Ligji Moore

Parashikon rritjen eksponenciale te fuqise kompjuterike duke zvogeluar madhesine e komponenteve kompjuterike dhe duke na lejuar te ndertojme procesore me komplekse dhe me te shpejte.

Perdoret per:

- Parashikimin e limitit te rritjes se fuqise kompjuterike
- Udheheqjen e zhvillimit kohor te industrise se gjysmepercuesve.
- Vendosjen e pritshmerive per avancim teknologjik

Tiparet kryesore:

- **Rritja eksponenciale**: Fuqia kompjuterike dyfishohet ne intervale te rregullta (zakonisht cdo 2 vite)
- **Miniatura**: Bazuar ne zvogelimin e madhesive se tranzistoreve, tranzistoret modere prodhohen ne 10nm, afersisht sa 50 atome te gjere.
- **Limiti fizik**: Komponentet arrije shkalle atomike – afrimi i barrierave themelore.
- **Realiteti ekonmik**: Procesoret e fundit kushtojne biliona dollare, me vetem 3 kompani ne mbare boten qe i afrohen shkalles.

Ligji Moore eshte drejt fundit per shkak te kufizimeve fizike, eshte shume e veshtire per te zvogeluar komponentet me tej, shume te shtrenjte per t'u prodhuar dhe me pak te besueshem pasi u afrohen dimensioneve atomike.

Proceset seriale dhe paralele

Proceset seriale: kryerja e detyrave me rradhe, njera pas tjetres. Detyrat nuk mund te ndahen apo paralelizohen. Paraqet bllokim ne shume llogaritje.

Proceset paralele: detyrat mund te ndahen dhe mund te kryhen njekohesisht nga procesore (core) te ndryshem. Mundeson speedup.

Ligji i Amdahl-it: parashikon speedup-in teorik kur shtohen me shume procesore ne kryerjen e detyres:

$$Speedup = \frac{1}{1 - P + \frac{P}{N}}$$

N – numri i procesoreve/cores

P – perqindja e e punes qe mund te paralelizohet

S = 1 – P : pjesa seriale e punes

Shtimi i vazhdueshem i procesoreve behet gradualisht me pak i perfitueshem pasi speedup maksimal varet nga shkalla e serializimit.

$$Speedup\ (\max) = \frac{1}{1 - P}$$

Ka faktore konkret qe zvogelojne speedup-in teorik si: koha e ndarjes se problemeve, komunikimi mes procesoreve, vonesa ne sinkronizim, bashkimi i rezultateve etj.

Shared Memory Parallelism (Memorja e perbashket)

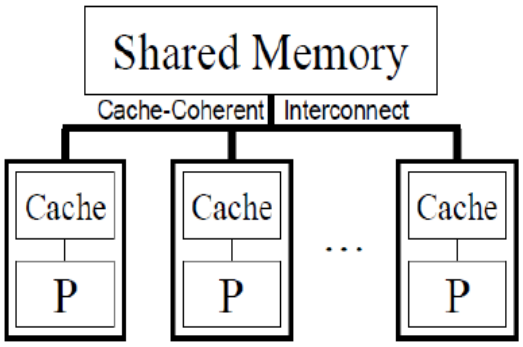
- Shume procesore (P) ndajne nje memorje te perbashket. Te gjithe procesoret mund te aksesojne te njejtat vendndodhje ne memorje. Ka Tight Coupling pasi procesoret jane shume te lidhur dhe komunikojne nepermjet memorjes se perbashket.
- Secili procesor ka cachen e tij per performancen
- Nderlidhja koherente me cache (cache-coherent interconnect) siguron qendrueshmeri te te dhenave ne te gjitha memorjet e perkohshme.

Shembuj: CPU Multi-core, Single Node Cluster, Symmetric Multi-Processing Systems

Metoda: OpenMP eshte vegla kryesore, threadet shperndajne variablat automatikisht, sinkronizimi ndodh nepermjet locks, barrierave dhe veprimeve atomike.

Avantazhet:

- Programohet lehte nepermjet variablave te shperndare (shared varaiables)
- Komunikim i shpejte (nepermjet aksesit ne memorje)
- Sherben per paralelizim ne detaje.



Disavantazhe:

- Shkallezueshmeri e ulet.
- Bottleneck ne memorje dhe overhead per koherences e cache.

Distributed Memory Parallelism (Memorja e shperndare)

- Shume procesore, secili me memorjen e tij personale.
- Procesoret lidhen nepermjet rrjetit (non-cache-coheret interconnect), perdoret loose coupling.
- Nuk ka hapesire te shperndare te adresave te memorjes (secili procesor mund te aksesoj vetem memorjen e tij).
- Informacioni duhet te dergohet ne menyre eksplicite ndermjet procesoreve.

Shembuj: Multiple node cluster, Supercomputers etj.

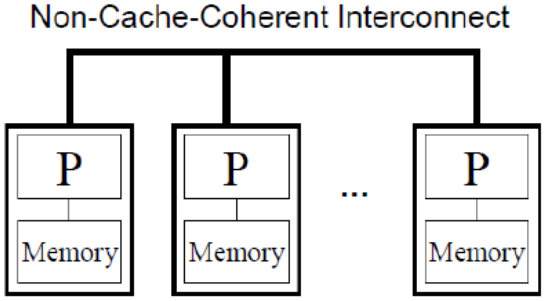
Metoda: MPI (message passing interface), dergim/marrje e veprimeve per shkembim informacioni, nuk ka shperndarje automatike te variablave

Avantazhet:

- Shume e shkallezueshme (mijera-miliona core) – mund te shtohen shume kompjutera
- Nuk ka bottleneck ne memorje – secili procesor ka memorjen e vet
- Me pak e kushtueshme duke perdorur hardware me pak te kushtueshem.

Disavantazhe:

- Programim me kompleks – duhet ti tregohet secilit procesor ekzaktesisht cfare duhet te bej
- Vonese me e larte ne komunikim
- Nevojitet menaxhim manual i te dhenave – informacioni duhet te levizet manualisht ndermjet kompjuterave, nuk ndodh automatikisht



Matja e kohes se ekzekutimit

#include <time.h> permban: *for clock_t, clock( ), CLOCKS_PER_SEC*

#include <unistd.h> permban: *for sleep( )*

- **Fillimi i matjes se kohes:** clock_t begin = clock();
- **Mbarimi i matjes se kohes:** clock_t end = clock();

Variablat e tipit clock_t e ruajne ne cikle clock kohen e matur. Per ta kthyer ne sec e pjestojme me variablin global te ndodhur ne librarine <time.h>:

double time = (double) (end-begin) / CLOCKS_PER_SEC;

OpenMP

Open Multi-Processing eshte nje API qe suporton programimin parallel ne memorje te perbashket. Suporton gjuhet si C, C++ dhe Fortran

Multithread: nje thread kryesor (master) krijon disa threade femije (slaves) dhe sistemi i ndan detyrat ndermjet tyre. Threadet ekzekutohen njekohesisht, me mjedisin runtime qe alokon threadet ne procesore (ose core) te ndryshem.

```
#include <stdio.h>
#include <omp.h>

int main() {
  int array[100], sum = 0, counter = 0;

  // 1. Basic parallel with private variables
  #pragma omp parallel private(id)
  {
    id = omp_get_thread_num();
    printf("Thread %d\n", id);
  }

  // 2. Parallel for loop
  #pragma omp parallel for
  for(int i = 0; i < 100; i++) {
    array[i] = i * 2;
  }

  // 3. Reduction (safe parallel sum)
  #pragma omp parallel for reduction(+:sum)
  for(int i = 0; i < 100; i++) {
    sum += array[i];
  }

  // 4. Critical section (safe counter)
  #pragma omp parallel
  {
    #pragma omp critical
    counter++;
  }

  // 5. Atomic operation
  #pragma omp parallel for
  for(int i = 0; i < 10; i++) {
    #pragma omp atomic
    counter++;
  }

  // 6. Different tasks for different threads
  #pragma omp parallel sections
  {
    #pragma omp section
    printf("Task A\n");

    #pragma omp section
    printf("Task B\n");
  }

  // 7. Only master thread executes
  #pragma omp parallel
  {
    #pragma omp master
    printf("Only master does this\n");
  }

}
```

Kompilimi: gcc -fopenmp program.c -o program

MPI

Message passing interface eshte nje API qe suporton programimin parallel ne memorje te shperndare. MPI realizon **"klonimin"** e një procesi në disa procese paralele për llogaritje paralele në CPU të shumëfishta.

Kompilimi: mpicc <program>

Ekzekutimi: mpiexec -np <procese> <program>

Libraria: #include <mpi.h>

- **Mpi_init(&argc, &argv)**

Starton mjedisin MPI, gjeja e pare qe duhet te behet kur punohet me MPI

- **Mpi_finalize**

Ndalon mjedisin MPI, gjeja e fundit qe duhet te behet para perfundimit te programit

- **MPI_Comm_rank(MPI_COMM_WORLD, &rank**)

Ruan vleren e process ID ne variablin rank, MPI_COMM_WORLD -> komunikuesi (grupi I te gjithe proceseve)

- **MPI_Comm_size(MPI_COMM_WORLD, &size)**

Kthen numrin total te proceseve dhe e ruan ne variablin size

- **MPI_Send(data, count, MPI_INT, dest, tag, MPI_COMM_WORLD)**

Dergon informacion ne nje proces tjeter specific. Parametrat (pointer ne informacionin qe dergohet, numri i elementeve qe do dergohen, lloji I te dhenave, ranku I procesit destinacion, label per mesazhin/ID, grupi I proceseve)

- **MPI_Recv(buffer, count, MPI_INT, source, tag, MPI_COMM_WORLD, &status)**

Merr informacionin nga nje proces tjeter. parametrat (pointer ku do te ruhet info i ri, numri i items qe pritet te merret, data type qe pritet te merret, burimi nga do merret MPI_ANY_SOURCE per cfaredo, mesazhi qe do pranohet, grupi I proceseve, statusi I meszhit te marre ose MPI_STATUS_IGNORE)

- **MPI_Bcast(data, count, MPI_INT, root, MPI_COMM_WORLD)**

Dergimi i informacionit Broadcast nga nje proces ne te gjithe proceset e tjera. Parametrat (data per te derguar ose buffer per te marre, numri i items, lloji I te dhenave, procesi qe po ben broadcast, grupi I proceseve)

- **MPI_Gather(sendbuf, sendcount, MPI_INT, recvbuf, recvcount, MPI_INT, root, MPI_COMM_WORLD)**

Mbledh te dhenat e te gjithe proceseve ne nje proces te vetem. Parametrat (data e derguar nga proceset, sa item dergon secili proces, lloji i te dhenave te derguara, vendi ku procesi prind ruan te dhenat, sa te dhena procesi prind pret nga cdo proces tjeter, lloji i te dhenave te marra, procesi qe mbledh gjithcka – prindi, grupi i proceseve)

- **MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD)**

Kombinon te gjitha te dhenat e proceseve duke perdorur veprime matematikore (data qe kontribuon cdo proces, aty ku prindi ruan rezultatin final, numri i elementeve qe do 'reduktohen', lloji I te dhenave, veprimi qe do kryhet SUM, MAX, MIN, PROD; procesi qe merr rezultatin final, grupi i proceseve)

- **MPI_Barrier(MPI_COMM_WORLD)**

Detyron te gjithe proceset te presin derisa secili prej tyre arrin te njejten pike ne kod.

Message Passing

Paradigma baze e komunikimit ne sisteme te shperndara ku proceset komunikojne duke derguar dhe marre mesazhe ne vend te ndarjes se memorjes (shared memory).

- Secili proces ka memorje te vecante.
- Nuk mund te aksesojne variablat e proceseve te tjera
- Komunikimi duhet te programohet ne menyre eksplicite
- Veprime Send/Receive

Mesazhet kalojne permjes rrjetit – mund te jene LAN (cluster) ose WAN (internet).

MPI eshte baza: pra realizohen proceset (broadcast, gather, reduce etj)

Mund te realizohet dhe me sockets ose message queues (RabbitMQ, Apache Kafka).

Mundëson shkallëzueshmëri dhe tolerance ndaj gabimeve, por kërkon programim më të kujdesshëm për të trajtuar kompleksitetet e rrjetit

Socket (Client-Server)

Mjet qe ben te mundur komunikimin mes nje klienti dhe serveri, nepermjet protokollit TCP/IP | <socket.h>

Per klientin TCP: socket() → connect() → send()/recv() → close()

- int sockfd; // Socket i klientit (vetëm 1)
- struct sockaddr_in server_addr; // Adresa IP dhe porti i serverit
- struct hostent *server; // IP e zgjidhur nga hostname
- char buffer[1024]; // Buffer për mesazhe

Per serverin TCP: socket() → bind() → listen() → accept() → send()/recv() → close()

- int sockfd; // Socket kryesor për dëgjim
- int newsockfd; // Socket i ri për komunikim me klient
- struct sockaddr_in server_addr; // Adresa IP dhe porti i serverit
- struct sockaddr_in client_addr; // Adresa e klientit që lidhet
- socklen_t addr_len; // Madhësia e strukturës
- char buffer[1024]; // Buffer për mesazhe**Libraria**: #include <socket.h>

Metodat Kryesore:

- **socket(int domain, int type, int protocol)** - Krijon socket të ri. Domain: AF_INET (IPv4), type: SOCK_STREAM (TCP), protocol: 0 (automatik). Kthimi: ID pozitive ose -1.
- **bind(int sockfd, struct sockaddr *addr, socklen_t addrlen)** - Lidh socket me IP lokale dhe port (vetëm server). Kthimi: 0 sukses, -1 gabim.
- **listen(int sockfd, int backlog)** - Bën socket pasiv për të pranuar lidhje. Backlog: numri maksimal i lidhjeve në radhë. Kthimi: 0 sukses, -1 gabim.
- **accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)** - Pranon lidhje nga klient (vetëm server). Kthimi: socket i ri për komunikim ose -1.
- **connect(int sockfd, struct sockaddr *addr, socklen_t addrlen)** - Lidhet me server (vetëm klient). Kthimi: 0 sukses, -1 gabim.
- **send(int sockfd, void *buf, size_t len, int flags)** - Dërgon të dhëna. Flags zakonisht 0. Kthimi: byte të dërguar ose -1.
- **recv(int sockfd, void *buf, size_t len, int flags)** - Merr të dhëna. Kthimi: byte të lexuar, 0 (lidhja mbyllur), -1 (gabim).
- **close(int sockfd)** - Mbyll socket dhe liron burime. Kthimi: 0 sukses, -1 gabim.
- **gethostbyname(const char *name)** - Konverton hostname në IP ("localhost" → 127.0.0.1). Kthimi: pointer strukturë ose NULL.

Threads

Threadet jane procese me peshe me te vogel qe ndajne memorjen e perbashket dhe burimet brenda nje programi. Ndryshe nga proceset qe krijohen me fork(), threadet jane:

- 10-100 here me te shpejte per t'u krijuar ne krahasim me threadet.
- Ndajne te njejten memorje (nuk ka nevoje per kopjim)
- Komunikojne lehte me ane te variablave shared

Metodat kryesore:

- **pthread_create(&thread_id, NULL, function_name, argument)** - Krijon nje thread te ri
- **pthread_join(thread_id, &return_value) –** Pret per perfundimin e threadit dhe merr vleren e kthimit
- **pthread_exit(return_value) –** Ndalon threadin qe po ekzekutohet dhe kthen vlere
- **pthread_t my_id = pthread_self() –** Merr ID e threadit aktual
- **pthread_detach(thread_id) –** I ben threadet e pavarura (pastrohen automatikisht kur perfundojne)

Sinkronizimi i threadeve:

Mutex (mutual exclusion): vetem nje thread mund te aksesoje resursin ne te njejten kohe.

- Lock() -> nese eshte i lire -> merr kontrollin; nese eshte i zene -> pret
- Unlock() -> liron mutexin, lejon threadin tjeter

Mekanzima te tjere lock: Read/Write Lock ose SpinLock

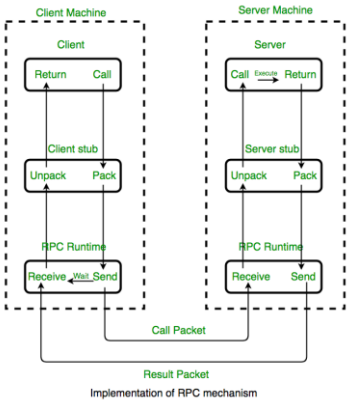
Problem: Deadlock (kur 2 thread presin njeri-tjetrin pergjithmone) -> zgjidhet duke marre lock ne rend te njejte

Remote Procedure Call (RPC)

RPC lejon nje program te therrase funksione ne makina remote sikur te ishin funksione lokale. E fsheh kompleksitetin e rrjetit te komunikimit.

Komponentet kryesore:

- **Client Stub:** Konverton thirrjet e funksioneve lokale ne mesazhe networku. Vendos parametrat ne paketa dhe i dergon drejt serverit.
- **Server Stub:** Merr mesazhet e rrjetit dhe i shnderron ato ne thirrje funksionesh lokale. I therret funksionet.
- **RPC Runtime:** Merret me vendosjen e komunikimit, trajtimin e gabimeve, sigurine etj.



Implementation of RPC mechanism

Hapat e funksionimit te RPC:

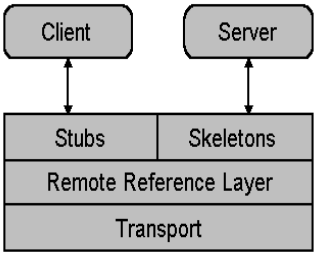
1. Klienti therret funksionin remote
2. Client stub vendos parametrat ne mesazhe
3. Mesazhi dergohet nepermjet rrjetit
4. Server stub merr dhe dekodon mesazhin
5. Serveri ekzekuton funksionin
6. Rezultati kodohet serish ne mesazh dhe dergohet
7. Klienti e merr rezultatin si te ishte thirrje lokale.

Remote Method Invocation (RMI)

RMI eshte version i RPC ne JAVA. Objektet JAVA therrasin metoda ne objektet qe po ekzekutohen ne makina remote sikur te ishin objekte lokale. Ndryshe nga RPC qe funksion me procedura dhe funksione, RMI funksionon me objekte dhe metoda.

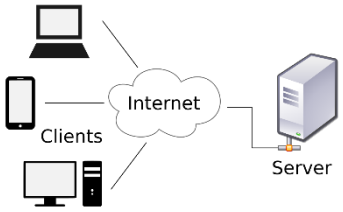
Komponentet kryesore:

- **Remote Interface:** percakton metodat qe mund te therriten remotely.
- **Remote Objects**: Implementon nderfaqe
- **Stub (Client):** Kodon parametrat dhe I degron ne server
- **Skeleton (Server):** Dekodon parametrat dhe therret objektin
- **RMI Registry**: U vendos emra referencave te objekteve remote.



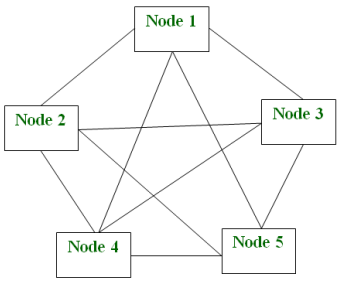
Arkitektura Client-Server

- **Centralizuar** - Të gjitha kërkesat kalojnë përmes serverit
- **Role të përcaktuara** - Klient dhe server kanë funksione të ndryshme
- **Asimetrike** - Klienti krijon kërkesa, serveri i përgjigjet
- **Server fiks** - Serverët kanë role dhe vendndodhje të paracaktuara
- **Varësi nga serveri** - Sistemi dështon kur serveri dështon
- **Lidhje të drejtpërdrejta** - Klienti komunikon drejtpërdrejt me serverin
- **Akses i kontrolluar** - Serveri kontrollon të gjitha burimet
- **Shkallëzueshmëria** - Varet nga kapaciteti i serverit
- **Single Point of Failure** - Pikë e vetme dështimi (single point of failure)



Arkitektura Peer to Peer

- **E pacentralizuar** - Nuk ka server qendror
- **Role simetrike** - Çdo nyje është njëkohësisht klient dhe server
- **Burime të shpërndara** - Burimet shpërndahen në çdo peer (nyje)
- **Vetë-organizuese** - Rrjeti mirëmban vetveten
- **Shkallëzueshmëria** - Përmirësohet me rritjen e nyjeve
- **Network overhead** - Nyjet duhet të zbulojnë njëra-tjetrën
- **Performance e paparashikueshme**
- **Overhead I larte:** per zbulimin e peer-eve



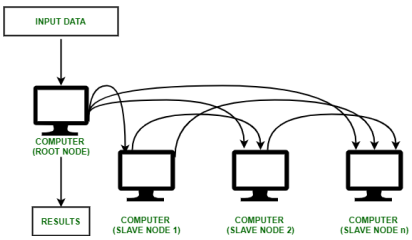
P2P Architecture

Cluster Computing

Nje grup kompjuterash te lidhur ngushte me njeri tjetrin qe veprojne si nje sistem i vetem.

- Homogjene: Hardware dhe Software te ngjashem
- Rrjet i shpejt i brendshem
- Menaxhim i centralizuar (nje pike kontrolli)
- Memorje te perbashket (Sistem filesh i perbashket)
- Single Point of Failure

Shembuj: Database clusters, Scientific computing etj.

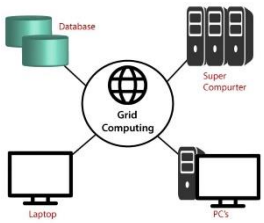


Grid Computing

Grupe heterogjene kompjuterash te shperndare gjeografikisht qe ndajne te njejtat burime.

- **Heterogjene:** Sisteme operative te ndryshme, hardware/aplikacione te ndryshme
- **Te shperndara gjeografikisht** (ne organizata apo shtete)
- **Ndajne burimet:** CPU, memorje, aplikacione
- **Organizim virtual:** bashkepunim dinamik

Shembuj: Research shkencor, modelim i motit etj.



Cloud Computing

Akses sipas kerkeses se burimeve nepermjet internetit.

Modele te ndryshme sherbimi si:

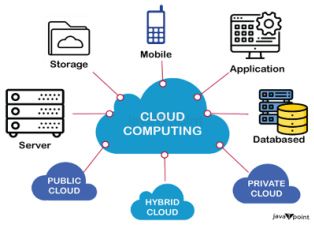
- Iaas: Infrastrukture (makina virtuale, memorje)
- PaaS: Platforme (mjedise zhvillimi)
- SaaS: Sofware (p.sh. GMAIL)

Modelet e zhvillimit:

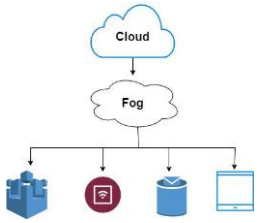
- Publik: i hapur per te gjithe (AWS, Google Cloud)
- Privat: nje organizate e vetme
- Hibrid: mix i publik dhe privat

Karakteristikat:

- Sipas kerkeses: Burimet jane te gatshme kur te nevojiten.
- Paguaj vetem ne rastet kur perdor burimet
- Te shkallezueshme automatikisht
- Ofruesi merret me mirembajtjen



Fog Computing



Eshte nje shtrese llogaritese ndermjet sherbimeve/pajisjeve cloud dhe edge.

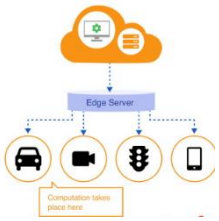
- E pacentralizuar: Informacioni procesohet afer burimit dhe jo ne cloud qendror
- Pergjigje e shpejte pasi informacioni nuk udheton gjate
- Ne cloud dergohet me pak informacion duke ruajtur bandwidth
- Heterogjene: punon me shume modele te ndryshme pajisjes

Shembull: smart city, self-driving cars, automation (IoT)

Edge Computing

Sherben per procesimin e informacionit fiks aty ku eshte krijuar (edge) pa patur nevoje te dergohet deri ne cloud.

Data Source -> Edge Device -> Fog Layer -> Cloud



- Informacioni procesohet menjehere (vonese shume e vogel)
- Vetem informacioni thelbesor dergohet ne cloud cka ruan burimet e rrjetit
- Mundeson vendimet e menjehershme per task-e qe duhet te procesohen shpejt.
- Mund te punoje dhe pa lidhje ne internet.

Aspekti	Cluster	Grid	Cloud	Fog	Edge
Location	Nje i vetem	Multi-site	Data centers	Regjionale	Lokale
Latency	Shume e ulet	Medium	Medium-High	E Ulet	Shume e ulet
Management	Centralizuar	Shperndare	Nga Provideri	Gjysme-shperndare	Lokale
Resources	Homogjene	Heterogjene	Standarde	Te Perziera	Limituara
Scalability	Limituar	E Larte	Shume e larte	Medium	E Ulet
Cost	E larte ne fillim	E shperndare	Pay-per-use	Medium	E Ulet

Protokolli HTTP

Protokolli baze per komunikimin ne web dhe sisteme te shperndara. Ne HTTP cdo kerkese eshte e pavarur – serveri nuk kujton kerekesat e meparshme. Klienti ben kerkesa dhe serveri pergjigjet.

HTTP realizon **dialogu request-response** midis komponentëve të shpërndarë:

Client/Service A ↔ Service B ↔ Database Service

HTTP HTTP TCP/SQL

Komunikimi realizohet nepermjet Request-Response Pattern: Client <-> HTTP Request <-> Server

KERKESA

<funksioni> <URI> <protokolli>
[<header>]
[…]
<CR><LF>
[<trupi kerkeses>]

Pergjigje

<protokolli> <status code>
[<header>]
[…]
<CR><LF>
[<trupi kerkeses>]

Cdo kerkese permban (fotoja perbri):

- URL e objektit (Unified Resource Locator)

  URL = <scheme>://<hostname>[:<port>]/<path>[?<query>]

  HTTP → komunikim me web service

  HTTPS → komunikim i enkriptuar

  FTP → transfer file-sh
- Metadata (header) per kontekstin
- Payload (data) nese nevojitet
- Funksioni mund te jete GET (te dhenat ne URI) , POST (te dhenat ne BODY), PUT, DELETE etj.

1. Perdoruesi i jep browserit URL e objektit te krijuar
2. Browser vecon emrin (ose IP) hostname (porta nese eshte specifikuar) dhe komunikon me shtresen TCP
3. TCP hap kanal komunikimi me aplikimin server
4. Browser i dergon serverit kerkesen, ku specifikohet adresa ne server e objketit te kerkuar.
5. Objektet ne server identifikohen me URI (Unified Resource Identifier)

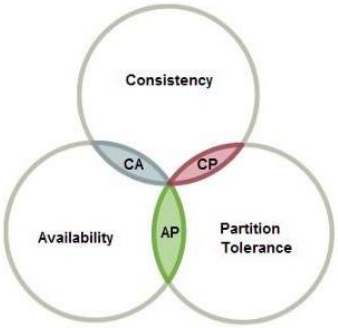
URI = <path>[?<query>] | **<path>** percakton skedarin objekt | **<query>** te dhena qe duhet ti kalohen objektit

Pergjigje: Ka format te ngjashem me te kerkeses. **Protokolli** (HTTP/1.0 ose 1.1), Status Code (1.1) mund te jene 200 (OK), 304 (not modified), 404 (not found), 412 (precondition failed), 501 (method not implemented)

Teorema CAP

Teoreme thelbesore per sistemet e shperndara qe percakton kufizimet e ketyre sistemeve.
Karakteristikat e CAP

- **C (Consistency): Qendrueshmeria**
  Te gjithe nyjet shohin te njejtat te dhena ne te njejten kohe.
  Cdo Read merr Write-n me te fundit
- **A (Availability): Disponueshmeria**
  Sistemi vazhdon te funksionoje dhe nese disa nyje deshtojne
  Cdo kerkese merr pergjigje
- **P (Partition Tolerance): Toleranca e ndarjes**
  Sistemi vazhdon te punon dhe kur ka nderprjerje rrjeti ndermjet nyjeve
  Rrjeti mund te ndahet ne pjese te vecanta.



Rregulli themelor: Ne nje sistem te shperndar, mund te garantosh vetem 2 nga 3 karakteristikat ne te njejten kohe. Kombinimet e mundshme

- **CP (mund te humbase availability):** kur ka ndarje, disa nyje ndalojne se funksionuari per te mbajtur konsistencen (p.sh MySQL, PostgreSQL ne distributed mode)
- **AP (mund te humbase consistency):** Sistemi vazhdon te funksionoje por te dhenat mund te jene te vjetra (p.sh DNS, WebCaching, NoSQL db etj).
- **CA (mund te humbase partition tolerance):** Funksionon vetem kur nuk ka nderprerje rrjeti (p.sh SingleNode databases, traditional RDBMS e tj).

Certifikatat Elektronike

Dokumente digjitale qe vertetojne identitetin e nje personi, organizate apo entiteti tjeter ne ambjentin elektronik. Sherbejne per te siguruar:

- **Autencitetin:** Verifikojne identitetin e pronarit duke vertetuar se informacioni po dergohet nga pala e duhur
- **Integritetin:** Ofrojne mbrojtje nga ndryshimet e paautorizuara te informacionit. Perdorimi i nenshkrimeve elektronike te bazuar ne certifikata siguron qe dokumentat te mos kene ndryshime te paautorizuara
- **Konfidencialiteti:** Certifikatat sigurojne kanale te sigurta te komunikimit, me perdorim protokollesh per enkriptimin e te dhenave

Ato perdorin teknologjine e celesave publike (Public Key Infrastrtucture – PKI)

Ato mbajne **private key (qe mbahet nga pronari)** dhe **public key (qe mund te shperndahet)**

Ato perfshijne te dhenat e pronarit dhe nje kopje te public key te tij