BMÜ-463 DAĞITIK SİSTEMLER
FİNAL SINAVI - 11.01.2013

1) MapReduce nedir, nasıl çalışır açıklayınız?

2-a) Web Servisi nedir, neden ihtiyaç duyulmuştur?

2-b) Web Servis standartlarını yazıp kısaca açıklayınız.

3) Hizmet Odaklı Mimari nedir, kısaca tartışınız?

5) Bulut Hesaplama ve Grid Hesaplama kavramlarını kısaca tartışınız.

0326049
Mete TELGEKEN

FIRAT ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ
BMÜ-463 DAĞITIK SİSTEMLER
ARA SINAV

1) Paralel Hesaplama nedir? Hangi amaçlar için yapılır?

2) Dağıtık Hesaplama kavramını kısaca açıklayınız.

3) Dağıtık Sistemlerde Orta Katman Yazılımı (Middleware) nedir, ne amaçla kullanılır? Şekille açıklayınız.

4) Dağıtık Sistemlerde ölçeklenebilirlik kavramını açıklayınız.

5) Dağıtık Sistemlerin temel özelliklerini ve karakteristiklerini yazıp kısaca açıklayınız.

Başarılar Dilerim
Dr. Galip AYDIN

— Vize 27.11.2013 —
13:15

Multi Core --

Cluster Computing: Süper olmayan evde kullanılan PC'lerin herhangi sayıda olabilecek sıradan. Bunlara paralel hesaplama yapma. Geliştirilmiş yazılımları var.

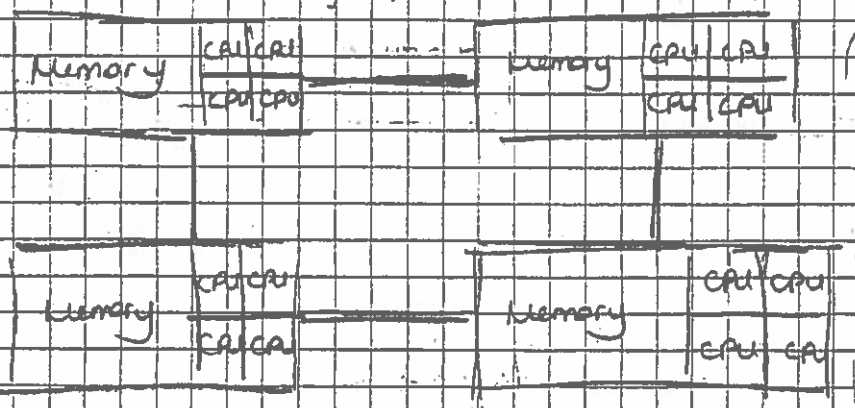
Super Computing / High performance Computing: Çok büyük hesaplamaları olan bilgisayarlar. Örn: 200.000 CPU'su var.

PARALEL HAFIZA MİMARİLERİ:

(1) Paylaşımlı Hafıza Mimarisi: Cache coherency olabilir de olmayabilir de.
Tüm işlemciler merkezdeki paylaşımlı hafızaya ulaşabilir. Bir işlemci değişiklik yaptığında diğerleri bunu haberdar oluyor.

a) UMA: Uniform Access Time: Tüm CPU'lar için aynı hafızaya erişim süreleri aynı.
Cache coherency: Eğer bir işlemci paylaşımlı hafıza bölgesini update ettiğinde diğerleri haberdar oluyor.


b) NUMA: Non Uniform Memory Access: Çoklu farklı makine her makine kendi hafızasına var. Aynı bus ile bağlı olmayan makineyi bir bus ile bağlıyoruz. (Ağ topolojisi bağlıdır)




Böyle yaparak RAM-CPU miktarını arttırıyoruz. Onlar her makine kendi içine hızlı erişiyor diğerlerine daha yavaş erişiyor.

CPU diğer hafızaları adresler (Ortak)

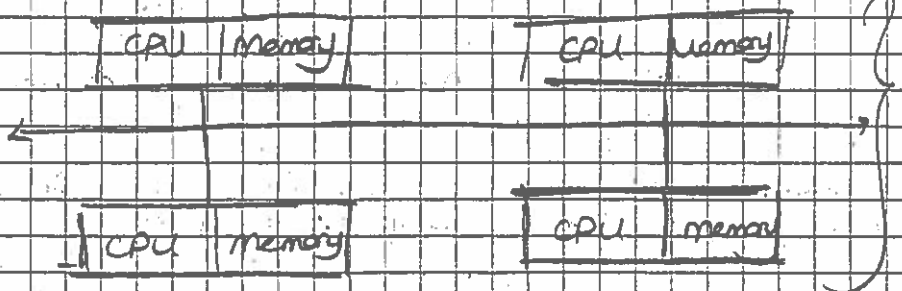
Cache coherency:
Her değişikliği CPU'ya bildirmek zordur NUMA'da çok zordur ve maliyetlidir. Modern çoğu CPU cache coherency'i sağlamayan durumlarda kullanılmak, yani bir değişikliğin başka verileri etkilemediği durumlarda.

Hafızanın adres uzaylarını kontrol etmek kolaydır, fakat bu maliyetin değişmesi cihazların kontrol etmek zordur. Makine sayısı ↑ iş karmaşıklaşır.

## (2) Distributed Memory:

Her processor kendi local hafızasını kullanabilir Global adres uzayı yoktur. Bir local değişikliği diğerlerine bildirme durumu yok.
Fakat bir işlemci diğerinin bilgisine erişmek istediğinde bazı algoritmalarla işlemciler haberleşir. İnternet veya boşta iletişim bu yapıda olabilir.

Avantajı: Ciddi anlamda ölçeklenebilir. Makine sayısı arttıkça hesap hızı artar.
Bu hızlı hesap hızını bilirler. Uzaydır.




Veri haberleşmesiyle ilgili işleri programın düşünüp yapması gerekir.
Bizde global değişkenler ile senkronizasyonu zordur.

Bir global değişkendeki değişikliği diğerlerine bildirmek zahmetlidir
Cache coherency yoktur

Hibrit Distributed: Değişik hafıza kullanarak çok hızlı işlem yapılabilir.
2 hafıza mimarisinin olumlu yönlerini almış

FIRAT ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ
BMÜ-463 DAĞITIK SİSTEMLER
ARA SINAV – 30.11.2011

Meltem Öztürk

SORU 1) Paralel Hesaplama nedir? Hangi amaçlar için yapılır?

CEVAP 1)
Birbirinden kısmen ya da tamamen bağımsız işlemlerin (process) birbirine paralel (koşut) biçimde çalıştırılması prensibine dayanır.

- Para ve zamandan tasarruf
- Daha büyük problemleri çözmek için
- Eş zamanlı çözüm üretebilmek için
- Yerel olmayan kaynakların kullanılması için
- Seri hesaplamanın sınırlarından dolayı

Paralel hesaplama yöntemleri
MPI modeli: Kaynaklar birbiriyle mesajlaşarak paralel hesaplama yapılır en çok kullanılanıdır
Data paralel model: Datalar parçalanır, sonuçlar birleştirilir

örneğin büyük bir fotoğraf olsun. Bu resmin fotoğrafını 100 artırmak için bu resi parçalara bölünür, işlemler yapılır, sonuçlar birleştirilir.

SORU 2) a) Michael J. Flynn çok işlemcili bilgisayarları hangi kriterlere göre sınıflar? b) Flynn's Classical Taxonomy'ye göre mimari sınıflarını ve kısaca ne anlama geldiklerini yazınız.

2 matris toplamı kadar uzun, kod yazılırken paralel olarak hesaplanacağı belirtilir.

CEVAP 2)
a) Instruction ve Data
instruction → Program kodunun paralelleştirilmesi
Data → Büyük veriler üzerinde paralelleştirme

b) SISD : Single Instruction, Single Data → Eşzamanlı yapar ama zamanı kısaltır, seri mantıktadır, sonuç tahmin edi
SIMD : Single Instruction, Multiple Data → Farklı datalar farklı CPU'lara gönderilip aynı işlem yapılır. Hızlı işl mi sağlar.
MISD : Multiple Instruction, Single Data → Aynı veri farklı filtrelerden geçirip sonuçta bir işlemin yapılması kriptolojik işlemlerde kullanılır.
MIMD : Multiple Instruction, Multiple Data → Yaygın kullanılan bilg. türüdür. Aynı anda birçok program çalıştırılı multitasking

SORU 3) Dağıtık Hesaplama kavramını kısaca açıklayınız. Ne amaçla gerçekleştirilir?

CEVAP 3)
Dağıtık sistemler fiziksel olarak dağılmış bilgisayarların bir araya getirilmesi ve koordine şekilde çalıştırılmalarıyla elde edilirler. Tanenbaum'a göre bir Dağıtık Sistem "Kullanıcısına tek bir bilgisayarmış gibi görünen ve bağımsız bilgisayarlardan meydana gelen sistemdir". Dolayısıyla bağımsız bilgisayarların tek ve tutarlı bir bilgisayarmış gibi çalıştırılmasını sağlayan sistemlerdir.

- Uzak kaynakların dağıtık şekilde kullanılabilmesi ve kolay erişim için
- Ölçeklenebilirlik için
- Sistemi oluşturan parçaların ayrıntılarından ve oluşabilecek hatalardan kullanıcının en az etkilenebilmesi için
- İşbirliğini güçlendirebilmek için
- Daha emniyetli çalışma için
- Mobility

SORU 4) Merkezi (Centralized) ve merkezi olmayan (Decentralized) dağıtık mimarilerin özelliklerini kısaca açıklayıp bu mimarilere örnek veriniz.

CEVAP 4)
Merkezi mimariler klasik server-client mantığıyla çalışırlar. Frameworkler, çok katmanlı mimariler, web sunucuları, veritabanı sunucuları vb. request/reply mantığıyla çalışan sistemlerdir. Geleneksel örneği üç katmanlı modeldir. Bu modelde User(interface/Processing layer/Data layer katmanları bulunur. Sıkı bağlı sistemler olduğundan ölçeklenebilirlik açısından sıkıntı yaşanabilir.

Merkezi olmayan mimariler genelde peer-to-peer denen sistemlerdir. Mantık olarak client veya server olmayan her iki rolü de üstlenen eş değerde bilgisayarlardan oluşurlar. Structured/Unstructured ve Hybrid P2P modelleri vardır. Bittorent ağları başarılı örneklerdir.

Başarılar Dilerim
Dr. Galip AYDIN

ir işin kısa sürede yapılması için 2 yöntem vardır.
1- Kodun optimizasyonunun yapılması 2- Paralel sistemler

Paralel hesaplamalarda BÖLME ve BİRLEŞTİRME işlemlerinde zaman kaybı vardır.

· Avantaj - Dezavantaj

* Aynı makina üzerinde hesaplama yapıldığında, aynı hafıza kullanıldığı için bu dezavantajdır.
* Paylaşılmış hafızalarda kullanıcıların kolaylıkla elde edebildiği sonuçları elde edebilir.
* farklı bilg. üzerinde ağ üzerinden mesajlaşmayla gerçekleşir. Bu da işlemi yavaşlatır.
* Paralel hesaplama genelde bilimsel alanlarda kullanılır, ticari işlerde kullanılmaz.

#

Cloud : Elimizdeki kaynakları servis olarak kullanmaktır. Bir server sanallaştırılarak birçok server elde edilir. API : Servis çalıştırma ve durdurma işlemlerini gerçekleştirir. Ör/amazon
→ Laas : Donanım hizmet olarak kullanmayı sağlar.

MapReduce :
- Hadoop = Varolan veri işlerin, paralel olarak yapılmasıdır.
- Mahout = Makina öğrenmesi

Web Servisler : WSDL : Bir servis ve hizmetin ne olduğunu tanımlar. Bir XML dosyasıdır.
SOAP : Başka bilg. nesnelere erişimi sağlar.
UDDI : Web servis database.

Servislerin beraber çalıştıkları sistemlere hizmet odaklı mimari (SOA) denir.

NoSQL : mongoDB
Cassandra
HBASE
Redis

---

Middleware katmanı : Çok sayıda makina üzerinde çalışan programları kullanıcıya göstermeden tek bir sistem varmış gibi gösterir.
* Publish - Subscribe = orta katman yazılımlarının kullandığı yöntemlerden bir. Bu yöntemde bir server var ve bu server kendisine ait kişilere mesaj gönderebiliyor

---

Küme hesaplama nedir? Neden gerçekleştirilir? Kriterleri yazınız. Dezavantajları yazınız.

Kafgaro Hesaplama kavramını açıklayın. Neden geliştirilmiştir?

## Paralel Hesaplama Özellikleri

1- İş daha kısa sürede yapılır.
2- Az sayıda kaynak kullanılarak, çok sayıda işlem yapılır.
3- Paralel hesaplama paralel sistemlerin amacını yerine getirmek için kullanılır.
4- Paralel hesaplamada, kaç tane CPU varsa hepsi kullanılabilir.
5- Genelde bilimsel alanlarda kullanılır, ticari alanlarda kullanılmaz.

## Paylaşılmış hafıza - Dağıtık Hafıza

* Paylaşımlı hafızada, hafıza üzerinde bir değişiklik yapıldığında tüm kullanıcılar aynı hafızayı kullandıklarında sonuçları hızlı bir şekilde görebilirler.

* Dağıtık hafızada, farklı makinelerde haberleşme network üzerinden haberleşmeyle olacağından yavaş olur.

# Paralel hesaplamanın Dağıtık Sistemler üzerinde kullanılması 2 aşamadan oluş
- → map aşaması
- → reduce aşaması

## Dağıtık sistemden İstenenler

1.) Üzerindeki çokluğu kullanıcıdan gizlemeli.
2.) Ölçeklenebilir olmalı.
3.) Sürekli çalışabilir olmalı.
4.) Yeni kaynakları hizmete alırken eski kaynakların kesilmemesi gerekir.
5.) Katmanlı mimari kullanılır. Herbir katmanda farklı işten sorumlu makinalar vardır.

⇒ middleware (orta katman) ; Çok sayıda makinayı tek bir sistem gibi görmemizi sağlar.

publish-subscribe ; Orta katman yazılımının kullandığı yöntemlerden biridir.

## Dağıtık Sistemler (hedefler)

→ Kaynaklar kullanıma açılır. (Grid)
→ Kaynakların network üzerindeki dağıtıklığı gizlenmeli.
→ Sistemin açık olması (Openness)
→ Ölçeklenebilir olması

Dağıtık Sistemin Gizliliği (2)

1.) Access 2.) Location 3.) Migration 4.) Concurrency (Tutarlılık) 5.) Failure

## ÖLÇEKLENEBİLİRLİK

İhtiyaç duydukça sistemin otomatik olarak genişleyebilmesidir. Çok sayıda isteğe aynı anda cevap verebilmesidir.

Çeşitleri:

1.) Size scalability : Bir server kaç tane istek karşılar? Server'lar farklı yerlerde dursa cevap hızı ne olur? gibi soruları yanıtlayan çeşitidir.

2.) Administrative scalability : Sistem ölçeklenebilirliği destekliyor mu? Sorunun karşılığıdır.
İdari ölçeklenebilirliğe internet zaten yapar.

3.) Geographical scalability : Ör/ Gittigidiyor sitesinin web serverları istanbul ve Ankara'da bulunsun, farklı bir bölgeden istek gelince sorun olur mu? sorusuna cevap verir.

- Ölçeklenebilirlik 3 noktada sorun yaşar:

1.) Networkten kaynaklanan haberleşme gecikmeleri

internet sitesinden cevap beklerken başka bir iş yapılabilmesi.
göndereceğimiz verinin boyutunu küçültmek (Sayfanın tamamını past etmek yerine gerekli kısmı past etmek (Ajax)
Server tarafında her istek için thread oluşturmak.

2.) Yükün dağıtılması

Yük = Hesaplama, veri

→ Hesaplama yükünü dağıtmak için farklı serverlarda çalıştırılarak paralel programlama yapılır.
→ Veri yükünü dağıtmak için farklı mekanizmalar bulunur. Ör/ DNS. Her bölgede farklıdır ve isteklere karşılık verir.

3.) Verinin Çoklanması (Replication)

Veritabanı kopyasının aynı anda birden fazla yerde tutulmasıdır. Verinin çoklanması ölçeklenebilirliği hızlandırır.
Verinin kopyalanması web sitelerinde mirrored web ile yapılır.
mirrored web: Web sayfaları birbirine taşıtılır.

Ölçeklenebilirlik problemleri

1.) Tutarsızlık
2.) Çok büyük sistemler için sağlanamaması
3.) Global senkronizasyonu sağlamada sorun yaşanması

## Dağıtık Sistemler Yapılandırılırken yapılan hatalar;

Network homojen değildir.
Network güvenli değildir.
Topoloji her an değişebilir
Bant genişliği sınırsız değildir.
Gecikme '0' değildir.

## Dağıtık Sistem türleri

1.) Dağıtık Hesaplama Sistemleri
2.) Dağıtık Bilgi Sistemleri
3.) Dağıtık Pervasive (Yaygın) Sistemleri)

---

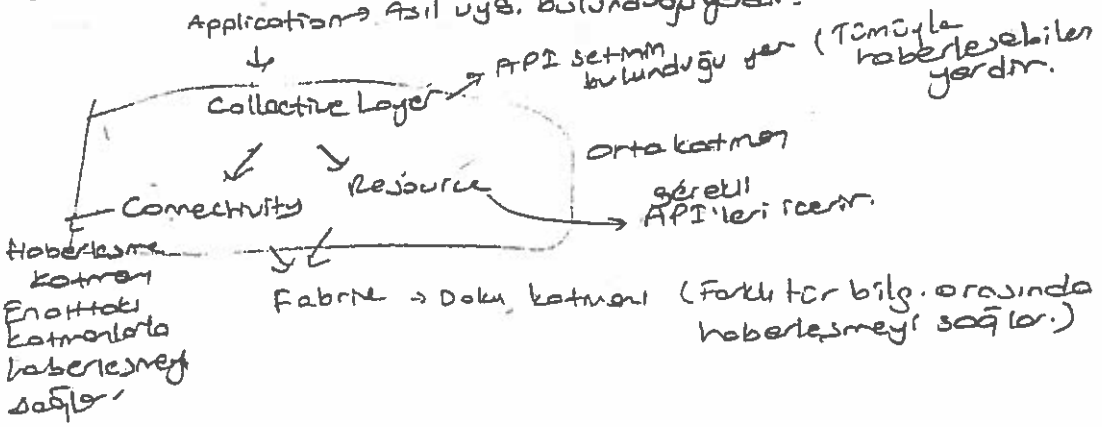
Dağıtık Hesaplama Sistemleri
→ Cluster hesaplama → Grid Hesaplama
→ Cloud Hesaplama → Hizmet odaklı mimariler

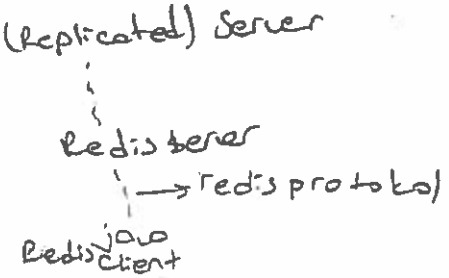
---

**Cluster Hesaplama**: Birden fazla bilgisayarın network üzerinden birbirine bağlanarak toplam kapasitenin arttırılmasıdır. Beowulf ve MPI paralel (ikincil) hesaplama yazılımlarındadır.


→ Dağıtık bellek yoktur.
→ Bazı uygulamalar için ölçeklenebilirliği yakalayarak azaltır
→ İletişim hızı, bellek okuma/yazma hızına göre yavaştır.

**Grid Hesaplama**: Clusterdan daha büyük, heterojen sistemlerde kullanılır. Amaç; global bir işletim sistemi sağlamaktır.






- Redis bir NOSQL türüdür. Key value şeklinde çalışır.

Redisin redis.io'dan kurulumunu yaptık

Terminal satırından

src/redis-server yazarak

farklı terminalden redis-cli yazarak client oluşturduk.

Bilgi sayarları bridge yapısını kullanarak sanal ipleri bağladık

~~Java client ve pom yapılandırma dosyalarını oluşturduk.~~

pom.xml

Cache mekanizması : En hızlı erişilebilecek bellek anlamına gelir. Server tarafında dinamik web sayfasının cache'leri tutulur. (2)
↳ JSP, ASP, PHP ...

Dışarıdan gelen istekler önce cache'e gönderilir eğer orda varsa direkt cevap gönderilir. Cache mekanizmasının en çok kullanıldığı yer web haritalarıdır. Server'da bir değişiklik yapıldığında bunlar cache yenilenir.

Image Pronrt : En tepede bir resim varsa, altta bir çok resim vardır. Bunlar file system'de tutulur.

Replication ve Cache'de en önemli sorun tutarsızlıklıktır.

Dağıtık Sistemler yapılandırılırken sıkça yapılan hatalar:

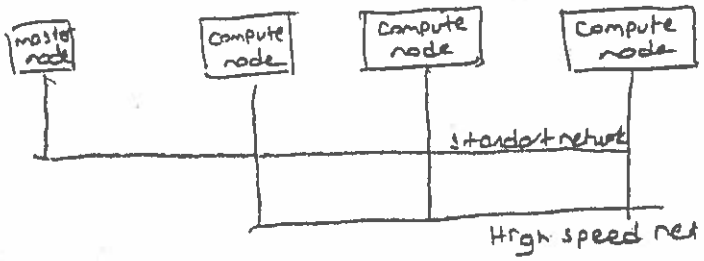
1. Network güvenilir değildir.
2. Homojen değildir, çok sayıda server vardır.
3. Topoloji her an değişebilir.
4. Bant genişliği sınırsız değildir.
5. Gecikme sıfır değildir.

Dağıtık Sistemin Türleri :

- ~~Dağıtık Hesaplama Sistemleri~~ (Distributed Computing Systems)
- Dağıtık Bilgi Sistemleri
- Dağıtık Pervasive (yaygın) Srst

Dağıtık Sistemlerde hesaplama yöntemleri :

1) Cluster Computing (Küme hesaplama) = Büyük bir problem örneğin geniş çapta bir simülasyon yapacaksak bunu ancak süper bilgisayarlarla yapabiliriz. Başka bir yol ise küme hesaplamadır.
MPI paralel programlamada sıkça kullanılır. Bir bilgisayar master seçilir. Bunlar kullanıcı ile iletişimi sağlar.



Neden dağıtık Sistem kullanıyoruz? (1)

1.) Hesaplama kaynaklarını çok kolay bir şekilde erişilebilir hale getirir.
Sistem kaynakları ortak kullanılır. Ör/ SETIEhome ve grid
2.) Kaynakların network üzerinde dağılmış olduğunu gizlemek.
3.) Sistemin açık olması
4.) Ölçeklenebilir olması

Dağıtık sistemden beklenenler

1.) Kaynak paylaşımı arttıkça güvenlik sorunu ortaya çıkar
2.) Güçlü sistemler elde etmek için birçok kaynağın kullanılması gerekir. Kaynakların nerede olduğunu kullanıcılar bilmez, sadece yönetici bilir. Yani dağıtıklığı gizlenmesi gerekir.

* Erişim ile ilgili bazı noktaların gizli olması gerekir. Verinin arka tarafta ne tip formatta bulunduğunun gizli olması gerekir.
* Serverların çalışıp çalışmadığının sürekli olarak kontrol edilmesi gerekir
* Bazen veritabanının başka bir yere taşınması gerekir. Buna migration denir. Bu taşınma işleminin gizlenmesi gerekir.

Kısacası dağıtık şeffaflığı : → Erişim, Location, Migration, Tutarlılık, Failure ← Bir hata oluştuğunda kullanıcıdan gizlenmesidir.

3.) Çok sayıda kaynağı bir araya getirdiğimizde, çok sayıda platformu bir araya getirmiş oluyoruz. Bu sistemlerin birbiriyle haberleşmesi gerekir. Bu da 3 şekilde gerçekleşir. a.) Hardware b.) Platform c.) Language

4.) Çok sayıda isteğe aynı anda cevap verebilmelidir. Ölçeklenebilirlik.

Ölçeklenebilirlik çeşitleri

→ Size scalability
→ Geographical scalability
→ Administrative scalability

Ölçeklenebilirliği etkileyen problemler

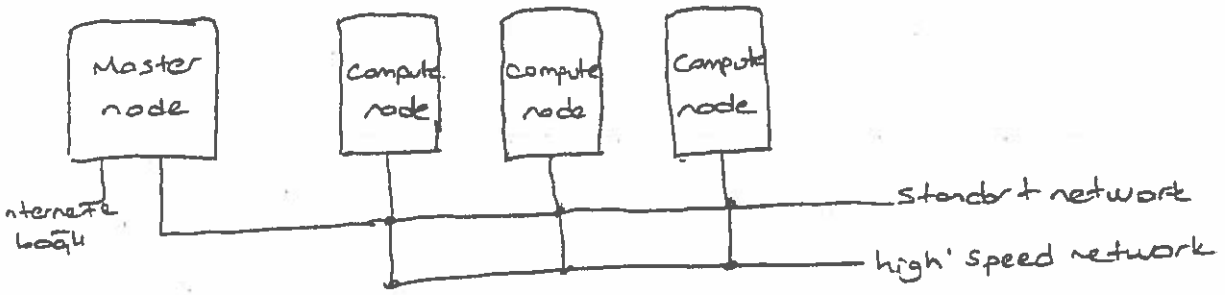
(a) Haberleşmedeki gecikmeler : Gecikmeyi önlemek için verinin boyutu küçültülür ya da kullanıcı başka sayfalara yönlendirilir. AJAX kullanılır daha hızlı olsun diye.

(bi) Yükü dağıtmak : Yük: Hesaplama ve veri
Hesaplamayı dağıtmak için farklı serverlarda çalıştırarak paralel programlama yapabiliriz.
Veriyi bölmek için farklı mekanizmalar bulunur.

(c) Replication (Verinin Çoklanması) : örneğin, bir database'imiz olsun. Bu database istekler alıyor. Her yerde verilerin kopyalarını tutmamız gerekir. Bu kopyalama işlemine replication denir. Bu işlem mirror (ayna) yöntemiyle çalışır.

## Cluster Hesaplama

- Birden fazla bilgisayarın birbirine network üzerinden bağlanarak, bilgisayarların toplam kapasitesinin arttırılmasıdır.
- Depolama, işletim sistemi, uygulama program arayüzü, uygulamalar gibi değişik sistem katmanlarında gerçekleştirilebilir.
- Cluster homojendir. Yani bilgisayarların hepsi aynı türden aynı hıza sahiptir.
- Beowulf ve MPI küme hesaplama yazılımlarındandır.
- MPI'da bir bilgisayar master olarak seçilir, master bilgisayara bir işlem geldiğinde bölüp diğer bilgisayarlara gönderir.
- Küme hesaplama dezavantajları:
  - ortak bellek yoktur.
  - ölçeklenebilirliği yoktur. Özel uygulamalar ram yazdır.
  - iletişim hızı, bellek okuma/yazma hızına göre yavaştır.




- Master node'un görevi işleri paylaştırma ve toplamadır.
- Node'ların kendi hafızaları üzerinde işlerini yapmaları çok hızlıdır. Fakat 2 node haberleşmesinde iş yavaşlayacağında high speed network kullanılır.

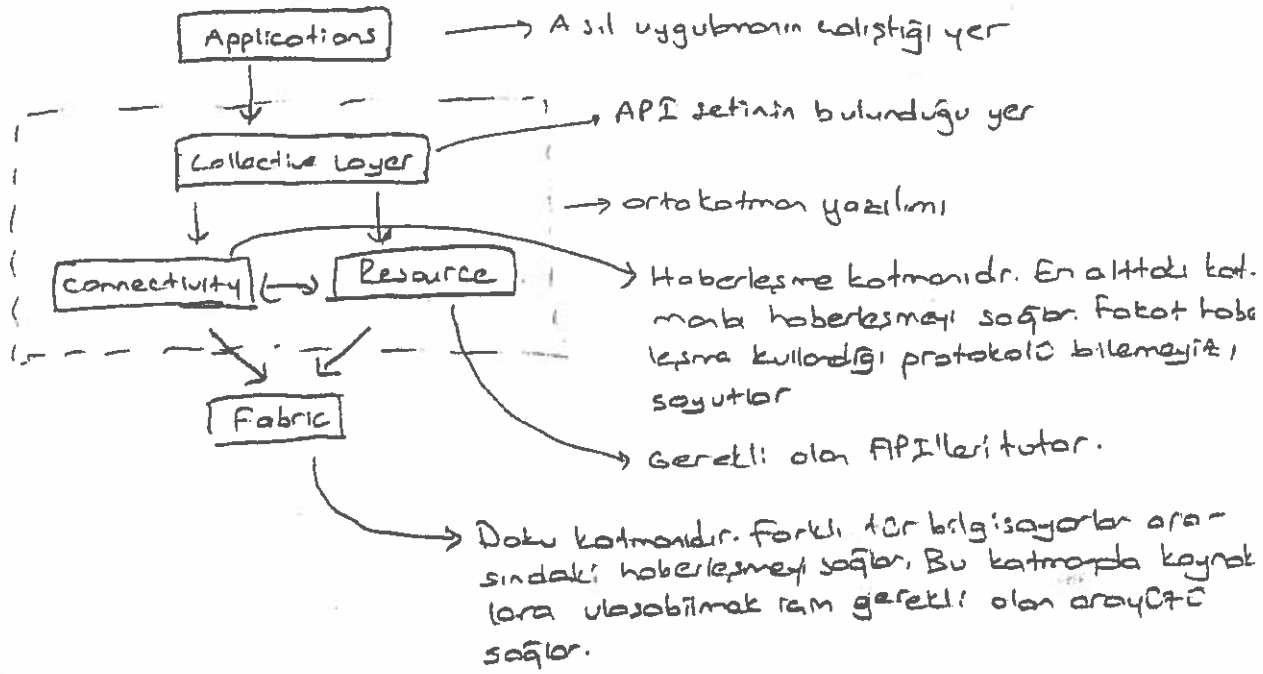
## CLOUD HESAPLAMA

- Server'ın özelliğini bilmeden istediğimiz kaynağı aldığımız mimari türüdür.
  - IaaS → Hizmet olarak Altyapı
  - PaaS → Hizmet olarak Platform
  - SaaS → Hizmet olarak yazılım.

## Grid Hesaplama



- Cluster'lardan daha büyük, heterojen sistemlerde grid hesaplama kullanılır.
- Grid hesaplama yapılacak bilgisayarın özellikleri, işletim sistemleri birbirinden farklı olabilir.
- Grid Hesaplamada amaç: Global bir işletim sistemi oluşturmak ve çok büyük hesaplamaları dünyanın her yerine yayılmış kaynakları kullanarak yapmaktır.




## HİZMET ODAKLI MİMARİLER

- SOA: Hizmet odaklı mimari örneğidir.
- Hizmet odaklı mimariler, dağıtık sistemlerin en başarılı ve en yaygın kullanılan mimarisidir.
- Birçok hizmet birleştirilerek büyük bir hizmet elde edilir.
- Web servislerden oluşur.

## Web Servislerin Çalışma Yapısı

- Servis sağlayıcısı Web servisini oluşturur ve bu servisi bir web sunucusu üzerine koyar.
- Oluşturulan servisi bir UDDI kaydı olarak tanımlayabilmek WSDL teknolojisi kullanılarak bir adet WSDL uzantılı dosya hazırlanır. (Bu işlem IDE tarafından otomatik olarak gerçekleştirilir.)
- Servis sağlayıcısı servisi bir UDDI kaydı olarak kaydeder.
- İstemcinin başka bir web servisi olabilir, UDDI sorgulamasını gerçekleştirerek servisin bir özetini kendi tarafında oluşturur.
- İstemci tarafı oluşan bu servisi kullanarak karşı taraftaki Web servise bir istek gönderir. Bu istek SOAP mesajları aracılığıyla gerçekleştirilir.
- Karşı taraf istenen bilgiyi SOAP aracılığıyla XML formatında geri döndürür.
- Alıcı taraf SOAP protokolünü kullanarak gelen XML'deki bilgiyi eşleşe dönüştürür ve sonucu uygulama içinde istediği şekilde kullanır.

---

Hesap.java — Source Package içinde

```
@WebService()
public class Hesap {
public Integer toplamaYap(
@WebParam(name="deger1") int deger1,
@WebParam(name="deger2") int deger2 ) {

return null;
}
}
```

---

(Web Services içinde)

```
@WebMethod(operationName="toplamaYap")
public Integer toplamaYap(@WebParam(name="deger1")
int deger1, @WebParam(name="deger2") int deger2) {

return deger1 + deger2;

}
```

## Web Servis Özellikleri:

- HTTP protokolü üzerinden iletişim kurulur.
- XML web servisleri SOAP üzerinden iletişim kurar.
- Açık protokolleri kullanarak haberleşirler.
- Diğer uygulamalar tarafından kullanılabilirler.
- XML tabanlı bir teknolojidir.
- SOAP basit ve msj tabanlı bir iletişim sağlar.

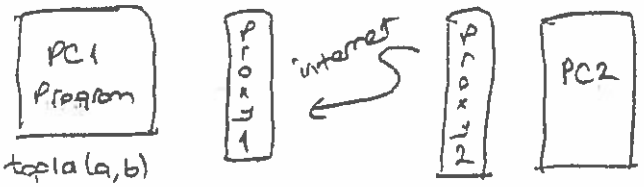
SOAP
UDDI

## WEB SERVİSLER



- Farklı bilgisayarlarda paralel programlama yapmak için birbiriyle haberleşmelerini sağlarlar.

RMI: Başka bir bilgisayardaki bir programı, bir metodu çalıştırmak için geliştirilen ilk teknolojidir.




Proxy2, Proxy1'e anlayacağı şekilde mesaj gönderir. Topla fonksiyonu ile a ve b parametrelerini gönderiyor Bu mesaj bir network mesajıdır TCP ya da UDP protokolüne kullanabilir. Proxy1 gelen mesajı çıkarıp Java nesnesine dönüştürüyor, PC1'e veriyor. Hesaplama yapılıp sonuç Proxy1 üzerinden PC2'ye network mesajı olarak gönderiliyor.

mimari karmaşık olursa RMI'yı burda kullanamayız.

Web servis: Esnek, herkesin kullanabileceği, platformdan ve programlama dilinden bağımsız bir şekilde kullandıracak mimaridir.

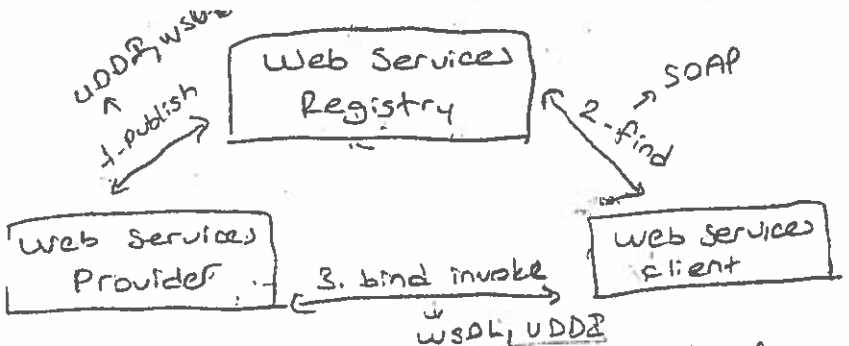
RPC, ORPC (DCOM, CORBA, JAVA RMI)

- Bunlar beraber çalışamazlar.
- Çoğunda firewall aşılamıyor.
- DCOM Windows'a / RMI Javaya çok bağlı.
- Karmaşıklardır.

Web Servisi standartları

- XML direkt gönderilmez. SOAP kullanır. SOAP üzerine XML mesajımızı da ekler öyle yollarız. Mesajlaşma HTTP → XML → SOAP şeklindedir.
- Web servisleri tanımlamak için WSDL dili geliştirilmiştir.
- Web servislerin keşfedilmesi için kullanılan register UDDI'dir. Web servisler kayıt ettiğimiz, arayıp bulduğumuz depolardır.

SOAP bir mesajdır, içeriği XML tipinde mesaj barındırır. Bu mesajın ne tipte olacağını WSDL belirler.




Web servis modeli (Publish – find – bind)



Küme çeşitleri
- Yüksek kullanılabilirlik
- Yük Dengeleme
- Yüksek Başarımlı Hesaplama
- veritabanı

YBH
YD
YK

Tanımlamalar

XML (Extensible Markup Language)
- web servislerinde veri tanımlamak için kullandığımız standarttır.
- HTML'de tag'lar bellidir, XML'de biz oluştururuz.
- XML oluşturma amacı; standart tiptaki veriyi bir yerden diğer yere iletmektir.
- Uluslararası bir standarttır.

UDDI (Universal description discovery and integration):
- web servisleri hakkında bilgi yayınlamak ve almak için kullanılan XML tabanlı bir teknolojidir.
- Web servislerin keşfedilmesi için kullanılan registerdır.

WSDL (Web Servis description Language):
- web servislerini, servislerin içerdikleri metodları, bu metodların dış ve iç tiplerini tanımlamakta kullanılır.
- WSDL dosyalarında çeşitli UDDI girdilerine ve SOAP mesajlarına referanslar bulunmaktadır. Karşı taraf WSDL dosyasını aldığında bu referansları inceleyerek web servisin içeriği hakkında tam bilgiye ulaşabilir.

SOAP (Simple Object Access Protocol): → haber → envelope: H, header: , body: /
- Web servisi ile web servisi istemcisi arasındaki haberleşmenin gerçekleştiği XML tabanlı protokoldür.
- Bir mesajdır içinde XML tipinde bir mesaj barındırır.

SOA (Service Oriented Architecture): → • Tekrar kullanılabilirlik • Bağlılığı azaltma • Basitleştirme yaklaşımı • Soyutlama
- Servis yönelimli mimari anlamına gelen bu yaklaşım web servisi tabanlı uygulamaların çalışma mantığını açıklar.
- Uygulamalar servisler halinde dağıtık bir şekilde yayınlanır. Kullanıcının gördüğü taraftaki yazılımlar ise bu servislerden ihtiyaç duydukları öğeleri ve kullanıcıya çıktı verir.

# DAĞITIK SİSTEMLER

## Paralel Hesaplama

Dağıtık sistemler, fiziksel olarak farklı alanlarda bulunan hesaplama araçlarının internet ile birbirine bağlandığı sistemlerdir.

Paralel hesaplama, işlemlerin farklı bilgisayarlarda gerçekleştirilip tek bir bilgisayarda toplanmasıdır. Paralel hesaplama için program paralel yazılması gerekir. Paralel program kodu programın çalıştırılacağı her bilgisayara eklenmelidir. Kodlarda özel derleyiciler tarafından derlenir.

Paralel hesaplamanın temel özelliği ilk başlarda farklı bilgisayarlarda yapmakken daha sonra aynı makinede birden fazla CPU'nun olduğu durumlarda gerçekleştirilmiş. Bir bard üzerine 4 CPU, 1 hafıza koyup hafıza ortak olarak kullanılmış. (Paylaşımlı hafıza) Paylaşımlı hafızanın avantajı:

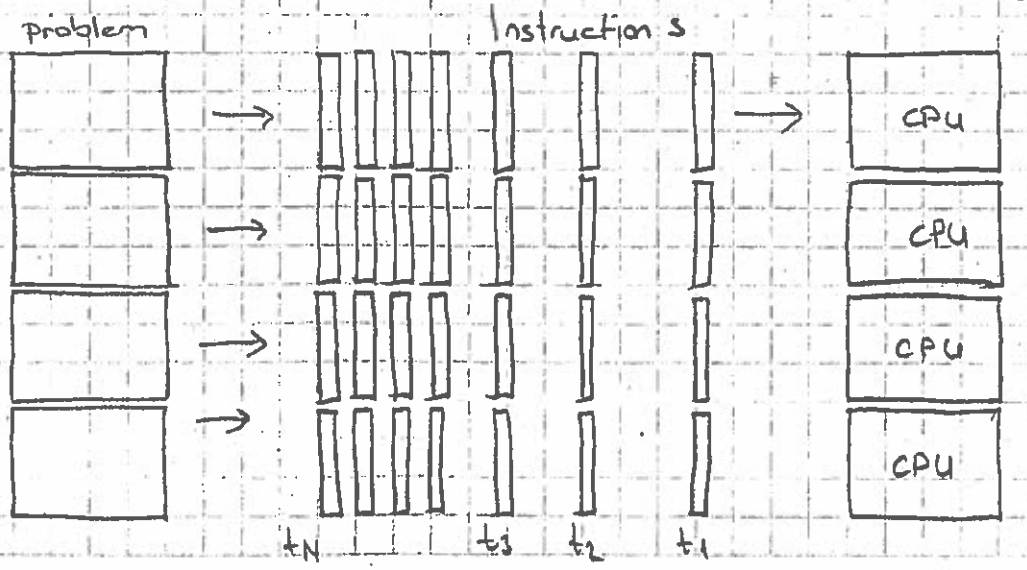
→ Verinin parçalanmasına gerek kalmamıştır.
→ Bilgisayarlar arasındaki haberleşmeden doğan zaman kayıpları engellenmiştir.

Paralel hesaplama elimizdeki bir problemi birden çok parçaya ayırarak daha hızlı çözebilmeyi amaçlar. Ancak bazı problemler vardır:

→ Birden fazla CPU olmalıdır. (Birden fazla komut çalıştırma)
→ Bir problemin paralel olarak çözülebilmesi için eşzamanlı olmalıdır.

"Bir problemi elimizdeki sınırlı kaynaklarla en hızlı şekilde nasıl çözebiliriz?" sorusuna cevap arar paralel hesaplama.

Parçalandığı zaman birbirini etkilemeyecek problemler olmalı. Komutlar eşzamanlı çalışmalı (Birden fazla CPU'ya ihtiyaç var)



Paralel Hesaplama Kullanım Alanları:

→ Atmosfer
→ Fizik (Kuantum fiziği, atom fiziği)
→ Biyoteknoloji, Genetik
→ Kimya
→ Jeoloji, Sismoloji
→ Bilgisayar Bilimleri
→ Database, data mining
→ Web search

Kullanım Sebepleri:

Temel sebebi daha kısa sürede bir problemi çözebilmek.

→ Zaman ve para tasarrufu
→ Büyük problemleri çözebilmek
→ Eş zamanlı (concurrency) çalışma imkanı sağlar. (accessgrid.com)
→ Elde olmayan kaynakları kullanabilmek.
→ Seri hesaplamalarda işlemlerin limitinin olması. Seri hesaplamaları kullanmamamızın nedenleri:
  * Fazla bilgisayar yok.
  * İşlem hızının sınırlı olması.
  * CPU'da bir hesaplama limitinin olması. Harddisklin sınırlı hızı (rpm) var. Veri transfer hızını, hesaplamayı kısıtlar.
  * Ekonomik kısıtlama.

Nasıl Yapılır?

→ Multiple execution units (Çoklu hesaplama birimleri - birden fazla CPU)
→ Pipelined komutlar
→ Multi-core

Paralel Hesaplama Sınıflandırma.

Flynn's tarafından yapılmıştır. Bu sınıflandırmanın temelinde;

→ Data
→ Instruction

Flynn's Classical Taxonomy

→ SISD (Single Instruction, Single Data) → Tek CPU
→ SIMD (Single Instruction, Multiple Data) → Birden fazla CPU
→ MISD (Multiple Instruction, Single Data)
→ MIMD (Multiple Instruction, Multiple Data) → CPU'lar bağımsız

Single Instruction, Single Data

→ Seri bilgisayarlar
→ Paralel hesaplamanın sonuçları belli
→ En eski, en çok kullanılan model

CPU'ya bir veri alınır ve sadece bir sonuç döndürüyor.

Single Instruction, Multiple Data

Birden fazla CPU aynı anda bir komutu işler. Veriler farklıdır. Örneğin, bir fotoğrafın %10 renk düşürme tek işlem ama birden fazla veri var. Ekran kartı işlemcilerinde, GPU'larda kullanılır. Yaygın değildir.

Multiple Instruction, Single Data

Bir veri birden fazla CPU üzerinde işlenir. Örnek olarak; şifreleme, ses filtreleme verilebilir.

Multiple Instruction, Multiple Data (Super Computer)

En çok kullanılan, bilgisayarlarda var olan model. CPU'ların hepsi bağımsız çalışır. Senkron veya asenkron çalışılabilir. Networkte bağlı bilgisayarlar cluster yapılır. Bağımsız veriler üzerinde çalışılır.

Aynı bilgisayar üzerinde yapılırsa, paralel programlama,

Farklı bilgisayar üzerinde yapılırsa dağıtık hesaplama adını alır.

SISD → Tek core'lu bilgisayarlar

SIMD → GPU'da kullanılır.

MISD → Filtrelemelerde kullanılabilir. Şu anda yok.

MIMD → Şu anda kullanılan bilgisayarlar. (En çok kullanılan)

Paralel Programlama Genel Terimler

Task: Hesaplama işinin birbirinden ayrılan parçaları, bağımsız çözülebilecek kısımlardır. Bir task'ın diğer task'larla ilişkisi yoksa paralel task (haberleşmesi gerekmiyorsa), diğer task'larla haberleşmesi gerekiyorsa seri al task denir.

Pipelining:

Shared Memory (Paylaşımlı Hafıza):

Bilgisayar üzerindeki CPU'lar bütün işlemler için aynı hafızayı kullanır.

Dezavantaj: 1. CPU'nun aynı hafızayı görüp üzerinde değişim yapabilmesi

Bilgisayarın yapısını karmaşıklaştırması

Avantaj: İşlemleri hızlandırması

Symmetric Multi-Processor (SMP):

Birden fazla CPU'nun tek bir hafıza alanını kullanması ve tüm kaynaklara eşitli paylaşılan mimari.

Distributed Memory:

Birden fazla kart, her kartta bir CPU ve hafıza vardır. Bunların aynı amaçla kullanılmak istenmesi. Bunun için network gerekli. Normalde CPU kendi hafızasını görür diğer hafızalarını göremez. Bu nedenle hafızayı diğerlerine açmak gerekir. Ek yazılım gerektirir.

Communication (Haberleşme)

En fazla zaman kaybının olduğu yerdir. Network sorun kayıpları burada ortaya çıkar. En iyi durumda 1. bit başında ve sonunda haberleşme gerekir. Paralel hesaplamanın haberleşme zamanı yüksekse yapılmaz.

Senkronizasyon:

Birbirinden ayrı olarak çözülen problem parçalarının daha sonra birleştirilmesi gerekir. Bunun için de parçaların birbirine senkronize edilmesi gerekir.

Granularity (Hassasiyet)

Hesaplama ile bulunur. Eğer hesaplama > haberleşme ise
Haberleşme

buna coarse denir. Hassasiyetin önemli olmadığı durumlarda kullanılır.

Fine: Hesaplama zamanı kısa, aralarda haberleşme yapmak gerekiyor. Zamanın çoğu haberleşmeye gidiyor.

Observed Speedup :

Yaptığımız zaman tahmini ile gerçek zaman arasındaki farktır. Bir iş seride ne kadar sürede yapılır, paralelde ne kadar sürede yapılır.

Paralel Overhead :

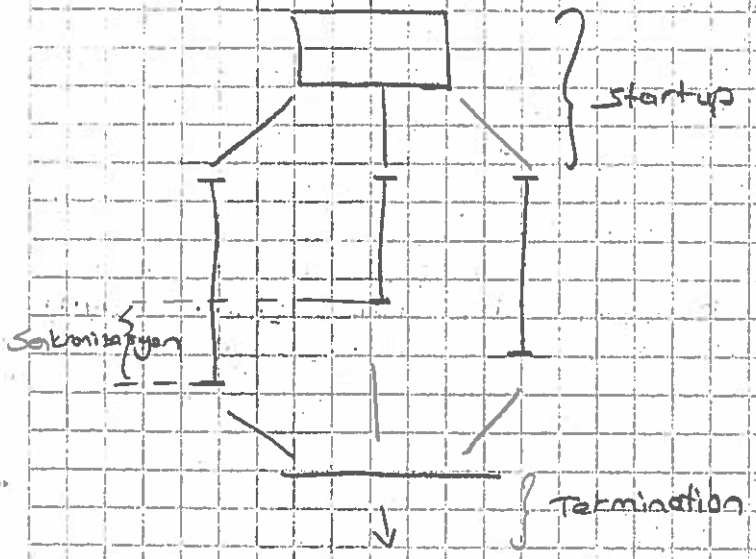
Ekstra yüklerdir Paralel Overhead'e neden olan faktörler:

→ Task start-up time : Başlatma zamanı uzun süren Değişkenlerin oluşturulması, değerlerinin atanmasından dolayı, ilk iterasyonda yüksek çıkar.

→ Senkronizasyon : n parçaya bölünen problemin 3'ü erken biri geç bitiyorsa geç bitenin süresi kabul edilir.

→ Data communication

→ Task termination Time.




Programın ne kadarı paralelleştirilebilir? Türleri var.

Massively Parallel :

Çok büyük, herşey paralel.

Embarrasingly Parallel :

Problem sayısı az, haberleşme yok.

Scalability (Ölceklenebilirlik):

Web serverlarda ölcekleme en önemli seydir. Bir program ölcek-lenebilirse gelen istege göre kendini arttırır. Ölceklenebilirlik ihtiyaca göre büyütebilmek. Örnegin 3 makine 3 günde, 1 makine yarım saatte cözüyorsa ölceklenebilirdir.

Multi-core Processors:

Cluster Computing:

Kümé bilgisayarlar. Cok sayıda bilgisayar ama süper bilgisayar degil. Clusterla birbirine baglayıp paralel hesaplama yapma. Bilgisayarları süper bilgisayar yapma.

Super Computing / High Performance Computing:

Cok büyük kaynaklı pc. 100 petabyte RAM, 200.000 CPU gibi

PARALEL HAFIZA MİMARİLERİ

1. Shared Memory:

Aynı makinede birden fazla CPU varsa tüm CPU'lar ortak hafı-zaya erişim sağlayabilirler. Yapılan degişiklik anında hepsi tarafından görülür.

Avantaj: Erişim süresi aynı, haberleşmeden dolayı zaman kaybı yok

→ UMA
→ NUMA

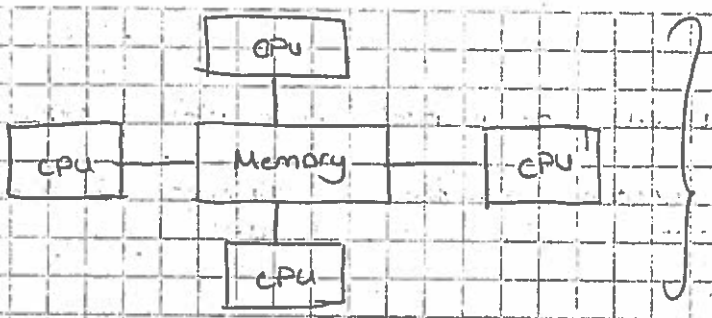
→ Uniform Memory Access (UMA)

Bütün CPU'ların özellikleri hafızaya uzaklığı (fiziksel olarak), erişim süreleri hepsi aynı.

Avantaj: Erişim süresi aynı oldugundan haberleşme kayıpları yoktur.

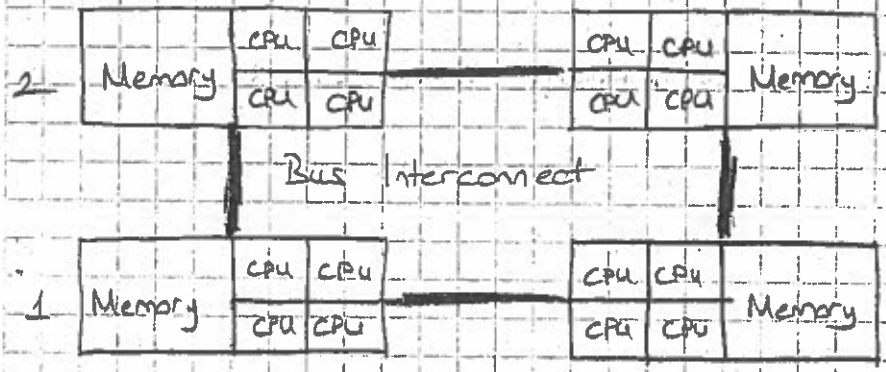
CC-UMA (Cache Coherent UMA) da denir. Cache coherent, cache tutarlılıgıdır. Hafıza update edilirse hepsi anında haberdar olur.

CPU'lar tek hafıza kullanır, Simetrik Multi Processor de denebilir. (SMP)




→ Non-Uniform Memory Access (NUMA)

4 farklı makine, 4 tane CPU, her bir makinenin ayrı hafızası vardır ve bu hafızalar birbirinden ayrıdır. Hafızalar bir bus'la birbirine bağlanır.




Bus hızı çok önemli

Her makine birbirinden bağımsızdır, birbirlerine network ile bağlanırlar.

Her makine doğrudan diğer makinenin hafızasına erişebilir. Ancak 1 nolu pcnin 2 nolu pcye erişme süresi ile 3 nolu pcye erişme süresi aynı değildir. Yani eşit sürede erişim yoktur.

Tüm makineler diğer makinelerin hafızalarını adresleyebilir.

Burada cache coherent sağlanıyorsa CC-NUMA denir.

Cache coherent yanlış sonuçların önüne geçmek için gerekir. Donanımsal olarak herhangi bir update durumunda diğerlerine haber vermesi gerekir yada belirli aralıklarda değerlerin alınması gerekir.

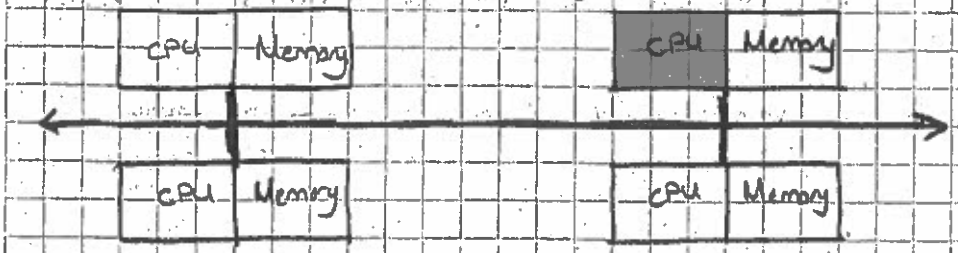
Avantaj: RAM ve hafıza arttı. Dolayısıyla çözülecek problem sayısı arttı. Global adres alanı sağlar.

Dezavantaj: Ölçeklendirilmesi zordur.

2. Distributed Memory:

İşlemciler sadece kendi local hafızasını adresleyebilir. Cache coherent yoktur.

Bir işlemci diğer işlemcinin adres uzayındaki bilgiye erişmek isterse o işlemciye bildirir. (İstek bus'a konur.)




Ölçeklenebilirdir. Ucuz bir yapıdır. Kolaylıkla oluşturulabilir bir mimaridir.

Avantaj: → Çok iddi ölçeklenebilirdir. (Ancak makine ne kadar pahalı olursa program kormaklaşır.)

→ Bus'ın hızı hesaplama hızını belirler.

→ Ucuz bir yapı

→ Herkes kendi local hafızasını hızlı bir şekilde adresler.

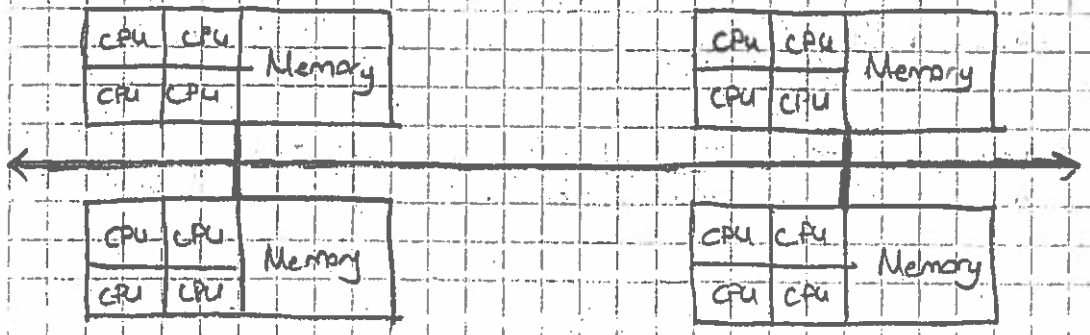
Dezavantaj: → Programcı herşeyi düşünüp kendisi yapmak zorunda.

→ Global değişkenlerin senkronizasyonu zordur. Ne kadar global değişken varsa bus trafiği o kadar pahalı olur.

→ Cache coherent yoktur.

3. Hybrid Distributed-Shared Memory

Ortak hafıza yok, herkes kendi hafızasını görüyor. İstek yapmak lazım.



Avantaj: → SMP kullanıldığından paralelleştirme yapılabilir.

→ Global değişkenleri senkronize edilmez. Gerekli global değişken için bus üzerinden istek gönderilir. Adres uzayı yönetimi yok.

Dezavantaj: Bus ne kadar hızlıysa paralel hesaplama o kadar hızlı olur.

!!! Paylaşımlı hafıza modeli NUMA ile aralarındaki fark, bu modelde diğer hafızadaki adreslerin bilinmesine gerek yoktur.

PARALEL PROGRAMLAMA MODELLERİ

Paralel hesaplama için genelde C, C++ kullanılır. Ancak en iyi diye birşey yoktur. Makine mimarisi ile programlama modeli birbirine uygun olmalı.

→ Shared Memory
→ Threads
→ Message Passing
→ Data Paralel
→ Hybrid

1. Shared Memory:

Aynı adres uzayını paylaşırlar. Klasik programlamaya göre farklı, hafızanın paylaşımlı olması. Problem parçalara bölünecek, her birini bir CPU işleyecek ve diğer CPU'larda bunu görmeli.

Semafor, kilit → Paylaşılan veriyi korumak gerekir. (Aynı anda 2 CPU bir değişkenin değerini değiştirmesin diye)

Çalışılan bilgisayar mimarilerine çok bağımlı. Global çözüm yok.

2. Thread Model:

Bir prosesin birden fazla iş parçacığı vardır. Threadler processlerin global adres uzayını, global değişkenlerini kullanır. Kendine ait hafıza ve değişkenleride vardır.
Mimariden bağımsızdır. Process içerisinde çalışır. Tek bir process gibi görünür.

Avantaj: → Threadler processlere göre daha hızlı oluşturulur. (Processler için kontrol blokları oluşturmak gerekir)

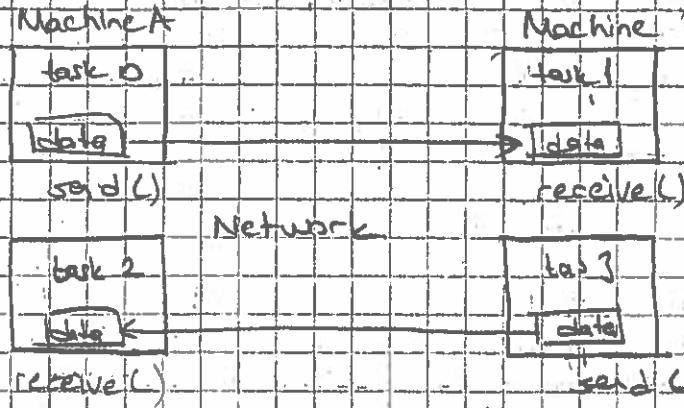
→ Global adres üzerinden haberleştikleri için zaman kaybı yok

→ Haberleşmeyi programcının düşünmesi gerekmez.

Threadlerin 2 farklı uygulaması : OPEN MP, POSIX

3. Message Passing Modeli (Mesajlaşma)

Paralel hesaplamadaki en önemli, en çok kullanılan modeldir. Mesaj bus üzerinden gönderilir. Mesaja ne istediği kaydır. İki taraftada Message Passing ile çağırması gerek. Karşıda alıcı olmalı.




Her zaman send() karşılık receive() olmalı.

Bağımsız bilgisayarlar, kendi CPU, kendi hafızası vardır. Mesajları kullanarak haberleşiyorlar. CPU'ların local hafızaları var işlemler orada gerçekleştiriliyor.

Mesaj ortak bus varsa bus üzerinden yoksa network üzerinden yapılır. Paralel hesaplamanın en çok kullanılan modeli (sıradan bilgisayarlar üzerine kurulabilir.

Programcı sadece paralelleştirmeyi yapar, mesajlaşma kütüphane ile MPI → En yaygın olan)

4. Data Paralel Modelle

Paralel hesaplamanın en önemli kısmı verilerin eşzamanlı işlenmesi üzerinde gerçekleşir. Veriyi işleyecek programları paralelleştirilir. Veri parçalanarak işlem yapılıyorsa data paralel modelidir. Örn; dizinin her elemanına 1 ekleme gibi. Tek program vardır. Program döngüyü parçalıyor. Program yazarken döngünün yanına yazılır. Sistem onu CPU'lara eşit böler. (sistem-MPI)

Cluterda kaç makine varsa o sayıya böleyi işi otomatikleştirmek amaçtır. Paralel constructlar (parale while, for)

Factron 90-95 → Paralel hesaplamada önemli yere sahip.

PARALEL PROGRAMLAMA TASARIMI (4. Slayt)

Paralelleştirme 2 şekilde olabilir

→ Otomatik → Manuel

→ Otomatikleştirme Problemin niteliklerinden dolayı önemli ölçüde gerçekleştirilemiyor. Karar programcı verir. Programı paralelleştiren compilerdır. Compiler kaynak kodu analiz edip paralelleştirilen yerleri otomatik olarak bulur. Bu işlem bir döngü şeklinde yapılır. Bu işlemden sonra paralelleştirmenin mantıklı olup olmadığına bakılır. 10 milyon elemanlı dizi olursa ... Daha sonra program derlenip çalıştırılır. İş karmaşık değilse bu yöntem bazen sağlar.

→ Manuelde paralelleştirilecek yerlere programcı karar verir. Programcı paralelleştirilecek yerlere kendi karar verir. Ne derece faydalı olacağını programcı hesaplamalıdır. Karar verirken programcı domaini iyi anlamalı. Paralelleştirme uygun olur mu? problemde buna bakılmalı.

Örneğin; bir molekülün etrafındaki diğer moleküllerle farklı bağ oluşturma ihtimalleri vardır. İşler birbirinden bağımsız ise %100 paralelleştirilebilir. (Paralel oluru)

Örneğin; Fibonacci serisi hiçbir zaman paralelleştirilemez.

Program ve Problemi Anlamak

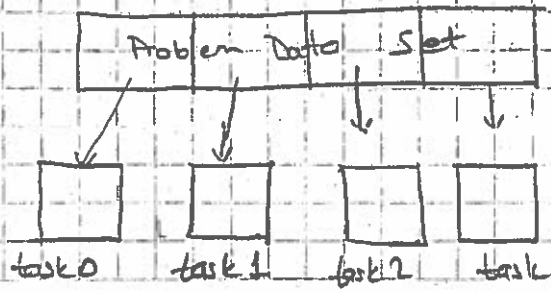
Hotspots: Programın profilini, temel noktalarını belirlemek. Profiler ve performans analizi yardımcı olur.

Bottlenecks: Programın yavaşladığı yerler. Bunlar paralel hesaplamanın yapıldığı yerlerin dışına atılmalı. Örn; I/O işlemleri programı yavaşlatır.

Partitioning: Programı ayrık parçalara bölmek 2 şekilde yapılabilir.

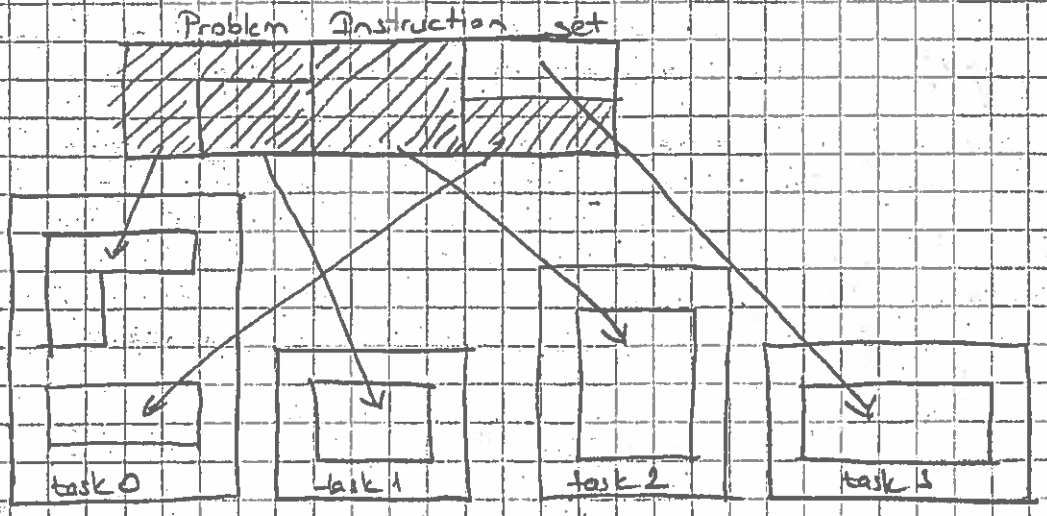
→ Domain Decomposition:

Veriyi parçalayıp parçaları işleme. İlk önce verinin bir kısmı işlenir, sonra geri kalan kısımları bölünür. Veri parçalara ayrılır. Her paralel görev veri bölgesinin bir kısmı üzerinde çalışır.



→ Functional Decomposition

Programı komutlarını parçalara bölme. Fonksiyonları, metotları, döngüleri bölme.



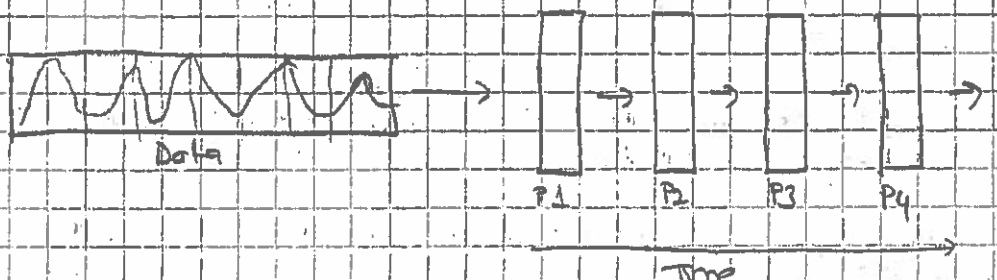

Problemi farklı görevlere parçalamak.

→ Ekosistem Modeli - Functional Decomposition

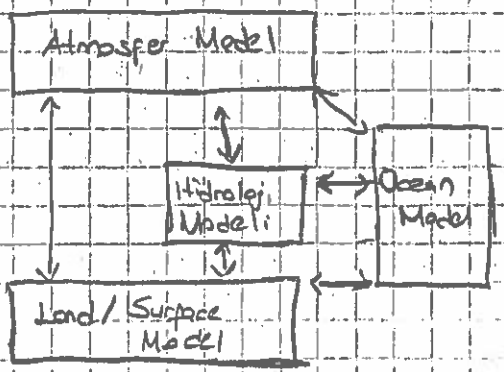
Başlangıç değerleri vardır. Bu başlangıç değerleriyle olgunlaşma durma noktaları vardır. Her birinde farklı modeller vardır. Fakat sonuçları bir önceki ve bir sonraki yapıyı haberdar eder.

→ Sinyal İşleme - Functional Decomposition

4 filtre var. 10 sn'lik kumu 1. filtreye verilir. Daha sonra 2., 3. ve 4. filtreye verilir. 4 tane process her biri farklı iş yapıyor, ancak bir problemin parçası.




→ Klima Modeli - Functional Decomposition




Atmosfer model rüzgar hızını üretiyor. Okyanus modeli alıp işliyor. Deniz yüzeyi modeline veriyor. Deniz yüzeyi modeli tekrar atmosfere veriyor. Birbirinden bağımsız işlemler.

PARALEL PROGRAMLAMA TASARLAMA (6. SLAYT)

Haberleşme maliyetini farklı lokasyonlardaki bilgisayarlar arttırır. İnternet, LAN gelirse haberleşme yavaşlar. Hesaplama 10 ns iken haberleşme 100 ms olur. Paralel hesaplama yavaşlar.

Paylaşımlı hafıza ile paralel hesaplama yapmak gereklir. (Süper computer pahalı) İlk önce haberleşmeye bakmak gerekir.

Problem parçalara ayrıldığında ara sonuç yoksa bunlara embarrassingly parallel denir.

Göz Önünde Bulundurulması Gereken Faktörler:

Haberleşme Maliyeti: En önemli olan şey görevler arası haberleşme maliyeti. Networku kaplıyor.

Latency vs Bandwith:

latency, A noktasından B noktasına mesaj göndermek için gereken zaman haberleşme gecikmesi

bandwith, birim zamanda gönderilecek veri.

Visibility of Communication:

→ Message Passing de mesajlar nereden gelmiş nereye gitmiş programcı bilir.

→ Data Paralel de programcı bilmez. Bant genişliğinin dolması, latency burda ortaya çıkar.

Senkron ve Asenkron Haberleşme:

Senkron karşılıklı haberleşmedir. Karşı taraf mesajı alana kadar bekler. Dezavantajı bloke etmesidir.

Asenkron mesajı gönderip işine devam eder. Avantaj: non-blocking. Diğer işlerini yapmaya devam eder.

Scope of Communication:

Haberleşme 2 şekilde olur.

→ Broadcast: Mesaj ağdaki herkese gönderilir. Hızlıdır, ancak güvenli değildir.

→ Scatter: Mesaj sahibinin posta kutusuna bırakılır. Güvenlidir.

Haberleşmenin Verimliliği

Haberleşme yavaş ve masraflıdır. Bir noktaya kadar ölçeklenebilir. Örneğin; bant genişliği 400 Mbps ise bunun 300'ü haberleşmeye kalabilmek. Proses kilitlenir, iş yapamaz hale gelir. MPI bunu önceden hesaplamaya çalışır.

Paralel hesaplamadaki kayıp zamanı haberleşmeden kaynaklanır.

MPI - Init

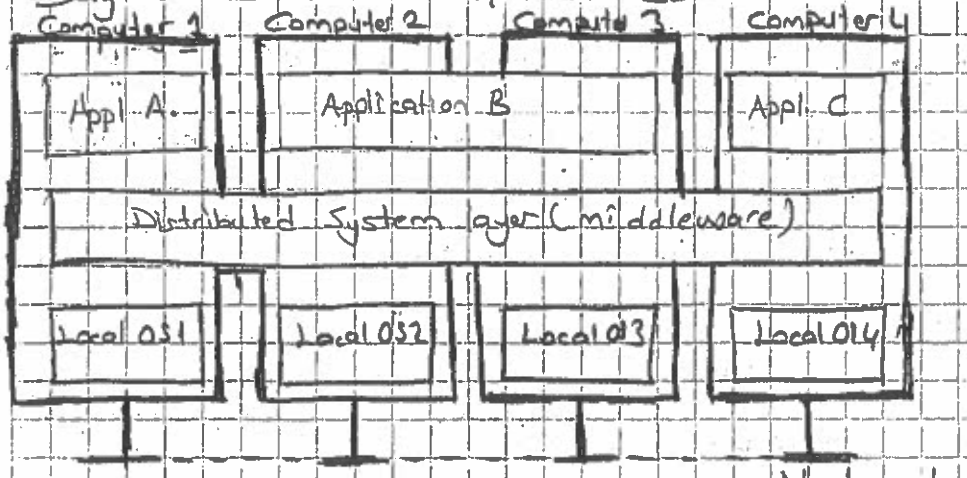
MPI - Rank

MPI - Size

# DAĞITIK SİSTEMLERİN TANIMLANMASI

Kullanıcılara tek bir sistem gibi görünen, ancak farklı coğrafi yerlerde olan pc'lerden oluşan sisteme dağıtık sistem denir. Tamamen bağımsız tek başına işlem yapabilen pc'ler. Distributed memory'le yakın. 3 pc'yi bir araya getirip dağıtık sistem kurulabilir. Paralel hesaplamadaki farkı budur. Paralel hesaplamada işler donanıma kalır. Dağıtık sistemlerde öyle değil.

Dağıtık Sistem Oluşturma Yöntemleri




Middware → Orta Seviye Katmanı

4 pc farklı yerlerde hepsinin işletim sistemi farklı. Bunları biraraya getirmek için haberleşmeyi sağlayacak bir katman lazım (Paralel hesaplamada pc'lerin aynı olması lazım) Buradaki yapı gevşek (İstenilen pc sisteme eklenebilir)

4 pc'yi birbirine bağlayıp tek bir pc gibi kullanmak bağlamında internetle olur. Middleware işletim sistemi üzerine kurulur. Bu katman üzerinden haberleşme olur. Bu sistem şeye=

sinde networke bağlı diğer PC'lerden haberdar olunur.

Dağıtık sistemin bir arayüzü olur. Bu arayüz arka taraftaki 4 pc'yi tek gibi gösterir. Arkada gridi için özelliğine göre sistemde boşta olan pcler üzerinde hangisinde çalışacağına karar verir. (uygun olanı seçer.) Her pc'nin anlık durumu görünür. Sistemdeki pc'lerden birinin gitmesi durumunda; diğerleri onun görevini devralır, sistem çökmez. Torrentler...

Public bir sisteme herkes girebilir. Değilse hesap gerekir. İdeal sisteme zaman içinde yaklaşılmıştır.

Önemli olan middleware'in özellikleri ve ne kadarını yapabildiğidir. Kullanılacağı yere göre özellikleri değişir. Bir veritabanının yedeğinin başka bir yer üzerinde tutulması pahalı bir yapı.

Transparency: Bazı özelliklerin gizlenmesi. (Tek pc olarak göstermek)

→ Access: Erişimin gizlenmesi. Arka plandaki kaynakların gizlenmesi. Asıl amaç, tek bir pc gibi göstermek. Global işletim sistemi kurmak en uç noktada yapılmak isteyen; hiçbir şekilde çökmeyecek. Bilim adamları için global işletim sistemi oluşturmak internet üzerine kurulmuş sistemler var. Örn: Grid, web arayüzü var. Kendi yaptığımız uygulamalar, işler girildiğinde görülebiliyor. Kaynaklara erişimin gizlenmesi gerekiyor.

→ Location: Kaynağın nerede olduğunu, konumunu gizleme.

→ Migration: Bir kaynağın başka bir yere taşındığını gizleme. Kullanıcı görmemeli. Veritabanları elazığla taşınmalı.

→ Relocation: Sunucuların yerini görmemeli ve bilmemeli.

→ Replication: Kaynağın çoğaltılmasını gizlemek.

→ Concurrency: Kaynağa eşzamanlı erişimin gizlenmesi. Alışveriş sitelerinde başkalarının siparişlerinin görünmemesi.

→ Failure: Sistemdeki hataların gizlenmesi. Örn: Veritabanının çöktüğü son kullanıcıya söylenmez.

→ Flexibility: Esneklik. Sistemi oluştururken yeni kaynak eklendiğinde ne olacak? Sistem daraltılıp genişletilebilir mi? Mikro kernel modeli. Sistemin yeni kaynak eklenmesi, çıkarılabilmesi durumunda ne kadar esnek olursa kullanması o kadar kolay olur.

→ Reliability: Güvenilirlik. Sisteme bir iş yaptırırken sisteme güvenilmeli. Sistemin çökmeyeceğinden emin olunmalı.

→ Performans : Dağıtık sistemin temel sebebi performans. Elimizde olmayan kaynakları kullanma. HPC → Hyper Performance Computing

Scalability :

Ölçeklenebilirlik. İhtiyaç oldukça sistemin kaynaklarını arttırma. 100 kişi için oluşturulan sistemin gerektiğinde otomatik olarak 1000000 kişiye yanıt verebilmesi. Gerekli donanımların otomatik sisteme alınması çok zor bir iştir. (3-4 yıl önceye kadar mümkün değilken şimdi bulut bilişimle rahatlıkla yapılıyor.

→ Serverlarda ölçeklenebilirlik istenir. (web ve application server) Web server kiralandı, sayfaya çok istek geldiğinde trafiği boldurmuyorsa data center'da trafik arttırılır. Bir noktadan sonra bu da problemi çözmez. Yük kontrol edilip otomatik server devreye alınmalı.

→ Merkezi Servis Yaklaşımı : Herkes aynı yere erişip işlemini yapıyor. Tek server üzerinde işlemleri yapmak. Yerine göre doğru bir yöntem. (Üni için tek server) Tüm ünilerin otomasyonunu tek serverda tutmak mantıklı değil. Bunun için aynı anda binden fazla server olacak. 3 makineye kurup gelen istekleri paylaştırmak gerekir. (Merkezi olmaktan çıkarılır.) Merkezi sistemden uzaklaşmak her zaman yapılabilecek bir iş değildir.
Mainframe'ler bankalarda önemli ölçüde işler üzerinde tutar. Ölçekleme açısından temel problem geliştirilmesi kolay değildir. (19 milyon data) Bir mainframe CPU, RAM bir yere kadar arttırılabilir. Bir bankaya çok fazla. Centralized servislerin temel problemi ölçeklenebilirliktir.

→ Client - Server Modeli : Mainframe'lere göre daha ucuz. Yük dağıtılabilir, belli bir sayıya kadar sunucu eklenebilir. Ölçekleme limiti network'le ilgili. Trafik belli bir limite ulaştığında ek server alınamıyor. En çok kullanılan model client-server modeli. Client - server'da sunucu ve istemci açık şekilde tanımlanmıştır.

→ Peer-to-Peer Modeli : Tüm bilgisayarlar birbirinin eşi. Peer'ler hem sunucu hem istemci olabilir. Dosyalar peerlere dağıtılıyor. Hem dosya sunup hem de client olarak diğerlerinden dosya çekilebiliyor. Hizmet sunumu için uygun değil. Ölçeklemeyi kolaylaştırır. Depolama miktarını arttırabiliyoruz. Hizmet sunmada algoritmayı peer-to-peer paylaştıramıyoruz.

→ Kalın ve İnce İstemciler : İnce istemci merkezi sunucuya bağlanıp sonuçlar ince istemcide gösteriliyor. 50 kullanıcı server'a bağlanıp uzak masaüstü ile

→ Çok Katmanlı Mimari: Değişik işler yapan sunucular var. Yükü mümkün olduğu kadar çok sunucuya dağıtma. Web sunucu ile database sunucusu ayrı yerlerde birbirleriyle haberleşiyorlar. Web serverlar 2 katmanlıdır. 3 katmanlı mimarilerde iş göster, hesapları yap, veritabanını tut (yaygın olanı) sunucular arka planda dağıtılıyor. (En yaygın olarak kullanılan hizmet sunumu için) Sayı olarak peer-to-peer fazla. Bunlar merkezi modele bağlanıyor. Bir tane merkezi sunucu var. Yük miktarı artınca belli bir yere kadar ölçekleyebiliyoruz.

Merkezi Olmayan (Decentralized) Sistemlerin Problemleri:

Hatadan kurtulmak önemli - Birden fazla pcde hata olsun

→ Hiçbir bilgisayar tam olarak sistem durumu hakkında bilgiye sahip değildir. Kısmi olarak bilinir.

→ Bilgisayarlar dosya paylaşırken sadece local bilgiye göre karar verirler.

→ Bir pcnin bozulması sistemi etkilemez. Bozulanın üstündeki veri bir diğer pcde de vardır. Örneğin, No SQL, dağıtık dosya sistemleri

NOT: Centralized sistemlerde cluster yapılsa bile veri parçalanamaz.

Ölçekleme yapmak için kullanılan methodları:

Client-server modellerinde formun kontrolü server tarafında kontrol edilebilir. Burada gereksiz işlem yapılır. Form kontrolü client tarafında yapılmalı.

DNSler her networkte bir tane bulunur. En yakın DNS server cevap verir. Ölçeklenebilirliğe güzel bir örnek. Network genişledikçe genişleyen yere bir tane DNS koyup diğerleri ile haberleşmesi sağlanır.

Dağıtık Sistemler Tasarlanırken Bazı Varsayımlar Yapılır:

→ Networkün güvenilir olduğu (İşi sonuna kadar yapması)

→ Networkün güvenli olduğu (Kullanıcı adı, şifre ile girilmesi)

→ Networkün homojen olmasını kabullenme

→ Topolojinin değişmediğini varsaymak

→ Latency (ağ gecikmesi) sıfır kabul etmek.

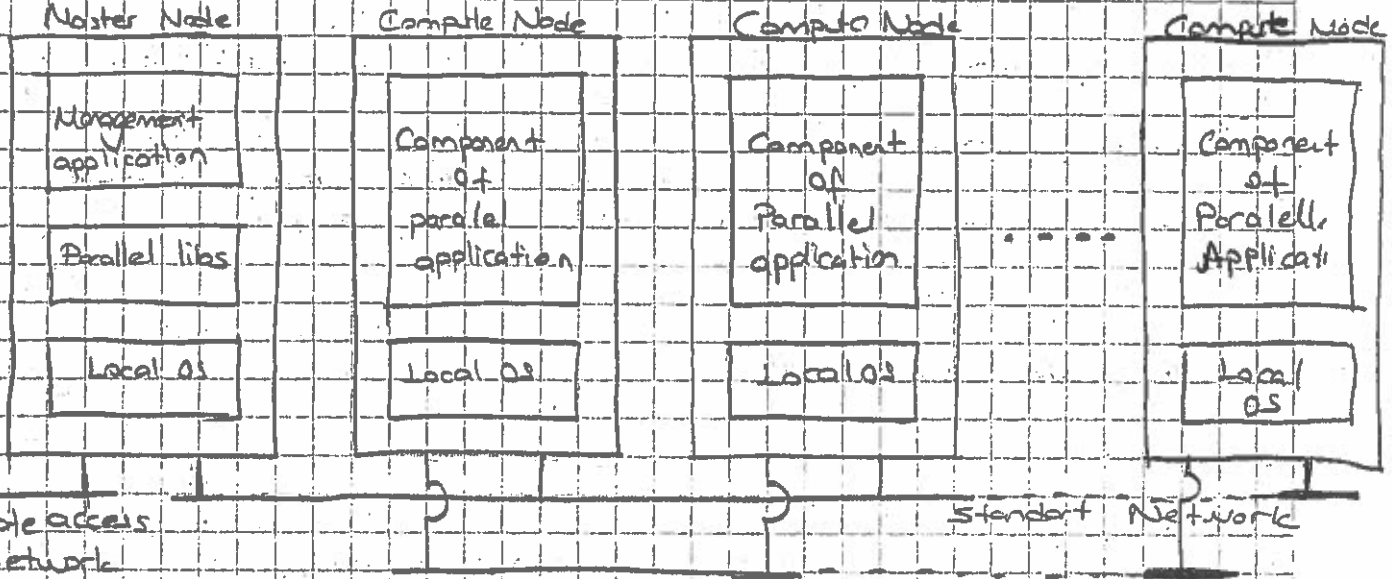
→ Bant genişliğinin sonsuz olmasının varsayılması

→ Bir yerden bir yere taşıma maliyetinin sıfır olduğunu kabul etmek

→ Tek bir yönetici olduğunu kabul etmek.

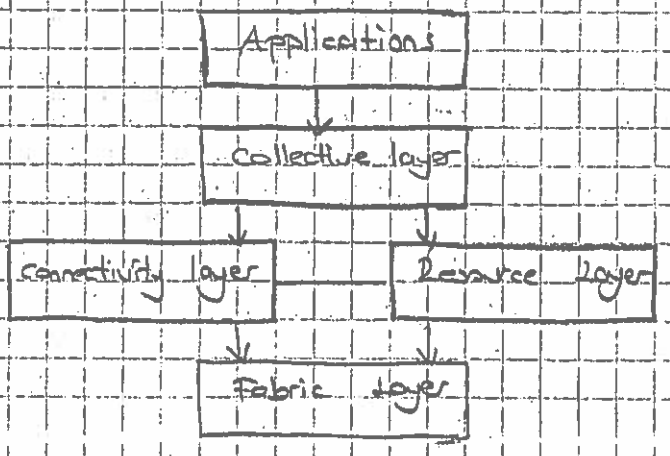
Bu noktalar mutlaka göz önünde bulundurulmalı. Bunlar çözümlenerek dağıtık sistem kurulmalı.

## Cluster Computing Systems (Kümeleme Bilgisayar)




Aynı türden pcler kullanılmalı, pcler birbirine bağlı olmalı. High speed network veri değişimi için kullanılır. Standart network iş göndermek, iş almak, web arayüzü çalışır. Bir tane yönetici node (master node) işi parçalamak, sonuçları toplamak için vardır. Paralel hesaplamanın parçası, diğer pc'lerdedir. Management node işi parçalara böler, yapabilecek pclere gönderir ve geri alır. Pcler aynı mekandadır.

## Grid Computing System :




PC'ler dünyanın her yerinde olabilir sanal organizasyonlar oluşturulur.

Fabric Layer : CPU, RAM, PC'yi kontrol eden ve yöneten katman.

Connectivity layer : Protokoller, nasıl erişilecek.

Resource layer : Hangi pc ne kadar meşgul, ne kadar CPU kullanıyor.

Collective layer:

Application layer: Bir programı collective layera verir. O dağıtır.

Clusterdan farkı pc'ler farklı yerlerde. Hizmet odaklı mimarinin temel adımı.

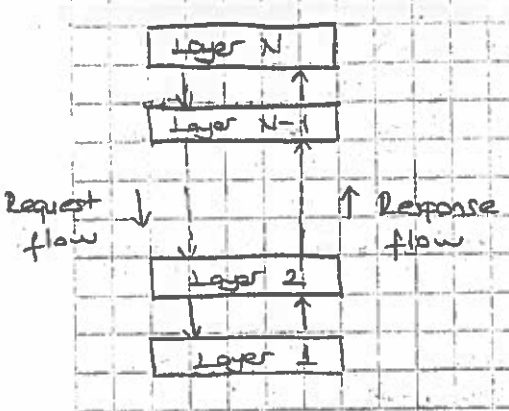
MİMARİ MODELLERİ:

4 mimari var.

- Katmanlı Mimari
- Nesne Tabanlı Mimari
- Veri Merkezli Mimari
- Olay Tabanlı Mimari

} Farklı zamanlarda geliştirilen farklı kullanım alanları vardır.

→ Katmanlı Mimari



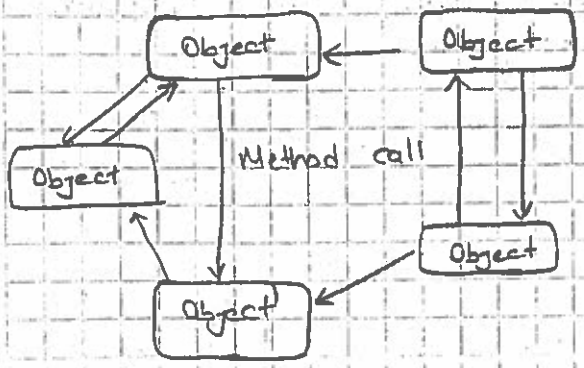

Aşağıdan yukarıya istek, yukarıdan aşağı cevapların geldiği model

Avantajları:

* Her katmanın ne iş yaptığının açık olduğu yerlerde kullanılır.
* Dağıtık veritabanının olduğu yerlerde kullanılır.

Dezavantajı: Katmanlar arası haberleşme tek yönlü. Aradaki katman atlanamaz.

→ Nesne Tabanlı Mimari

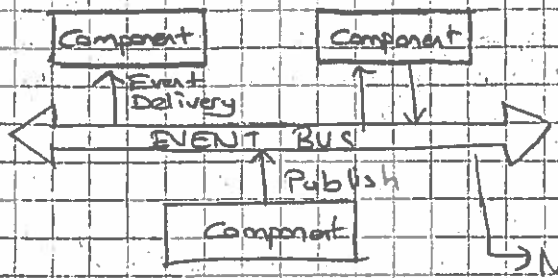



Eldeki programı başka pc'ye kullandırmak Amaç, başka makinelerdeki nesneleri kullanabilmek. RMI, RPC yazılımı ile yapılabilir.

Client-server mimarisine benzer. En yaygın kullanılan modeldir.

Veriyi paylaşan sistemler için geliştirilmiş modeller (Veri Tabanlı, Olay Tabanlı)

→ Olay Tabanlı Mimari




Event bus var. Ortak veri yolu. Bağlı pcler bustan geçen herşeyi görürler. Bir olay olduğunda busta mesaj üretilir.

Publish subscribe modelleri, son yıllarda geliştirilen en çok kullanılan sistem.

→ Mesajın ulaşıp ulaşmadığından emin olunamaz.

Bir broker çeşitli yerlerden veri gelebiliyor. Her biri farklı konuya yayınlanıyor. Müşteri olarak yayınlara üye olunuyor.

Kullanım alanı farklı (web server çalışamaz.

Avantajı: Çok fazla mesaj üretiminin olduğu sistemlerde kullanılıyor.

Problem:

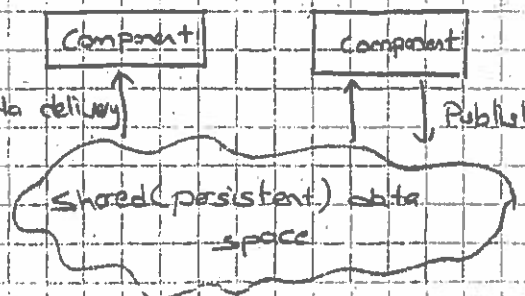
Publish yapan kişi subscriberlardan haberdar değil. Mesajı gönderen yayına üye olan kimselerden haberdar olmaz. Buna dikkat in ---- denir. Bu sağlanırsa üyelerle ilgilenmek zorunda olmaz.

→ Mesajların yerine ulaşıp ulaşmadığı bilinmez. Hassas işler yapılıyorsa bunun için ek birşey eklemek gerekir.

→ Kayıp mesajları subscriber ile broker arasında gelen mesajların numaralarını bildirmesi gerekir.

Bu problemlerden dolayı event bus server a dönüşüyor.

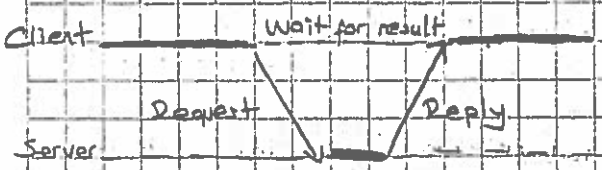
→ Veri Tabanlı Mimari




Haberleşme paylaşımlı veri uzayında yapılır. İşler dosyalar üzerinden yapılıyor. Veriler dosyalar üzerine yazılır.

4 farklı dağıtık sistem modeli. Bunları gerçekleştirirken 2 temel mimari var.

## 1. Centralized Architectures




Serverların yaptığı işler tanımlıdır. Her server bir hizmet verir.

Server başka serverlara clientlik de yapıyor olabilir. Serverın yaptığı iş tam olarak belli ise server hizmeti sunabilmek için diğer serverlara clientlik yapabilir. (Decentralized mimari)

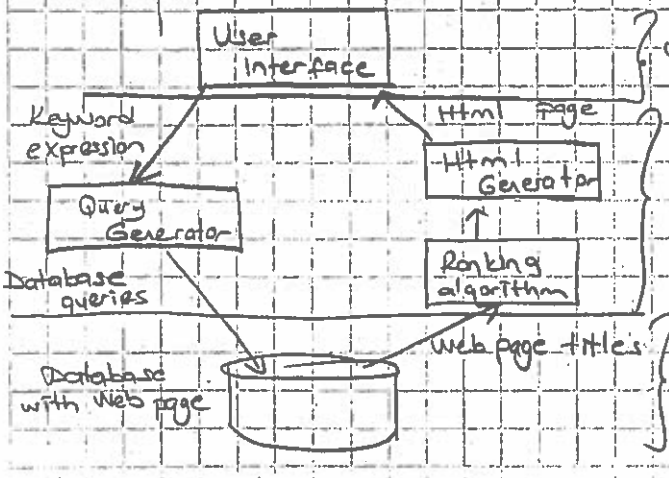
Client-server arasında bağlantının güvenilir olduğu düşünülür. İstek yapılır, bağlantı kesilir. Server adrese göre cevabı gönderir. Bağlantısız dedir. Avantajı çok sayıda clientın servera bağlanabilmesi.

Problem: Hesaba para yatırma isteğin yerine ulaşıp ulaşmadığı kontrol etmek gerekir. Pratikte mümkün değil. Belli bir zaman içinde ulaştığı ile ilgili mesaj gelmezse tekrar gönderir isteği. (1. isteğin id'si ile 2. gönderilir.) İşleri karmaşıklaştırır.

Temel olarak kabul edilen 3 katman vardır.

→ User Interface Katmanı: Web arayüzünde görünenler, kullanıcılar.

→ Processing Katmanı: En altta bir veritabanı. Örnek: google, arama motorları.




→ Data Katmanı

Çok Katmanlı Mimari : 2 katman var. Biri client diğeri server. Katman sayısını tam olarak belirtmek mümkün değildir. İstek cevap döngüsü vardır.

Doğrulama kullanıcı tarafında.

2) Decentralized Mimariler :

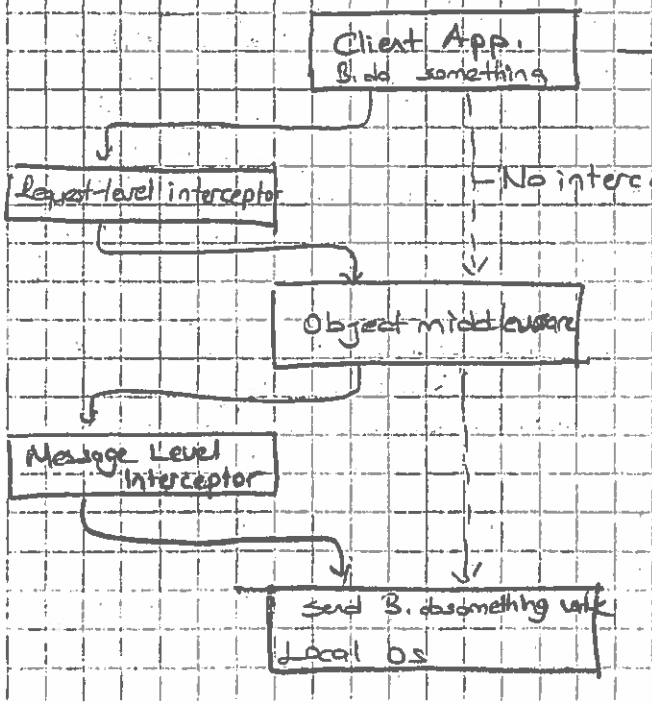
Server ile client arasındaki ayrım net olarak yapılamaz. Bir makine hem server hem client olabilir. Bunlara merkezi olmayan mimari denir. Ör :

Peer-to-Peer Networkler : Makine hem server hem istemci. Networke üye oluyor. Hem dosya paylaşan, hem de dosya indiren bir makine oluyor. Makineler eş olarak bakılıyor. Buna peer-to-peer (eşler arası paylaşım) denir. Asıl önemli olan distributed hashler. Hızlı çalışan mimarilerdir.

Yapısal olan ve yapısal olmayan yaklaşım var. Her node'un unic id'si var. Üye olunca otomatik id veriliyor. Dosyanında unic id'si var. Dosya id ile unic id eşleştiriliyor ve hashtablede saklanıyor. Tek tek sorup bulma.

ÖDEV : Peer-to-Peer Networklerde structer-unstructer nasıl çalışıyor? Distributed hashtable nasıl çalışıyor, bunun için geliştirilen yaklaşım ?

Interceptors :




Buradaki amaç hizmet kalitesini yükseltmek.

Dağıtık sistem üzerinde elimizdeki kaynakları kullandırmak için farklı yaklaşımlar var. Kodu yeniden kullandırmak amaç.

Remote-object invocations: Mesajların arasına girmek. Katmanların arasına girmek.

Web'de kullanılıyor. Amaç hizmetin kalitesini yükseltmek. Görselleştirme, simülasyon varsa bunları başka katmanlara yaptırmak lazım. Modülerliği sağlıyor. Server sadece simülasyonu yapmaz, sadece cevap verir. Interceptorlar kendilerini client'a gizlerler.

Centralized ve decentralized'ın geliştirilmesi için bazı yaklaşımlar düşünülmüş. Adaptif sistemler, otomatik ayarlıyor kendini.

→ Separation of Concerns: Yapılmak istenen işler birbirinden ayrılır. İşlemler birbirlerinden ayırt edilecek katmanlara bölünür.

→ Computational Reflection: Runtime anında değişiklik yapma (reflection) yansıtma yöntemiyle yapılır. Diğer çağırmak yerine aynı nesneyi kullanan başka programdan çağırarak yapılıyor. Programı çalışırken modifiye etme. Ortama adapte etme. Java da var.

→ Component-based Design: Programı parçalara ayırma. Veritabanı kısmını ayırma. Hesaplama kısmı ayrı model, user-interface kısmını ayrı bir model yapma.

Çalışmama Nedeni:
Bu 3 şart tam olarak sağlanamamış. Reflection RPI tüm değişikliklere izin vermiyor. Problem birbirinden ayrılamayabiliyor her zaman.

Otonom sistem: Kendilerini otomatik olarak ayarlayan sistemler.

PROCESSLER

Bir programın derlenip RAM üzerine yüklenmesi ve processlerin sırasıyla CPU'da çalıştırılması. Process kontrol bloğu oluşur. Birden fazla CPU gibi kullanım multiprocessing denir.

Avantaj:

Dezavantaj: Her bir processin oluşması için bazı şeyler gerekiyor. Ayrı bir hafıza, zaman alıyor, hem de pahalıya mal oluyor.

Thread: Processleri küçük parçacıklara bölme. Threadler kendi processlerinin alanlarını, hafızalarını kullanır. Threadler 2 seviyede oluşturulur: User ve Kernel

User: Kullanıcı her türlü şeyi gerçekleştirebiliyor.

Kernel: Her process için birden fazla thread oluşturuluyor. I/O olursa threadlerden birine yönlendiriyor, diğerleri bloke olmuyor. Bu her iki modeli kullanan birçok hibrit sistem var.

Lightweight process: Hibrite benzer, şuanda kullanılmıyor. Kernel tarafında lightweight process denilen (biraz process, biraz thread) oluşturuluyor. I/O bunlardan birine yönlendiriliyor.

Thread paralelliği, paralel iş yapabilmeyi sağlar. Bir işin parçalara bölünmesi için programlama dilinin bunu desteklemesi lazım.

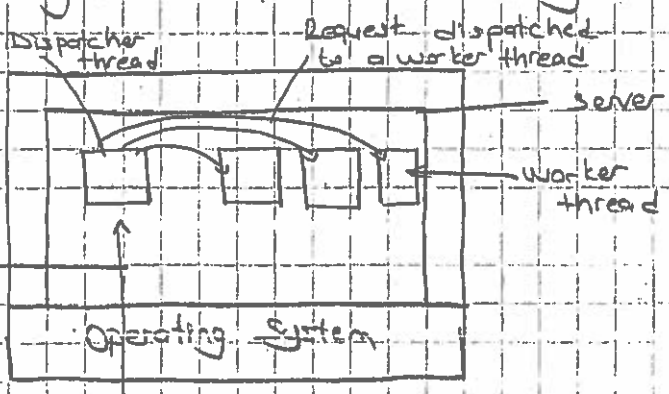
Excelde yazı yazarken hesaplamaların yapılması diske kaydedilmesi, grammer kontrolü yapmak için birden fazla process çalıştırmak yerine bir process için belli bir alan ayrılıp bu işleri yapması için threadler oluşturuluyor. DOS'ta yazı yazarken başka birşey yapılamıyor.

Serverlar, clientlar multithreaded şeklinde yazılıyor.

Multithreaded Servers (1)

Clientten gelen isteğe cevap vermek. Daha önceki isteklere cevap vermesi, diğer istekleri düşürmemesi lazım. Bunun için serverda birden fazla thread (10) oluşturuluyor. Bu threadler gelen istekleri cevaplasın (Basit çözüm)

11. istek gelirse 10 thread cevap veremez. Thread sayısı sınırlı. Bunun yerine başka çözüm geliştirilmiş.




Dispatcher thread isteği worker threade iletiyor. Kendisi dinlemeye devam ediyor. Network haberleşmesi yapan tek thread var. Dispatcher 80 nolu portu dinleyip worker threade veriyor. Workerlar hepsi aynı birbirinin kopyası. Dağıtık sistemlerde birçok yerde kullanılır.

Client tarafında tarayıcı açıp sayfanın yüklenmesi sırasında html kodları haricinde java scriptler, csslerin yüklenmesi gerekir. (Tek thread olmasından dolayı böyle) Bunu çözmek için client tarafında multithreaded kullanılmaya başlandı. Head kısmında harici dosyalar varsa bir tane thread bunları getirmeye başlar. Geçici bölgeye yükleyor. Tarayıcı bu resimlerin hepsini aynı anda yüklüyor. (Birden fazla thread) Bir sayfayı hızlı bir şekilde yüklüyor.

Process Çalışması

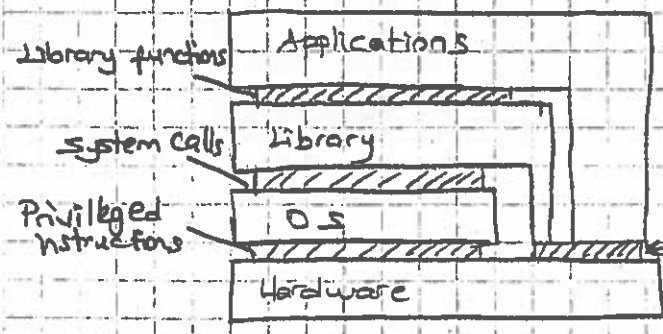
Process
↓
RAM de yer ayrıldı.
↓
Process Kontrol Bloğu Oluştu
↓ → CPU Dispatcher CPU'ya verdi processi. Birden fazla process aynı anda yükleyebilir.
Processi alan CPU sıryla bunları çalıştırdı.

Sanallaştırma:
İşletim sisteminin kullanacağı max kaynak miktarının bir adım ötesi. Aynı anda birden fazla İşletim sistemi donanımdan bağımsız olarak yazılımı donanım üstünde çalıştırmak. (Virtualization)
Sanal processler oluşturup, bunu tek bir process gibi çalıştırmak. Birkaç farklı katmanda yapılabilir.

→ Birinci Seviye: Donanım ve yazılım arasında bir interface

→ İkinci Seviye: sistem çağrıları. İşletim sistemi sadece yapar. İşletim sistemi hakkında programların sistem çağrısı yapması için interface

→ Üçüncü Seviye:
Privileged instructions, kernel seviyesinde (Driverlar)




→ 4/5. Seviye: Virtual Machines Monitore, donanımı gizleyecek katman. Donanımla haberleşme yeteneği olacak. Üzerine yazılımlar kurulacak. Bu yazılımlar donanımı kullanıyormuş gibi olacak. Bir donanım gibi hareket edecek. İstediğin kadar işletim sistemi kurabilirsin. Virtual Machine Monitore ile ilgili ilk makale 2001 de yayınlanmış.

Avantajları:

→ Sistem çağrıları, interruptlar işletim sistemini boş bekleten şeylerdir.

→ Bir makine kullanılabilecek kadar kullanma.

5'i gerçekleştirmek için donanım desteğine ihtiyaç var. 64 bit adresleme uzayı.

Hizmet olarak sunulabilir mi? (Cloud Computing) Alt yapıyı servis gibi sunmayı yapıyor. Donanımı hizmet olarak sunuyor.

Cloud Computing ihtiyaç olunca sanallaştırmayı otomatik yapıyor.

Amazon → sanallaştırma

# VERİ MADENCİLİĞİ YILİÇİ SINAVI

Fen Bilimleri Enstitüsü Yrd. Doç. Dr. Şule Öğüdücü

## Çözümler

1. Veritabanı ve veri madenciliği işlemleri arasındaki farkları yazınız. Örnekle açıklayınız.

   **Cevap:**

   - Veritabanı sorgulamaları tanımlıdır. Veri madenciliği işlemlerinde aranan bilgi veri içinde gizli olduğundan tanımlı bir sorgulama yoktur. Veritabanı sorgulamaları için SQL sorgulama dili kullanılır. Veri madenciliği işlemlerinde kullanılan yaygın sorgulama dili yoktur.
   - Veritabanı işlemleri canlı veri üzerinde yapılır. Veri madenciliği işlemleri, üzerinde işlem yapılmayan veri üzerinde yapılır.
   - Veritabanı işlemlerinin çıkışı bellidir ve verinin bir altkümesidir. Veri madenciliği işlemlerinin sonucunda elde edilecek bilgi belirgin değildir.
   - SQL sorgulamasına örnek: DVD satın alan tüm müşterileri bul.
   - Veri madenciliği işlemlerine örnek: DVD birlikte sıkça satın alınan ürünü bul

2. Veri madenciliği uygulamalarında kullanılan modeller kaça ayrılır? Açıklayınız.

   **Cevap:** İkiye ayrılır.

   - kestirime dayalı: sınıflandırma, eğri uydurma, zaman eğrileri
   - tanımlayıcı: demetleme, özetleme, bağıntı kuralları, sıralı diziler

3. Bir veri madenciliği uygulaması tasarlanacaktır. Çalışılacak veri kümesindeki niteliklerden biri *yaş* bilgisidir. Bu niteliğe ait değerler şu şekildedir: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 30, 45

   (a) *yaş* niteliğine ait veri için gürültüyü azaltmak amacıyla eşit aralıklarla bölme genişliği 3 olacak şekilde bölmeleme (*binning*) yapılacaktır. Bu işlemi yapmak için gerekli adımları gösteriniz. Yukarıdaki veri için bu işlemin etkilerini yorumlayınız.

   **Cevap:**

| Bölme No | Veri | Ortalama | Alt-Üst Sınır |
|---|---|---|---|
| 1 | 13-15-16 | 14,7-14,7-14,7 | 13-16-16 |
| 2 | 16-19-20 | 18,3-18,3-18,3 | 16-20-20 |
| 3 | 20-21-22 | 21-21-21 | 20-22-22 |
| 4 | 22-25-25 | 24-24-24 | 22-25-25 |
| 5 | 25-30-45 | 33,3-33,3-33,3 | 25-25-45 |

   Aynı değere sahip veriler farklı bölmelerde yer aldıkları için farklı değerler alıyorlar (Örnek: 16 hem 1. hem de 2. bölmede var. 1. bölmede ortalama kullanıldığında 14,7 değeri ile, 2. bölmede 18,3 değeri ile düzeltiliyor). Ancak alt ve üst sınır kullanılarak düzeltme yapılırsa bu problem ortadan kalkıyor.

   (b) Verideki aykırılıkları belirlemek için hangi işlemler yapılabilir?
   **Cevap:** Demetleme

(c) Verideki gürültüyü düzeltmek için başka hangi yöntemler uygulanabilir?
**Cevap:** Eğri uydurma

4. Aşağıda verilen tablodaki verileri sınıflandırmak için karar ağacı oluşturulacaktır.

| A | B | Sınıf |
|---|---|---|
| T | F | + |
| T | T | + |
| T | T | + |
| T | F | - |
| T | T | + |
| F | F | - |
| F | F | + |
| F | F | - |
| T | T | - |
| T | F | - |

(a) A ve B nitelikleri için bilgi kazancını hesaplayın. Karar ağacı algoritması hangi niteliği kullanarak bölmeleme yapar?
**Cevap:**
A ve B niteliklerine ait sınıf dağılımları

| | A=T | A=F |
|---|---|---|
| + | 4 | 0 |
| - | 3 | 3 |

| | B=T | B=F |
|---|---|---|
| + | 3 | 1 |
| - | 1 | 5 |

Bölünmeden önce entropi:

$$E_{orig} = -0.4 \log 0.4 - 0.6 \log 0.6 = 0.9710$$

A niteliği kullanılarak bölündüğü durumda bilgi kazancı:

$$E_{A=T} = -\frac{4}{7} \log \frac{4}{7} - \frac{3}{7} \log \frac{3}{7} = 0.9852$$

$$E_{A=F} = -\frac{3}{3} \log \frac{3}{3} - \frac{0}{3} \log \frac{0}{3} = 0$$

$$Gain(S, A) = E_{orig} - 7/10 E_{A=T} - 3/10 E_{A=F} = 0.2813$$

B niteliği kullanılarak bölündüğü durumda bilgi kazancı:

$$E_{B=T} = -\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} = 0.8113$$

$$E_{B=F} = -\frac{1}{6} \log \frac{1}{6} - \frac{5}{6} \log \frac{5}{6} = 0.6500$$

$$Gain(S, B) = E_{orig} - 4/10 E_{B=T} - 6/10 E_{B=F} = 0.2565$$

Veriyi bölmek için A niteliği seçilir.

(b) A ve B nitelikleri için Gini index değerini hesaplayınız. Karar ağacı algoritması hangi niteliği kullanarak bölmeleme yapar?

**Cevap:**

Bölünmeden önce Gini index

$$Gini_{orig} = 1 - 0.4^2 - 0.6^2 = 0.48$$

A niteliği kullanılarak bölündüğü gini göstergesi için kazanç:

$$G_{A=T} = 1 - (\frac{4}{7})^2 - (\frac{3}{7})^2 = 0.4898$$

$$E_{A=F} = 1 - (\frac{3}{3})^2 - (\frac{0}{3})^2 = 0$$

$$Gain(S, A) = G_{orig} - 7/10 Gini_{A=T} - 3/10 Gini_{A=F} = 0.1371$$

B niteliği kullanılarak bölündüğü gini göstergesi için kazanç:

$$G_{B=T} = 1 - (\frac{1}{4})^2 - (\frac{3}{4})^2 = 0.3750$$

$$E_{B=F} = 1 - (\frac{1}{6})^2 - (\frac{5}{6})^2 = 0.2778$$

$$Gain(S, B) = G_{orig} - 4/10 Gini_{B=T} - 6/10 Gini_{B=F} = 0.1633$$

Veriyi bölmek için B niteliği seçilir.

(c) Bilgi kazancı ve gini index değerlerinin bölmeleme için farklı nitelikleri seçmesi mümkün müdür? Açıklayınız.

**Cevap:**Gini göstergesi ve entropi ölçütleri benzer özellikler gösterir, ancak bu ölçütleri kullanarak hesaplanan kazanç aynı yukarıdaki örnekte olduğu gibi aynı özelliği göstermeyebilir.