemleri için kod seviyesinde soyutlama sağlayan sınıflar

kümesidir.

- Servis Tabanlı Sistemler, sistemlerinin birbirlerine servisler ile bağlandığı mimarilerdir.
- Genel Kurumsal Planlama Programları, ERP
- Sınıf Kütüphaneleri
- İlgiye Dayalı Yazılım Geliştirme (Aspect Oriented Software Development); yazılıma kod seviyesinde esneklik ve yapılandırılabilirlik sağlamak için ilgileri ayırmayı amaçlar. Bir başka yeniden kullanım yöntemi olarak; belirli bir uygulama alanı içerisinde tekrar kullanılabilir yazılım bileşenleri geliştirmeye dayanan Alan Mühendisliği ve Yazılım Ürün Hatları çalışmanın geri kalanında sıkça bahsedilecektir. Burada önemli bir konu belirli bir yazılım sistemi için bu yöntemlerden hangisinin kullanımının en uygun olacağının belirlenmesidir. Genellikle bu geliştirilecek sistemin gereksinimlerine, teknolojilere ve o alandaki sahip olunan deneyime bağlıdır.

Dağıtık uygulamalarda, sunucu bileşenlerinin istemci bileşenleri ile iletişim sağlayabilmesi için uygulamada olması gerekenler ;

1. İç Süreç (in-process) için bir gereklilik bulunmaz.
2. Dış Süreç (out-process) için iletişim, süreçler arası iletişim (IPC-Inter-Proces Commnication) ile sağlanır. Bunu sağlayan teknolojiler tarihsel sırada aşağıda verilmiştir:

a) Clipboard

b) Message Queue and Send Message

c) Shared Memory

d) LRPC (Lightweight Procedure Call)

e) RPC (Remote Procedure Call)

f) COM (Component Object Model), DCOM (Distributed COM)

g) CORBA (Common Object Request Broker Architecture)/RMI(Remote Method Invocation)

h) HADOOP

3. Ağlar Arası Süreç (inter-network server) için iletişim, "Paketleme" (marshalling) ile sağlanır. Bunu sağlayan teknolojilerin başlıcaları:
4. JAXB (Java Architecture for XML Binding),
5. .NET Remooting

4. Uzak Süreç (remote server) için iletişim, ortak sözleşme (common contract) ile sağlanır. Bunu sağlayan teknolojilerin başlıcaları:
5. ASP.NET Web Service, WCF (Windows Communication Foundation),
6. JAX-WS (Java Architecture for XML Web Service)

Burada birinci madde dışındaki süreçlerle geliştirilen yazılımlar dağıtık yazılımlar olarak adlandırılır. Bu durumda **dağıtık programlama** (distributed programming) yapılır.

Üçüncü ve dördüncü maddede belirtilen sürteçlerde, istemci tarafından bir başka süreçteki sunucu bileşenlerinin yöntemleri çağrılarak programlama yapıldığından bu programlama türüne **uzak programlama** (remooting) adı verilir.

DAĞITIK SİSTEM KURMA YÖNTEMLERİ

RPC

Server üzerindeki servisleri kontrol ettiğimizde karşımızı çıkan RPC (remote procedure call) arka planda haberimiz olmadan birçok şeyi gerçekleştiren bir servistir.
RPC temel anlamda istemci/client ve sunucu/server arasında yapılan işlemlerin iletişimi için dizayn edildi.Bir işlemin gerçekleşmesi için bir gönderici (server) ve birde alıcı (client) vardır.Örneğin Microsoft Outlook ve Exchange server ikilisi gibi.Aynı şekilde server üzerindeki bir çok serviste bu mimariyi kullanarak birbirleriyle haberleşirler.İşte bu haberleştirmeyi güvenli kılan ve kolaylaştıran şey RPC'dir.

***RPC Yapımında Temel Adımlar*:**

1) İstemci taslağa çağrıda bulunur. Bu çarı normal yolla stack'e itilen parametreli yere prosedür çağrısıdır.

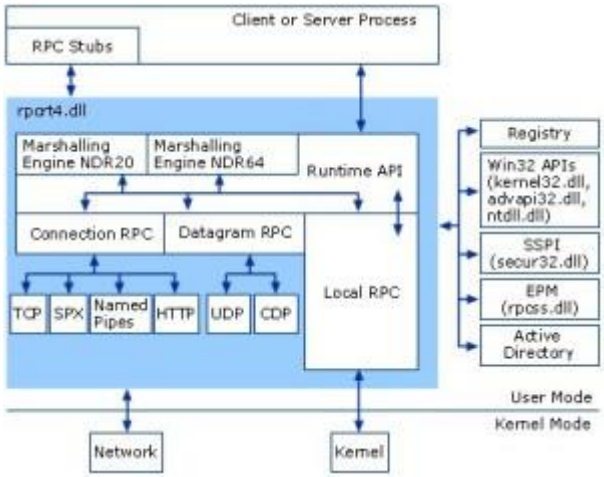
2) İstemci taslağı parametreleri bir mesaja paketler ve mesajı göndermek için bir sistem çağrısı yapar. Paketlenmiş parametreler marshalling diye adlandırılır.

3)Çekirdek istemci makineden sunucu makineye mesajı yollar.

4)Çekirdek gelen paketleri sunucu taslağa iletir.

5)Son olarak,sunucu taslak sunucu prosedürünü çağırır.

Rpc Mimarisi



Yukarıdaki mimaride gördüğümüz componentlerden bahsedicek olursak;

Client or server process:
Bir RCP isteğini başlatan yada yanıt veren program yada servis

RPC stubs:
RPC isteğini başlatmak için client yada server tarafından kullanılan program

Marshalling engine(NDR20 or NDR64):
RPC client ve server ları arasında ortak bir RPC interface'i oluşturur.NDR20 32 bit mimarisi için, NDR64 ise 64 mimarisi için tasarlanmıştır.Client ve server bu marshalling engine sayesinde iletişim için karşılıklı anlaşmayı sağlarlar.

Runtime application programming interface (API):
Server yada client'a RPC için direk bir arayüz oluşturur.RPC client ve serverları RPC'yi başlatmak içnin runtime API yi çağırırlar.

Connection RPC protocol engine:
RPC bir connection"“oriented protocol isteğinde bulunduğunda kullanılır.Burada RPC nin dışarıya doğru bir bağlantıdamı yoksa dışarıdan içeriye doğru bir bağlantıdamı olduğu dizayn edilir.

Local RPC protocol engine:
Server ve client aynı host içerisinde barınıyorsa kullanılır.

Registry:
RPC servisinin ilk yüklemesinde erişim sağlanır.Buradaki registry anahtarları, RPC kullanıdığı ip port aralıklarını yada network aygıtlarının isimlerini barındırır.

Win32 APIs(kernel32.dll, advapi32.dll, ntdll.dll):
Kernel32.dll , sistem servislerinin hafıza yada kaynak yönetimlerini sağlayan Windows NT tabanlı bir API client dynamic-link library (DLL) dosyasıdır.
Advapi32.dll , gelişmiş bir Windows 32 base API DLL dosyasıdır.Güvenliği destekler.
Ntdll.dll , Windows NT nin sistem fonksiyonlarını kontrol eder.

SSPI(secur32.dll):
RPC için bir güvenlik arayüzü oluşturur.Kerberos, NTLM, ve Secure Sockets Layer (SSL) ın authentication ve encryption için kullanımını sağlar.

RPC için kullanılan network portları aşağıdadır.

Service Name	UDP	TCP
HTTP	80, 443, 593	80, 443, 593
Named Pipes	445	445
RPC Endpoint Mapper	135	135
RPC Server Programs	<Dynamically assigned>	<Dynamically assigned>

Desteklenen network protokolleri:

Protocol	RPC Type
Transmission Control Protocol (TCP)	Connection-oriented
Sequenced Packet Exchange (SPX)	Connection-oriented
Named Pipe	Connection-oriented
HTTP	Connection-oriented
User Datagram Protocol (UDP)	Connectionless
Cluster Datagram Protocol (CDP)	Connectionless

COM

Yeni bileşenler oluşturmak veya varolan bileşenleri uygulama programları veya başka bileşenler içerisinden kullanmak için Microsoft Firması tarafından tanımlanmış bir altyapıdır. COM (Component Object Model), Microsoft'un pekçok teknolojisi/uygulaması için temel altyapıyı oluşturan bir teknolojidir. Uzun zamandır yazılım dünyasının karşılaştığı birçok probleme çözüm olan ve sürekli yenilenerek zamana ayak uyduran bir teknolojidir. Bu teknolojinin arkasında bulunan temel düşünce yazılım bileşenlerinin en az işgücü ile en verimli biçimde yeniden kullanılabilmesidir. Bu tanım bir yazılım geliştirici için daha önceden yazılmış olan bir kodun tekrar yazılmaması, bir BT yöneticisi için ise bu kodun tekrar yazılmasından kaynaklanan maliyet ve zamandan tasarruf demektir.Nesne tabanlı bir programlama modeli olan COM uygulamaların birlikte kullanılabilirliğini sağlamıştır. Yani iki veya daha fazla uygulama veya bu uygulamalara ait bileşenler birbirleriyle iletişim sağlayabilmektedirler. Burada esas can alıcı nokta bu uygulamaların farklı dillerde yazılmış olduklarında ve hatta farklı bilgisayarlar üzerinde bulunan farklı

işletim sistemleri üzerinde çalışıyor olmaları durumunda bile bu iletişimin başarıyla sağlanmasıdır.

COM teknolojileri Windows işletim sistemi altında değişik şekillerde kullanıcıların karşısına çıkmaktadır. Örneğin:

* Uygulama geliştirme aracı içerisinden kullanılabilen hazır kontroller,
* Bir Word dokümanı içerisine Paintbrush resminin eklenmesi
* Pek çok programda bulunan taşı-ve-bırak (drag-and-drop) özelliği,
* WEB sayfaları içerisine ActiveX kontrollerin eklenmesi,
* Word, Excel gibi bir uygulamanın VBScript gibi basit bir dille kontrol edilmesi,

DCOM

Microsoft tarafından bilgisayarlar arasında dağıtılmış uygulamaların daha rahat oluşturulabilmesi ve çalışabilmesi için yazılmıştır.

COM'un ağ üzerinde farklı bilgisayarlarda bulunan bileşenlerin iletişim kurmasını sağlamak üzere geliştirilmiş sürümüdür. DCOM'un sağladığı altyapı ile uygulamalar ağ üzerinde dağıtılabilir. DCOM ile bileşenin bulunduğu yer bilgisi istemciden gizlenir. Bu sayede istemci bileşenin türü ne olursa olsun bileşene bağlanmak için aynı kodu kullanır.
DCOM, COM programlama modeline yerden bağımsızlık, yeni güvenlik mekanizmaları, bileşen yönetimi ve bellek yönetimi konularında ekler getirmektedir. COM ile istemci uygulamaları sunucunun paketleme türü ne olursa olsun aynı

programlama modelini kullanmaktadır. COM nesneleri istemci prosesi içerisine yüklenebileceği gibi aynı makine üzerinde farklı bir proses olarak da yüklenebilir. DCOM bu konum saydamlığı özelliğini daha da genişleterek bileşenlerin ağ üzerinde herhangi bir bilgisayarda olmasını olanaklı kılmaktadır.

CORBA

Common Object Request Broker Architecture, kısaca **CORBA**, (**Ortak Nesne İstem Aracısı Mimarisi**), Nesne Yönetim Grubunun'nun (OMG) Nesne Yönetim Mimarisi'nin (OMA) ana bileşenlerinden birisidir.

Nesne Yönetim Mimarisi Nesne Modeli ve Referans Modelinden oluşur. Nesne Modeli heterojen bir ortamda dağılmış nesnelerin nasıl tanımlanabileceğini belirler. Referans Modeli ise nesneler arası etkileşimleri tanımlar. Dolayısıyla Nesne Yönetim Mimarisi heterojen ortamlara dağılmış beraber

işleyebilen dağıtık nesnelerin geliştirilmesine ve konuşlandırılmasına yardımcı olur.

CORBA sayesinde programcılar kullandıkları nesnelerin hangi dilde yazıldığına, dağıtık olup olmadıklarına, işletim sistemlerine ve iletişim protokollerine bakmaksızın programları geliştirebilirler. Dağıtık nesnelerin iki yönü vardır. Bunlar sunucu ve istemci yönüdür. En basit haliyle bu ilişki şöyle gerçekleşir:

Sunucu uzak bir ara yüz (remote interface) sağlar ve istemcide sağlanan uzak ara yüzü çağırır. Bu ilişki RMI ve CORBA gibi birçok dağıtık nesne standartlarında benimsenen bir yaklaşımdır. Bu bağlamda, sunucu ve istemci terimlerinden kastedilen uygulama düzeyindeki etkileşimlerden ziyade nesne

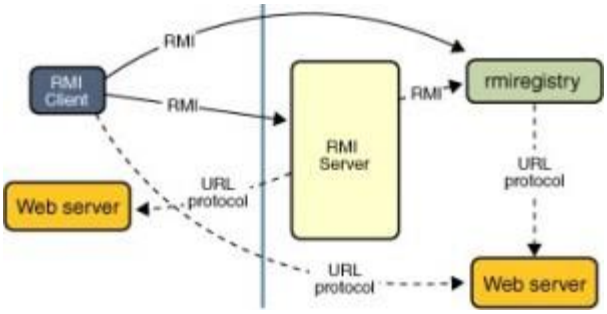
düzeyindeki etkileşimlerdir. Yani bir uygulama, herhangi bir nesne için sunucu olabilirken başka bir nesnenin istemcisi olabilir.

JAVA / RMI

Remote Method Invocation yani RMI farklı sunucularda olan metodların başka bir sunucudan çağırılarak kullanılmasında kullanılmaktadır. Java Nesnelerinin kullanmak için farklı bir sunucuya/sanal makina'ya bağlanıtyorsa buna biz Dağıtık Programlama diyoruz. RMI, Dağıtık Programlama yapmamıza olanak sağlamaktadır.

RMI Mimarisi

Katmanlardan oluşmaktadır. Bu katmanlar bize istek, cevaplama , yönlendirme işlemlerini yapmaktadır.



- **Stub**

Uygulamadan gelen isteklerin ulaştığı yer Stub'dur. Stub Bağlantı ile başlatır JVM (Java Virtual Machine) ile iletişime geçer. Hataları yakalar, Skeleton'dan gelen sonuçları istemciye iletir.

- **Skeleton**

Ulaşan isteklerin ne olduğuna bakarak yönlendirme görevini sağlar. Parametre okur, uzak nesneyi çağırır, metotdan gelen sonucu almakla görevlidir.

- **Remote Reference**

Uzaktaki nesnelerin ne yapması gerektiğinin anlaşıldığı yerdir. Skeleton'dan gelen istekleri Transport katmanının anlayacağı şekle çevirir. Nesne işlemlerinin yürütüldüğü yerdir.

- **Transport**

Bağlantı ile ilgili işlemlerin yapıldığı katmandır. TCP'nin kullanılması işlemlerini yapmaktadır.

Uygulamamız 3 ayrı classdan oluşmaktadır.Bunlar :

Client

Server

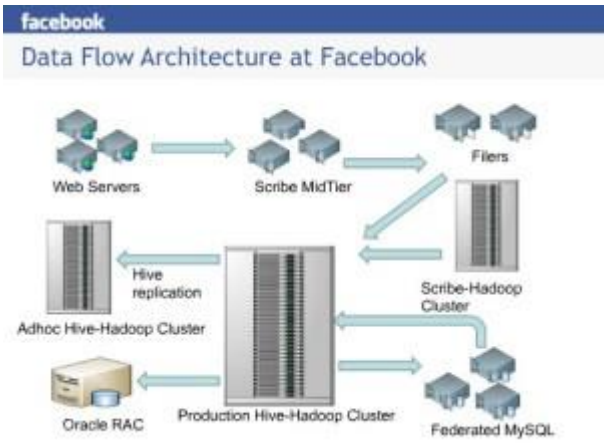
ve Remote

HADOOP

Dağıtık sistemlerde en çok kullanılan program Hadoop'tur. Hadoop, Java programlama dilinde yazılmış framework'tür. Amacı Google Dosya Sistemi teknolojisi olan Map-Reduce algoritmasını birçok bilgisayara dağıtarak uygulayı sağlamaktır.

Google, Yahoo ve Facebook gibi şirketler devasa boyutlarda veriyi analiz etmek için artık SQL, RDBMS kullanmıyorlar. Hadoop adlı anahtar-değer tekniğini kullanan bir ürünü kullanıyorlar. Arama motoru pazarında başarılı olmak öncelikle *teknolojik altyapının dağıtımlı çalışması*ndan geçiyor. Yani bütün arama, depolama, ve indeksleme gibi işleri birden fazla bilgisayara dağıtarak yapmak. Hatta böyle bir altyapısı olduğu için Google'dan *aslında bir dağıtımlı bilgisayar şirketi* diye bahsedilir. Yani herkes internetten bilgileri toplayıp depolayabilir ama bunu yüksek performansta yapmak yükü bir bilgisayar tarlasına dağıtabilmekten geçiyor. Bilgisayarlar arasındaki bu iş bölümünü Google kendi geliştirdiği MapReduce denilen bir yazılım platformu ile yapıyor. Ancak MapReduce ile aynı işi yapan ve açık kaynaklı olan bir yazılım platformu daha var: Hadoop.

Facebook, Google, Yahoo gibi dev firmalarların milyonlarca müşteriyi memnun edebilmesinin altında yatan sır Hadoop'tur.. Bir örnek ile Hadoop'u anlatacak olursak; Facebook'ta bir kişiyi aradığımızda karşımıza onbinlerce seçimi sadece birkaç saniye sunması biraz imkansız gibi görünmüyor mu? İşte bu Hadoop sayesinde mümkün oluyor. Veya Amazon'da bir ürünü aradığımızda onlarda veri arasında bize doğru sonucu nasıl oluyor da bu kadar hızlı sunuyor?



Kaynaklar:

http://www.docstoc.com/docs/42482421/Distributed-Programlama-Nedir#

http://www.sistemvenetworkmuhendisi.com/dagitik-mimari-nedir-nasil-calisir/

http://akademik.maltepe.edu.tr/~akhanakbulut/Parallel%20and%20Distributed%20Systems%20(CEN483-BIL483)/TR%20Ders%20Notlari/dagitikSistemler.pdf

http://yazilimgunlugu.com/dagitik-uygulama-ve-net-cozumleri-1-makalesi/272.aspx

http://www.slideshare.net/tatalala1/chapter-1-introduction-13235584?qid=2a455cf4-1c79-4c45-b498-e4105876e8c9&v=qf1&b=&from_search=62

http://en.wikipedia.org/wiki/Distributed_computing

http://chimera.labs.oreilly.com/books/1230000000929/ch14.html

http://burakisikli.wordpress.com/category/distributed-systems/

http://www.emo.org.tr/ekler/21a3a42db785e7f_ek.pdf

http://www.slideshare.net/hiteshmarkam/concepts-of-distributed-computing-cloud-computing

http://www.emo.org.tr/ekler/de8edbbe1547786_ek.pdf

http://web.firat.edu.tr/feeb/kitap/C12/69.pdf

http://anilerduran.com/rpc-remote-procedure-call-mimarisi-nedir-nasil-calisir/

http://tr.wikipedia.org/wiki/Uzaktan_yordam_%C3%A7a%C4%9Fr%C4%B1s%C4%B1

http://www.tameroz.com/content/files/complus.pdf

http://www.turkpaylasim.com/cevahir/2009/07/01/com-ve-iliskili-kavramlar/

http://belgeler.cs.hacettepe.edu.tr/yayinlar/eski/CORBA_20122034.pdf

http://tr.wikipedia.org/wiki/CORBA

http://ceng.ktu.edu.tr/labs/bs_java.pdf

http://bengiloglu.com/?p=85

http://devveri.com/hadoop-nedir

http://mehmetayberk.com/hadoop-nedir/