Distribuirani Sistemi

1. Uvod

- Distribuirani sistem je kolekcija nezavisnih računara koji se korisnicima prikazuju kao jedan povezan sistem. Imaju bolji odnos cena-performanse od mainframe računara.
- **Prednosti DS nad centralizovanim**: Može imati veću ukupnu moć obrade nego mainframe; Može se postupno povećavati dodavanjem novih računara (inkrementalni rast); Omogućeno je da više korisnika pristupa zajedničkim bazama podataka i periferijama; Omogućava da se opterećenje rasporedi na raspoložive računare na najefikasniji način.
- **Mane DS nad centralizovanim:** Mnogo veća kompleksnost od centralizovanog – teže je razviti softver za DS nego za centralizovane sisteme; Umrežavanje – mreža može otkazati ili postati zagušena; Bezbednosni problemi – Laka dostupnost resursa dovodi do problema bezbednosti.
- **Osnovne osobine DS:** Heterogenost, Transparentnost, Otvorenost, Skalabilnost.
- **Heterogenost** se ogleda u sledećem: Hardver računara – različit skup instrukcija, različita interna prezentacija podataka; Operativni sistemi – interfejs za razmenu poruka se razlikuje od OS do OS; Programski jezici – karakteri i strukture podataka se različito predstavljaju, što može biti problem ako aplikacije trebaju međusobno da komuniciraju.
- **Transparentnost (skrivanje nečega):** DS koji je u stanju da sakrije činjenice da su njegovi procesi i resursi fizički distribuirani je transparentan. Tipovi transparentnosti su: **Pristupna transparentnost** – podacima i resursima se pristupa na jedinstven načina, bez obzira da li se oni nalaze na udaljenom ili lokalnom računaru; **Lokacijska transparentnost** – korisnik ne zna gde je resurs fizički lociran u sistemu; **Migraciona transparentnost** – resurs može promeniti lokaciju a da klijent to ne primeti; **Transparentnost konkurencije** – omogućava da više procesa jednovremeno koristi isti resurs a da korisnici to ne primete; **Transparentnost replikacije** –omogućava postojanje više kopija istog resursa u DS u cilju povećanja performansi i pouzadnosti, a korisnici nisu svesni postojanja više kopija; **Transparentnost za otkaze** – korisnik ne zna da je resurs u kvaru.
- **Otvorenost:** Otvoreni DS dozvoljavaju dodavanje novih servisa i omogućavaju dostupnost servisima od strane različitih klijenata. Nude usluge shodno standardnim pravilima koje definišu sintaksu i semantiku usluga. Kod DS usluge su definisane interfejsima koji se opisuju pomoću posebnog jezika (IDL – Interface Definition Language).
- **Skalabilnost** se može posmatrati kroz tri dimenzije: skalabilnost u odnosu na broj korisnika i resursa, skalabilnost u odnosu na geografsku udaljenost korisnika i resurs, i administrativna skalabilnost (sistemom se može lako upravljati i ako se prostire kroz više administrativnih domena). Postoje tri tehnike skaliranja: **Skrivanje komunikacionog kašnjenja –** koriste se asinhrone komunikacije umesto sinhronih (kada se pošalje zahtev, klijent se ne blokira već radi neki drugi posao), download-uje se deo koda na klijentsku stranu da bi se ubrzala obrada**; Distribucija –** podrazumeva da se komponente dele na manje delove a zatim se ti delovi distribuiraju na više mašina u sistemu**; Replikacija –** pomaže da se poveća dostupnost i balansira opterećenje u sistemu da bi se postigle bolje performanse (u sistemima koji su geografski distribuirani poželjno je da postoji kopija resursa blizu mesta korišćenja kako bi se smanjilo komunikaciono kašnjenje).

- **Middleware** su protokoli opšte namene koji su od koristi za mnoge aplikacije, a ne mogu se kvalifikovati kao transportni protokoli. Usluge koje ovi protokoli pruzaju su autentifikacija, autorizacija i sinhronizacija (vremenska sinhronizacija ili sinhronizacija kod pristupa deljivom resursu). Osnovni cilj middleware je da sakrije heterogenost platforme na kojoj je sistem izgrađen od aplikacije.
- **RPC model** zasniva se na pozivu udaljenih procedura u kome se resursi modeliraju kao funkcije. Procesima je omogućeno da pozivaju procedure čija je implementacija locirana na drugoj mašini. Kada se poziva takva procedura, parametri se transparentno prenose do udaljene mašine na kojoj se zatim procedura izvršava, a rezultat šalje pozivaocu. Zbog transparentnosti, pozivajući proces ima utisak da se procedura izvršila lokalno.
- **Objektno orijentisani pristup:** resursi se modeluju kao objekti koji sadrže skup podataka i funkcija nad podacima. Udaljenom resursu se pristupa kao objektu. Ovaj prilaz doveo je do pojave različitih middleware sistema koji podržavaju koncept distribuiranih objekata.
- **Tipovi DS:** Na arhitekturnom nivou: Klijent Server, Peer-to-peer; U odnosu na oblasti primene: Distribuirani računarski sistemi, Distribuirani informacioni sistemi, Sveprisutni DS.
- **Distribuirani računarski sistemi** se koriste za izvršenje visokoperformansnih zadataka (izračunavanja). Mogu biti **Klasteri** – računari se sastoje od grupe sličnih PC koji se nalaze na malom rastojanju, povezani preko brze lokalne mreže, pri tome svaki čvor izvršava isti OS; i **Grid** – sastoji se od grupe računarskih sistema koji mogu biti u različitim administrativnim domenima i mogu se razlikovati u pogledu hw, sw i mrežne tehnologije.
- **Sistemi za obradu transakcija**: transakcija predstavlja skup operacija koje se obavljaju kao jedna nedeljiva (atomična) operacija. Sistem za obradu transakcija obezbeđuje da sve ili ni jedna operacija u transakciji budu izvršene bez greške. Nakon obavljanja transakcije sistem mora da ostane u konzistentnom stanju. Transakcije moraju da zadovolje **ACID test** pre nego što se dozvoli modifikacija sadržaja (Atomicity – transakcija se obavlja kompletno ili se uopšte ne obavi; Consistency – transakcija ne ugrožava skup konstanti sistema; Isolation – transakcije su međusobno izolovane; Durability – kada se transakcija obavi ne može se poništiti).
- **Sveprisutni DS:** uređaji u ovim DS su uglavnom mali, napajaju se pomoću baterija i mobilni su, imaju samo bežične veze. Važna osobina ovakvih DS je odsustvo administrativnog upravljanja.

2. Komunikacija

- **RPC:** Osnovna ideja RPC je da poziv udaljene procedure izgleda što sličniji pozivu lokalne procedure. Obezbeđuje infrastrukturu neophodnu za transformisanje poziva procedure u poziv udaljene procedure na uniforman i transparentan način. Transparentnost se postiže na sledeći način: Ako je read udaljena procedura, drugačija verzija read koja se zove klijent stub se poziva koja na osnovu primljenih parametara gradi poruku i poziva lokalni OS da prenese poruku do udaljenog servera. Stub funkcija izgleda kao funkcija koju korisnik želi da pozove, ali ona stvarno sadrži kod za slanje i prijem poruka kroz mrežu.
- **Koraci kod poziva RPC:** Klijent poziva lokalnu proceduru (klijent stub); Klijent stub pakuje argumente za udaljenu proceduru, gradi mrežne poruke i poziva lokalni OS; OS šalje poruku udaljenom OS-u; Nakon prijema udaljeni OS prosleđuje poruku serverskom stub-u. Serverski stub prima poruku, raspakuje je i izvlači argumente za poziv lokalne procedure; Stub poziva željenu serversku proceduru i predaje joj primljene parametre; Server izvršava pozvanu proceduru i vraća rezultate u stub;

Serverski stub pakuje rezultat u poruku i poziva OS; Serverski OS šalje poruku klijentskom OS-u; Lokalni OS prosleđuje poruku klijent stub-u; Klijent stub izvlači rezultat iz poruke i vraća klijentskom procesu. Svi detalji prosleđivanja poruka su skriveni u klijentskim i serverskim stub funkcijama.

- **Prenos parametra kod RPC:** Prenos po vrednosti je jednostavan, vrednosti se kopiraju u mrežne poruke. Prenos po referenci je teško implementirati zato što nema nikakvog smisla proslediti adresu udaljenoj mašini, tako da je zamenjen tehnikom copy/restore (kod poziva procedure parametri se kopiraju u stek kao kod poziva po vrednosti, ali nakon okončanja poziva vrednosti se upisuju preko originalnih parametara kao kod poziva po referenci).
- **Lociranje udaljenog servera i procedure:** može se rešiti na dva načina: **Statičko povezivanje** – klijent zna koji host treba da kontaktira (adresa hosta se nalazi u klijent stub-u), a kada klijent pozove odgovarajuću proceduru, klijent stub prosledi poziv serveru (Jednostavno i efikasno, ali klijent i server postaju čvrsto povezani); **Dinamičko povezivanje** – postoji centralizovana baza podataka smeštena u name i directory serverima koja može locirati host koji obezbeđuje željeni servis (Serveri vraćaju adresu servera na osnovu potpisa procedure koja se pozove. Ako server promeni adresu, dovoljno je promeniti ulaz u name serveru).
- **Semantika poziva udaljenih procedura:** udaljena procedura može se izvršiti nula puta (ako server otkaže pre izvršenja serverskog koda), jednom (ako je sve ok), jednom ili više puta (ako je komunikaciono kašnjenje veliko ili se izgubi odgovor od strane servera pa klijent vrši retransmisiju). Većina RPC sistema obezbedjuje **bar jednom** semantiku (ako su u pitanju funkcije koje se mogu izvršavati bez posledica više puta – idenpotentne) ili **samo jednom** semantiku (funkcija nije idenpotentna).
- **Sun RPC** obezbeđuje jezik za definisanje interfejsa (XDR) i interfejs kompajlera (rpcgen) koji se koristi za generisanje klijent i server stub funkcija. Rpcgen kompajler generiše tri fajla: **header fajl** – sadrži identifikator interfejsa, definiciju tipova, konstanti i prototipova funkcija, mora biti uključen u klijent i server kod sa #include; **klijent stub** (primer_clnt.c) – sadrži procedure koje će klijent program pozivati; **server stub** (primer_svc.c) – sadrži procedure koje se pozivaju kada stigne poruka do servera. Klijent mora znati ime udaljenog servera. Server registruje svoje usluge preko portmapper deamon procesa na serverskoj mašini na portu 111.
- **Sun XDR definicija interfejsa:** sve udaljene procedure se deklarišu kao deo udaljenog programa. Procedure mogu imati samo jedan argument, tako da je potrebno napraviti novu strukturu. **Primer** - servis KALK sa dve procedure: SABERI(x, y), KVADRIRAJ (x): struct operandi { int x; int y;}; program KALK { version KALK_VERSION { int SABERI(operandi) = 1; int KVADRIRAJ(int) = 2; } = 1 }
- **Distribuirano računarsko okruženje (DCE)** je middleware baziran na RPC projektovan kao nivo apstrakcije između mrežnog OS i distribuirane aplikacije baziran na modelu klijent-server. Sve komunikacija između klijenta i servera se odvija pomoću RPC. DCE servisi su RPC, Distribuirani fajl servis, Direktorijumski servis (servis koji na osnovu imena objekta vraća informaciju o imenovanom), Bezbednosni servis (omogućava da pristup resursima bude dozvoljen samo autorizovanim korisnicima), Servis distribuiranog vremena. Podržava obe semantičke opcije kod poziva udaljene procedure.
- **IDL:** u klijent-server sistemu baziranom na RPC lepak koji sve drži na okupu je definicija interfejsa specificirana u IDLu. Omogućava deklaraciju procedura slično deklaraciji prototipa funkcija u C-u. Ključni element u svakom IDL fajlu je globalno jedinstveni identifikator interfejsa. Kada je IDL fajl potpun poziva se IDL kompajler koji generiše isti izlaz kao rpcgen.

- **Pozivezivanje klijenta i servera:** da bi klijent mogao da poziva server, neophodno je da server bude registrovan i spreman da primi poziv. DCE klijent pronalazi server u dva koraka: lociranje serverske mašine i lociranje servera na serverskoj mašini. Da bi klijent mogao da komunicira sa serverom, mora da zna broj porta na serverskoj mašini na koji može poslati poruke.
- **RMI** (poziv udaljenih metoda) je RPC na OO način. Skriva svoju unutrašnjost uz pomoć dobro definisanog interfejsa. Objekat može imati skup metoda koje se mogu pozivati sa udaljenih računara koji su definisani u remote interface fajlu. Ne postoji ni jedan drugi način da se pristupi podacima unutar objekta osim preko metoda koji su dostupni preko interfejsa. Kada klijent pozove metod udaljenog objekta, poziv se prosledjuje serverskom procesu u kome se nalazi udaljeni objekat. **Binder** sadrži tabelu sa preslikavanjem tekstualnog imena u referencu udaljenog objekta. Server registruje udaljene objekte u binder-u, a klijent koristi binder da pronadje referencu udaljenog objekta.
- **Tipovi komunikacije** su perzistentne (istrajne), tranzijentne, sinhrone i asinhrone.
- **Perzistentne komunikacije:** poruke koje se prenose se pamte u komunikacionim serverima dok se ne proslede sledećem komunikacionom serveru. Izvor ne mora biti aktivan nakon što dostavi poruku komunikacionom serveru, kao ni prijemnik kada izvor pošalje poruku.
- **Tranzientne komunikacije:** kod tranzientnih komunikacije poruke se skladište samo dok se izvorna i odredišna aplikacija izvršavaju, dok ako se ne izvršavaju, poruke se odbacuju. Komunikacioni server u ovom slučaju je store-and-forward ruter.
- Kod **asinhronih komunikacija** pošiljalac nastavlja sa radom odmah nakon što prosledi poruku za slanje. Kod **sinhronih komunikacija** pošiljalac se blokira dok poruka ne stigne do odredišta.
- Postoji nekoliko mogućih kombinacija ovih tipova komunikacija: **Perzistentne asinhrone** – poruka je zapamćena ili u baferu lokalnog hosta ili u prvom komunikacionom serveru sve dok se ne isporuči, pošiljalac nije blokiran dok čeka isporuku; **Perzistentne sinhrone –** poruka mora biti zapamćena u odredišnom hostu da bi se pošiljalac deblokirao; **Tranzijentne asinhrone** – poruka se privremeno pamti u lokalnom baferu izvornog hosta nakon čega aplikacija nastavlja sa izvršenjem (ako prijemnik nije aktivan, poruka se gubi); **Tranzientne sinhrone** – zahteva blokiranje izvora dok se poruka ne zapamti u baferu odredišnog hosta.
- **MPI** (Message Passing Interface) je middleware sistem koji se bazira na prenosu poruka. Namenjen je za podršku komunikacijama u paralelnim sistemima. Podržava komunikaciju između konkurentnih procesa (point-to-point i grupna komunikacija). Podržava skoro sve oblike tranzientnih komunikacija. U slučaju komunikacije u okviru poznate grupe procesa svakoj grupi se dodeljuje identifikator i svakom procesu unutar grupe lokalni identifikator koji zajedno identifikuju izvor ili odredište poruke.
- **Sistemi sa redovima poruka:** važna klasa middleware sistema zasnovana na slanju poruka. Namenjeni su aplikacijama kod kojih je dozvoljeno da prenos poruka traje i nekoliko minuta. Ne zahtevaju ni od izvora ni od odredišta da buda aktivni za vreme prenosa. Aplikacije komuniciraju smeštanjem poruka u posebne redove čekanja.

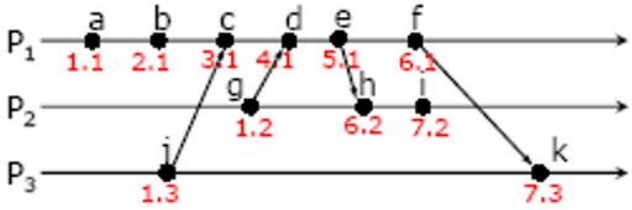
3. Sinhronizacija

- **Fizički časovnik:** svi računari imaju kolo koje beleži vreme (tajmer - kvarcni kristal koji kada je pod naponom osciluje na poznatoj frekvenciji). Svakom kristalu su pridružena dva registra: Brojač koji se dekrementira svakom oscilacijom i resetuje se na početnu vrednost iz Holding registra nakon 0. Kada

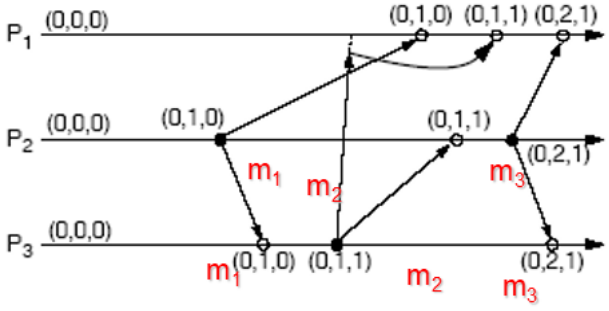
postoji samo jedan procesor i jedan časovnik, svi procesi na mašini koriste isti časovnik tako da će biti interno konzistentni.

- **Kristijanov algoritam** (eksterna sinhronizacija): u sistemu postoji jedna mašina sa WWV prijemnikom (server vremena – time server), a cilj je da sve ostale mašine budu sinhronizovane sa tom mašinom. Svaka mašina periodično šalje poruku serveru vremena tražeći informaciju o vremenu. Vreme se računa sa $T_{novo} = C_{UTC} + \frac{T_1 - T_2}{2}$ gde je C primljeno vreme a T_1-T_2 propagaciono kašnjenje.
- **Berkeley algoritam** (interna sinhronizacija): koristi suprotan prilaz od Kristijanovog algoritma, server obavlja prozivku svake mašine periodično da dozna koliko je vreme na datoj mašini, izračunava srednje vreme na osnovu odgovora i saopštava ostalim mašinama kako da podese svoje časovnike.
- **NTP protokol** je internetov klijent-server protokol za sinhronizaciju časovnika u klijentima sa UTC vremenom. Serveri su izvori informacija o tačnom vremenu i hijerarhijski su organizovani u slojeve (stratum-e). Postoje tri režima za sinhronizaciju časovnika u hostovima: **Simetrični režim** – obezbeđuje najprecizniju sinhronizaciju i koristi se za sinhronizaciju master servera; **RPC režim** – klijent kontaktira NTP server da bi sinhronizovao svoj časovnik tako što u poruci navodi lokalno vreme. Server odgovara porukom u kojoj se nalazi vreme pristizanja poruke, vreme na serveru i vreme slanja poruke. Klijent beleži vreme pristizanja odgovora i određuje tačno vreme; **Multicast režim** – NTP server periodično broadcastuje vreme koje ostali hostovi koriste.
- **Logički časovnici:** prioritet je redosled događaja, ne fizičko vreme. Dovoljno je da se mašine koje ostvaruju interakciju usaglase oko relativnog vremena, i to vreme ne mora da se slaže sa fizičkim vremenom.
- **Lamportov algoritam sinhronizacije:** definiše relaciju A⟶B, odnosno A se desilo pre B. Ova relacija je dozvoljena u dva slučaja: Ako su A i B događaji u istom procesi u ako A nastupa pre B, tada je relacija A⟶B tačna; Ako je A događaj koji predstavlja slanje poruke iz jednog procesa, a B događaj prijema poruke u drugom procesu, tada je A⟶B takođe tačna. Ova operacija je tranzitivna tako da ako je A⟶B i B⟶C, tada je i A⟶C. Ako su se događaji desili u dva procesa koji ne razmenjuju poruke, kaže se da su događaji konkurentni. Ako je relacija A⟶B tačna, potrebno je ostvariti sinhronizaciju tako da važi T(A)<T(B). Svakom događaju dodeljuje se timestamp. Sinhronizacija se ostvaruje korekcijom logičkih časovnika tako što se časovniku dodaje neka vrednost.
- **Pravila Lamportovog algoritma:** Logički časovnik Li se inkrementira pre izvršenja bilo koje aktivnosti u procesu Pi, Li=Li+1; Kada proces Pi šalje poruku m on u nju ubacuje vrednost t=Li; Kada proces Pj primi poruku (m, t) on podešava svoj lokalni logički časovnik tako što određuje Lj=max(Lj, t) i primenjuje pravilo 1, a zatim prosleđuje poruku aplikaciji.

- **Problemi sa Lampartovim algoritmom:** moguće je da više događaja koji nisu međusobno uslovljeni imaju identične vremenske markice. Ovo može dovesti do konfuzije ako više procesa treba da donese odluku na osnovu vremenskih markica dva događaja. Mora se prisiliti da svaka markica bude jedinstvena definisanjem globalne vremenske markice (Ti, *i*) gde je Ti lokalna Lamportova markica, a *i* broj procesa.



- **Vektorski časovnici** rešavaju problem što na osnovu Lampartovih markica nije moguće odrediti koji su događaji međusobno uslovljeni, a koji su konkurentni. Vektorski časovnik u sistemu N procesa je vektor od N celobrojnih elemenata. Svaki proces ima sopstveni vektorski časovnik kojim beleži lokalne događaje i šalje se sa svakom porukom. **Pravila:** Vektor se inicijalizuje na 0 u svim procesima; Proces Pi inkrementira i-ti element vektora u lokalnom vektoru pre nego što obeleži događaj; Kada Pj primi poruku, poredi lokalni vektor sa primljenim element po element i postavlja element u lokalnom vektoru na veću od vrednost. Za događaje A i B važi: ako je A→B, tada V(A)<V(B); Ako je V(A)<V(B), tada je A→B. Ako se ne može uspostaviti relacija V(A)≤V(B) ili V(A)≥V(B) tada su događaji konkurentni (na primer A(2,2,0) i B(0,0,1)). Za proces Pi, Vi[i] predstavlja broj događaja koji su se desili u procesu Pi. Ako je Vi[j]=k, tada Pi zna da se u Pj desilo k događaja.
- **Pravila vektorskih časovnika:** Pre emisije poruke m, proces Pi inkrementira svoj vektorski časovnik - $V_i[i]=V_i[i]+1$. Markica tm= V_i se šalje sa porukom m. Na prijemnoj strani poruka m se ne prosleđuje procesu Pj dok se ne zadovolje sledeća dva uslova: **$tm[i]==V_j[i]+1$** – ovaj uslov obezbeđuje da proces Pj primi sve poruke koje je poslao Pi pre slanja poruke m. Ako je tm[i]=8, a stigao je Vj[i]=5, znači da poruke 6 i 7 još nisu stigle; **$tm[k]\leq V_j[k]$, za svako $k\neq i$ –** obezbeđuje da proces Pj primi sve poruke koje je primio Pi pre slanja poruke m. Kada se poruka prosledi procesu, njegov lokalni časovnik se ažurira.



- **Uzajamno isključivanje:** u sistemima sa zajedničkom memorijom može se postići hardverskim mehanizmima ili softverski pomoću semafora ili monitora, što nije moguće u DS gde je jedini način razmenom poruka. Algoritmi uzajamnog isključivanja u DS mogu biti centralizovani, distribuirani i bazirani na žetonima (token).
- **Centralizovani algoritam** oponaša jednoprocesorski sistem. Jedan proces u DS je odabran za koordinatora tako da kada proces želi da pristupi deljivom resursu šalje zahtev koordinatoru. Ako ni jedan drugi proces nije u datoj kritičnoj sekciji, koordinator šalje dozvolu I označava da proces koristi datu KS. Ako je neki drugi proces u KS koordinator ne šalje odgovor, a proces koji je uputio zahtev se blokira. Kada proces okonča pristup KS šalje poruku oslobađanja koordinatoru, a koordinator može zatim poslati dozvolu pristupa procesu koji je blokiran.
- **Distribuirani algoritam Ricart & Agrawala:** koristi multicast I logičke časovnike. Proces koji želi da uđe u KS kreira poruku koja sadrži identifikator, ime resursa i logički časovnik, šalje zahtev procesima u grupi i čeka na dozvolu od svih procesa. Kada proces primi zahtev ako nije zainteresovan za korišćenje resurs šalje potvrdu, ako je u KS ne odgovara i smešta zahtev u red čekanja, a ako je prijemnik poslao zahtev za pristup KS poredi vremenske markice poslatog i primljenog zahteva i šalje potvrdu procesu sa manjom markicom. Kada okonča pristup KS šalje potvrdu svima koji su bili u redu čekanja.
- **Algoritam sa prstenom i žetonima** (token ring) – konstruiše logički prsten koji nema veze sa fizičkim vezama između računara dodelom rednih brojeva procesima. Inicijalno proces 0 dobija žeton za resurs

R. Žeton se prenosi u pravcu kazaljke na satu od procesa do procesa dok se ne dođe do procesa koji zahteva resurs R. Proces koji zahteva resurs R čeka dok ne dobije žeton.

- **Algoritmi izbora** su algoritmi za selekciju koordinatora ili servera. Cilj algoritama je da pronađu aktivni proces sa najvećim identifikatorom i proglase ga za koordinatora. Proces izbora se obično odvija u dve faze: Selekcija lidera sa najvećim brojem, Obaveštavanje svih procesa o pobedniku.
- **Bully algoritam:** proces Pi koji detektuje da koordinator ne odgovara startuje izbor. Izborna poruka se šalje svim procesima sa većim identifikatorom i čeka se odgovor. Ako u određenom vremenu ne stigne odgovor ni od jednog procesa to znači da ni jedan proces sa većim identifikatorom nije aktivan i pobednik je proces koji je inicirao izbore. Ako pristigne odgovor, proces Pi se povlači i čeka da dobije poruku od novog koordinatora.
- **Ring algoritam izbora:** procesi formiraju logički prsten. Ako neki proces detektuje da koordinator ne funkcioniše startuje algoritam izbora novog koordinatora tako što šalje poruku sa svojim ID svom susedu u prstenu. Ako sused nije aktivan, poruka se prosleđuje sledećem procesu u prstenu. Po prijemu poruke, svaki proces joj dodaje svoj ID i prosleđuje je sledećem procesu. Kada poruka stigne do procesa koji je inicirao izbore, on detektuje u poruci svoj ID i na osnovu poruke zaključuje ko su članovi prstena. Šalje novu poruku "KOORDINATOR" sa brojem procesa sa najvećim identifikatorom da obavesti ostalne procese o novom koordinatoru.

4. Konzistentnost i replikacija

- **Modeli konzistencije** definiše pravila po kojima se replicirani podaci moraju ažurirati. Ako se procesi slože da poštuju određena pravila, skladište podataka garantuje da će funkcionisati korektno. Konzistencija je definisana u kontekstu read i write operacija nad deljivim podacima. Modeli mogu biti: Striktna, Sekvencijalna, Kauzalna i FIFO.
- **Striktna konzistencija** je najjači model konzistencije. Poštuje pravilo da bilo koja operacija nad podatkom X vraća rezultat poslednje write operacije nad X. Jednoprocesorski sistemi podržavaju striktnu konzistenciju, ali u DS striktnu konzistenciju je nemoguće postići zato što se zasniva na apsolutnom globalnom vremenu. Ako je skladište striktno konzistentno, svi upisi su trenutno vidljivi svim procesima što je u DS nemoguće, tako da se uvode slabiji modeli koji nisu bazirani na apsolutnom vremenu.
- **Sekvencijalna konzistencija** je slabiji model od striktne. Kada se procesi izvršavaju konkurentno na različitim mašinama, bilo koje validno preplitanje read i write operacija je prihvatljivo, ali svi procesi vide isto preplitanje operacija.
- **Primer sekvencijalne konzistencije:** neka su data tri procesa koja se konkurentno izvršavaju:

Process P1	Process P2	Process P3	
x = 1;	y = 1;	z = 1;	write
print (y, z);	print (x, z);	print (x, y);	read

X, Y i Z su celobrojne promenljive inicijalizovane na 0 i zapamćene u deljivom sekvencijalno konzistentnom skladištu. Dodeljivanje odgovara write naredbi, print odgovara read operaciji dva argumenta. Moguća se 6! (720) preplitanja, mada neka narušavaju programski redosled. Razmotrimo

5! (120) sekvenci koje počinju sa x=1. Pola njih ima print(x, z) pre y=1 i tako narušava sekvencijalno uređenje programa, a pola od preostalih će imati print(x, y) pre z=1 što takođe narušava redosled, tako da je validno samo 30. Za y=1 na početku i z=1 na početku validno je još po 30, tako da postoji 90 validnih sekvenci koje generišu <64 različitih rezultata koji su dozvoljeni pod pretpostavkom sekvencijalne konzistencije.

- **Kauzalna konsistencija:** upisi koji su potencijalno uslovljeni moraju se videti u svim procesima u istom redosledu. Konkurentni upisi se mogu videti u različitom redosledu u različitim procesima, na primer W(x)a→R(x)a, R(x)a →W(x)b, ⇒ W(x)a →W(x)b, ova dva upisa su potencijalno uslovljena. Ovakvo skladište podataka je uslovno konzistentno, ali ne i sekvencijalno. Da bi se implementirala uslovna konzistencija mogu se koristiti vektorski časovnici za uređenje međusobno zavisnih događaja.

P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

- **FIFO konzistencija:** upisi koje obavi jedan proces se vide od strane drugih procesa po redosledu u kome su izdati. Ne postoje nikakve garancije o tome kako različiti procesi vide upise drugih procesa, osim što dva ili više upisa iz istog izvora moraju stići po redosledu.
- **Repliciranje sadržaja:** ključno pitanje u DS koji podržava replikaciju je gde, kada i ko može obavljati repliciranje i koji se mehanizmi koriste za održavanje konzistencije. Postoji tri tipa replika: Permanentne replike, Replike inicirane od strane servera, Replike inicirane od strane klijenta.
- **Permanentne replike** mogu se posmatrati kao početni skup replika koje obrazuju distribuirano skladište podataka. Broj permanentnih replika je mali.
- **Replike inicirane od strane servera:** Formiraju se na inicijativu vlasnika skladišta podataka da bi se poboljšale performanse ili rasteretio server ako u nekom trenutku dolazi veliki broj zahteva sa neke udaljene lokacije. To su privremene replike postavljene u regione iz kojih dolazi veliki broj zahteva. Algoritam dinamičke replikacije uzima u obzir broj obraćanja fajlu i odakle dolaze zahtevi. Ako dva klijenta C1 i C2 imaju isti najbliži server P, svi zahtevi za pristup fajlu F u serveru Q od strane C1 i C2 se jedinstveno registruju u serveru Q kao da dolaze od servera P. Ako je broj obraćanja fajlu F na serveru Q viši od praga replikacije rep(Q, F), donosi se odluka da se izvrši replikacija fajla F na server P. Kada broj obraćanja fajlu F na serveru P padne ispod praga del(P, F), fajl može biti uklonjen sa servera P pod uslovom da to nije jedina kopija tog fajla. Ako je broj zahteva za pristup između rep(Q, F) i del(P, F), fajl može samo migrirati sa jednog servera na drugi.
- **Replike inicirane od strane klijenta** su poznate pod nazivom klijent keš. Keš je lokalna memorija koja se koristi od strane klijenta da privremeno zapamti kopiju podataka koje je upravo zahtevao. Upravljanje kešom je u potpunosti prepušteno klijentu i skladište podataka nema nikakvu obavezu u održavanju konzistencije keša. Koristi se za smanjenje vremena pristupa.
- **Distribucija ažuriranja:** operacije ažuriranja mogu biti inicirane od strane servera (push prilaz) u kome se ažuriranje prenosi replikama na inicijativu servera, mada replike to nisu zahtevale, i na zahtev klijenta (pull prilaz) kada server ili klijent zahtevaju od drugog servera da mu prosledi ažuriranje ako je postojalo.

- **Protokoli konzistencije** opisuju implementaciju određenog modela konzistencije. Postoje tri vrste: Protokoli zasnovani na postojanju primarne kopije, Protokoli sa više ravnopravnih replika, Keš koherentni protokoli. Ako se zahteva postizanje sekvencijalne konzistencije najčešće se koriste protokoli zasnovani na primarnoj. Kod ovih protokola svaki podatak X u skladištu podataka ima pridružen primar koji je zadužen za koordinaciju upisa u X. Može se napraviti razlika u odnosu na to da li je primar fiksiran na udaljenom serveru, **Remote-write protokoli**, ili se write operacija može obaviti lokalno nakon što se primar pomeri na proces koji je inicirao write, **Local-write protokoli**.
- **Remote-write protokoli**: sve read i write operacije se obavljaju na udaljenom serveru. Podaci nisu replicirani, već se nalaze na jednom serveru odakle se ne mogu pomerati. Svi zahtevi se upućuju primarnoj kopiji, a ostale kopije služe kao backup. Postoji i modifikacija, **Primar-backup protokoli**, koji dozvoljavaju procesima da obave read na lokalno raspoloživoj kopiji, a operacije write na fiksnoj, primarnoj kopiji.
- **Local-write protokoli:** primarna kopija migrira između procesa koji žele da obave write operaciju. Kada proces želi da ažurira podatak X, on prvo locira primarnu kopiju X a zatim je privlači u sopstvenu lokaciju. Ovaj prilaz se koristi u distribuiranim fajl sistemima.
- **Replicirani write protokoli:** kod ovih protokola write operacija se može obaviti na više replika, umesto na jednoj kao kod protokola zasnovanim na primarnim kopijama. Sve replike su ravnopravne i svaka replika ima proces koji obavlja ažuriranja. Postoje protokoli zasnovani na kvorumu i većinskom glasanju.
- **Protokoli bazirani na kvorumu** su definisani preko Giffordove šeme koja kaže da ako ima N replika, klijent mora da postigne read kvorum na skupu od najmanje N_R ili više servera. Da bi modifikovao fajl, write kvorum od najmanje N_W se zahteva. Moraju da važe sledeći uslovi za N_R i N_W: $N_R + N_W > N$; i $N_w > N/2$. Prvi uslov sprečava read-write konflikte, a drugi write-write konflikte. Samo ako se odgovarajući broj servera složi može se obaviti read ili write. Izbor servera može biti ROWA (read one, write all).
- **Keš koherentni protokoli:** klijenti vode računa da keš bude konzistentan, a ne server. Nekonzistentnosti se mogu detektovati statički – kompajler obavlja analizu pre izvršenja i otkriva koji su podaci potencijalni problem i ubacuje instrukcije da bi se izbegla nekonzistentnost, i dinamički – nekonzistentnost se detektuje u toku izvršenja. Konzistentnost sa kopijama na serveru se održava tako što kada se podaci modifikuju server šalje komandu invalidacije ostalim keševima ili šalje komandu ažuriranja.

5. Otpornost na greške

- **Tolerantnost sistema na defekte** predstavlja osobinu sistema da korektno funkcioniše uprkos pojavi grešaka. DS mogu biti otporniji na defekte od centralizovanih sistema, ali sa više hostova verovatnoća individualnih defekata se povećava. Defekt je trajni ili privremeni nedostatak koji se javlja u nekoj hw ili sw komponenti. Graške su manifestacija defekta i prestavljaju odstupanje vrednosti podataka od očekivane. Otkaz nastupa kada sistem više ne može da obavlja funkciju za koju je projektovan. Defekti se mogu javiti zbog grešaka u projektovanju, realizaciji, tehnologiji izrade ili zbog spoljnih uticaja. Greške se u odnosu na trajanje mogu podeliti na prolazne, periodične i stalne, a druga klasifikacija je na mirne (fail-silent, komponenta prestaje sa radom i ne generiše izlaz) i Vizantijske (komponenta nastavlja sa radom i generiše pogrešan rezultat).

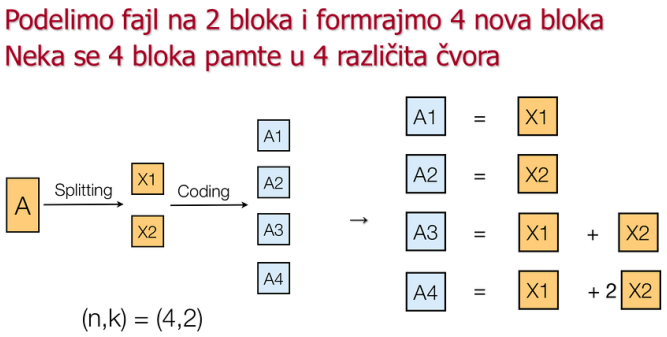
- **Redundansa** je opšti prilaz u projektovanju sistema otpornih na greške. Može biti informaciona (otpornost na greške se postiže repliciranjem ili kodiranjem podataka), vremenska (postiže se izvršenjem operacija više puta u vremenu, npr retransmisija) ili fizička redundansa (hardverska ili softverska).
- **Hardverska redundansa** je tehnika za postizanje visoke pouzdanosti kroz fizičku redundantnost. Najjednostavnija varijanta je TMR (Triple Modular Redundancy) u kojoj se koriste tri primera komponente da bi se detektovala i korigovala jednostruka greška. Ako hardver repliciramo tri puta, pouzdanost sistema će se povećati zato što postoje tri paralelna puta. TMR se primenjuje u sprezi sa glasačima koji ispituju da li je izlaz iz svake komponente isti.
- **ABFT** (Algorithm base fault tolerance) **tehnika** je tehnika koja se može primeniti kod određene klase algoritama, tipično za matrična izračunavanja.

$$A=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \quad A_c=\begin{bmatrix}1 & 1\\ 1 & 1\\ \hdashline 2 & 2\end{bmatrix}$$

$$B=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \quad B_r=\left[\begin{array}{cc:c}1 & 1 & 2\\ 1 & 1 & 2\end{array}\right]$$

$$C_f = A_c * B_r = \begin{bmatrix}1 & 1\\ 1 & 1\\ 2 & 2\end{bmatrix} * \begin{bmatrix}1 & 1 & 2\\ 1 & 1 & 2\end{bmatrix} = \left[\begin{array}{cc:c}2 & 2 & 4\\ 5 & 2 & 4\\ \hdashline 4 & 4 & 8\end{array}\right]$$

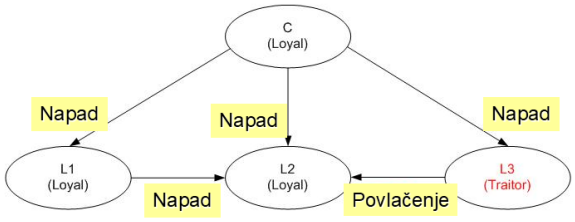
- **Erasure coding** je tehnika kodiranja kod koje se skup od k podataka kodira skupom od n, k<n podataka tako da se na osnovu bilo kog skupa od k podataka mogu regenerisati originalni podaci.



- **Usaglašavanje u prisistvu grešaka** je potrebno postići u konačnom broju koraka. Posmatraju se dva odvojena scenarija: Ispravni procesori a nesavršeni komunikacioni kanal (Problem dve armije), Nesavršeni procesori a pouzdani komunikacioni kanal (Problem vizantijskih generala).
- **Problem dve armije** pokazuje da i u prisustvu ispravnih procesora usaglašavanje između dva procesa nije moguće ako su komunikacioni kanali nepouzdani. Problem je poznat pod nazivom problem višestrukih potvrda.
- **Problem vizantijskih generala:** pouzdane komunikacije, nepouzdani procesori. Postoji n generala svaki sa svojom divizijom i treba da donesu odluku da li da napadnu neprijatelja ili ne. Komunikacije su pouzdane, ali m generala su izdajnici i pokušavaju ostale da spreče da postignu konsenzus šaljući

im netačne i kontradiktorne informacije. Svaki general daje svoj predlog za akciju: napad N ili povlačenje P šaljući poruku svim ostalim generalima, i prima poruke od ostalih sa njihovim predlozima.

- **Lamportov algoritam za problem vizantijskih generala:** zaključak je da bilo koje rešenje za prevazilaženje problema m generala izdajnika zahteva najmanje 3m+1 učesnika, odnosno više od 2/3 generala mora biti lojalno. Potrebno je obaviti m+1 rundu razmene poruka i razmeniti $O(mn^2)$ poruka. Problem n vizantijskih generala može se posmatrati kao n identičnih problema: postoji jedan general i n-1 poručnik, m poručnika (ili m-1 poručnik i general) mogu biti izdajnici. Lojalni general šalje svim poručnicima istu poruku. Svaki poručnik primljenu poruku prosleđuje ostalim poručnicima (ponaša se kao general). Kada se sve poruke prikupe, primenjuje se većinsko glasanje.



- n=4 generala; m=1 izdajnik
- L2 računa majority(Napad, Napad, Povlačenje) = Napad

- **Komunikacija sa RPC:** cilj RPC je da sakrije komunikaciju tako što pokušava da poziv udaljene procedure izgleda kao poziv lokalne. Ako nastupe greške ovo nije uvek moguće. Greške koje mogu nastupiti u RPC sistemu su: **Klijent nije u stanju da locira server** – razlog može biti otkaz servera ili pogrešna verzija klijent stub-a; **Zahtev upućen od klijenta ka serveru je izgubljen** – klijent treba da startuje časovnik kada uputi zahtev i ako istekne time out pre nego što stigne potvrda izvrši retransmisiju; **Otkaz servera nakon prijema zahteva od klijenta** – postoje varijante da server otkaže nakon izvršenja zahteva a pre slanja odgovora i da otkaže pre izvršenja zahteva. Klijent ne može razilovati ove sličajeve. Moguća su rešenja da klijent šalje zahteve sve dok ne dobije odgovor (at least one semantika) ili da se odma signalizira greška nakon isteka time-outa (at most one semantika). Najbolje bi bilo obezbediti tačno jedno izvršenje, ali nema načina da se to garantuje**; Odgovor servera klijentu izgubljen –** klijent može ponovo poslati zahtev serveru, ali server vodi računa o rednim brojevima klijentskih zahteva i može otkriti duplikat i odbiti da izvrši ponovo zahtev, ili postaviti u zaglavlju poruke RETRANSMISSION bit kako bi razlikovao originalni zahtev od retransmisije.
- **Strategije za oporavak od greške** podrazumevaju vraćanje sistema u korektno stanje. Može se postići na dva načina: **Povratak u stanje pre nastupanja greške** (Backward Recovery) – vratiti sistem u neko prethodno korektno stanje korišćenjem checkpoint-ova, a zatim nastaviti sa radom, Checkpointing može imati veliku cenu; **Prelazak u novo stanje** (Forward Recovery) – prevesti sistem u korektno novo stanje nakon čega se može nastaviti sa radom, neophodno je predvideti sve potencijalne greške.

6. NFS (Sun Network File System)

- **NFS** dozvoljava da svaka mašina bude i klijent i server u isto vreme. Svaki NFS server eksportuje jedan ili više svojih direktorijuma za pristup udaljenim klijentima. Kada se direktorijum učini dostupnim, svi poddirektorijumi postaju dostupni. Klijent pristupa eksportovanim direktorijumima tako što ih

montira na svoj fajl sistem (mount). Proces montiranja nije transparentan, već je potrebno navesti ime hosta na kome se nalazi direktorijum koji se montira. Sastoji se iz tri nivoa: **Nivo sistemskih poziva** – klijent pristupa fajl sistemu pomoću sistemskih poziva koje nudi lokalni OS; **Virtuelni fajl sistem** (VFS) – kada korisnik pristupa fajlu, njegov poziv prosleđuje VFS, takođe nije važno koji OS klijent i server koriste; **NFS klijent/server nivo** – sva komunikacija između klijenta i servera se odvija preko RPC. NFS server može montirati direktorijume koje eksportuju drugi serveri u svoj fajl sistem, ali ih ne može eksportovati svojim klijentima.

- **File handle** su identifikatori koji se koriste za pristup udaljenim fajlovima. Ime fajla se preslikava u fajl handle lookup komandom. Sadrži sve informacije koje su potrebne serveru da pristupi odgovarajućem fajlu. Sve operacije na udaljenom fajlu se izvode korišćenjem file handle kao identifikatora. VFS za svaki fajl zapamti v-node (virtuelni i-node) tako da ako je fajl lokalni v-node sadrži i-node (fajl ili direktorijum), a ako je udaljeni sadrži file handle.
- **Keširanje u NFS:** NFS koristi keširanje i na strani klijenta i na strani servera. Keširaju se fajlovi, delovi fajla, timestamps fajlova, file handle. Ako više klijenata čita isti fajl, svaki može dobiti kopiju fajla i zapamtiti je u kešu. Ako više klijenata vrši upis u isti fajl razultat može biti potpuno proizvoljan.

7. DFS (Distribuirani fajl sistemi)

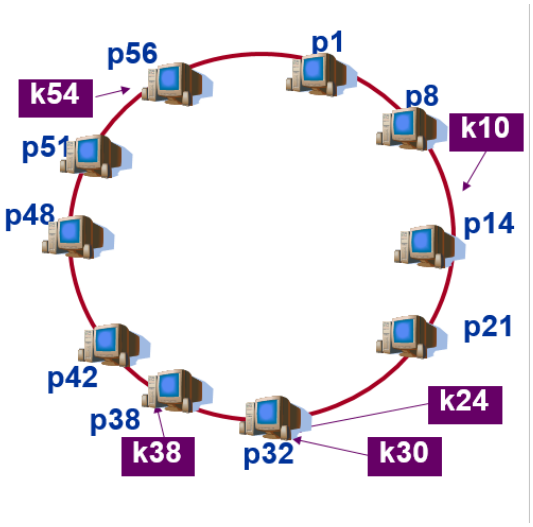
- **Fajl sistem** je odgovoran za organizaciju, skladištenje, imenovanje, pretraživanje, deljenje i zaštitu fajlova. Obezbeđuje korisnički interfejs za rad sa fajlovima. Smeštanje fajlova na disku može biti: **Kontinualno –** blokovi fajla su zapamćeni u sukcesivnim sektorima na disku, direktorijum ukazuje na prvi blok; **Ulančano** – blokovi jednog fajla mogu biti rasejani na disku, kao lančane liste, direktorijum ukazuje na prvi blok; **Indeksirano** – poseban blok sadrži pointere na blokove fajla, direktorijum ukazuje na indeks blok (i-node).
- **Implementacija DFS** zahteva sve komponente tradicionalnog fajl sistema sa dodatnim komponentama za klijent-server komunikaciju, imenovanje i lociranje fajlova u DS. Nalazi se na više autonomnih mašina i sastoji se od direktorijumskog servisa i fajl servisa.
- **Modeli pristupa udaljenom fajlu** mogu biti: **Upload/Download model** – jedini obezbeđeni servisi su read i write (read preuzima fajl sa servera i pamti ga na klijentu, operacije nad fajlom se vrše lokalno i sa write vraćaju na fajl server); **Model udaljenog pristupa** – fajl servis obezbeđuje izvršenje operacija na udaljenoj mašini, operacije se izvršavaju na serveru.
- **Stateless server** je server koji ne pamti nikakvu informaciju o klijentskim zahtevima. Sve informacije da bi se obslužio zahtev mora pamtiti i obezbediti klijent pri svakom zahtevu. Otporniji je na otkaz servera jer se nikakve informacije ne pamte na serveru. **Stateful server** pamti informacije o tome koji klijent je otvorio koji fajl. Kraće su poruke u kojima klijent šalje zahteve serveru, ali je teže postići tolerantnost na otkaze.
- **Imenovanje fajla** se može vršiti na tri načina: **Imenovanje fajlova kombinacijom imena hosta i lokalnog imena fajla** – garantuje jedinstvenost imena, ali je ime fajla lokaciono zavisno i nema transparentnosti; **Mounting** – lokaciona transparentnost se postiže nakon montiranja i montiranim direktorijumima se pristupa kao lokalnim; **Totalna integracija komponenti fajl sistema** – svi fajlovi u sistemu pripadaju jedinstvenom prostoru imena, ime fajla je isto bez obzira sa kog čvora se pristupa fajlu.

- **Sekvencijalna semantika:** obezbeđuje apsolutno vremensko uređenje svih operacija. Svaka read operacija vidi efekte prethodnih write operacija. Lako je ostvariti ovakvu semantiku u DFS ako svakim fajlom upravlja jedan server i klijenti ne keširaju fajlove, ali se onda javlja problem sa performansama. Ako se dozvoli keširanje javlja se problem konzistencije.
- **Semantika sesije:** Sesija predstavlja seriju pristupa između otvaranja i zatvaranja fajla. Promene učinjene na fajlu su vidljive samo klijent procesu, dok su vidljive drugim procesima tek nakon zatvaranja i ponovnog otvaranja fajla.
- **Politike upisa u keš** su: **Write-through** – zajedno sa modifikacijom keša modifikuje se i fajl na serveru; **Politika zakašnjenog upisa** – upis na server se obavlja nakon određenog vremena; **Upis u trenutku zatvaranja fajla** – redukuje se broj upisa na disk. Provera konzistentnosti može biti inicirana od strane klijenta ili servera.

8. Peer-to-Peer Sistemi

- **P2P arhitektura:** ne postoji server u centru aplikacije, već krajnji hostovi direktno komuniciraju. Peer hostovi se povremeno konektuju u mrežu i imaju promenljive IP adrese. Osnovni problem kod P2P aplikacija je efikasno lociranje čvorova koji sadrže tražene podatke. Mehanizmi za lociranje podataka mogu biti: **Centralizovani** – koriste jedan ili više servera koji sadrže centralne direktorijume gde svi peer čvorovi registruju svoje sadržaje koje su spremni da dele, ali nije pravi p2p već hibridni; **Lokalni** – svaki peer čvor sadrži tabelu sa sadržajem koji je spreman da deli sa drugima, pronalaženje se ostvaruje korišćenjem bujice; **Distribuirani** – informacije o podacima su rasejane po čvorovima u P2P mreži. Za pronalaženje informacija koriste se tabele slične ruting tabelama, ali se pretraživanje obavlja na osnovu ključa (Distributed Hash Table sistemi).
- **Centralizovani model** (Napster): kada aktivni peer dobije novi objekat ili ukloni objekat, on informiše centralni server koji zatim ažurira svoju bazu podataka. Da bi održao bazu podataka validnom server mora biti u stanju da odredi kada se neki peer čvor diskonektuje. Peer čvor se može diskonektovati zatvaranjem P2P aplikacije ili diskonektovanjem sa interneta. Ima slabu otpornost na greške i može biti bottleneck u velikim P2P sistemima.
- **Distribuirani model** (Gnutella): klijenti formiraju abstraktnu logičku mrežu koja čini graf (overlay mreža). Ako peer X ima TCP konekciju sa peer Y, tada postoji poteg između X i Y. Svi aktivni peer čvorovi čine overlay mrežu. Poteg ne predstavlja fizički link. Jedan peer čvor je najčešće povezan sa < 10 overlay suseda. Graf mreže nije struktuiran. Sadržaj se locira tako što peer šalje poruku sa upitom preko suseda dok se ne locira traženi fajl koji se vraća obrnutim putem. Peer čvor može dobiti kao odgovor na upit poruke od više drugih peer čvorova koji poseduju traženi fajl.
- **Hijerarhijska organizacija peer čvorova** (KaZaA): podseća na Gnutellu jer ne koristi server, ali nisu svi peer čvorovi jednaki, već čvorovi sa većom propusnom moći i većom povezanošću su lideri grupe. Svaki peer je ili lider ili član grupe tako da se uspostavlja konekcija između lidera i člana, kao i lidera međusobno. Lideri vode evidenciju o sadržajima svih članova grupe. Svaki fajl je identifikovan parom (ključ, ime_fajla) gde se ključ dobija primenom hash funkcije nad imenom. Klijent šalje upit postavljanjem ključnih reči lideru grupe koji ako pronađe uparivanje ključnih reči sa ključem šalje odgovor za svako poklapanje sa ključem i IP adresom. Ako ne nađe poklapanje, prosleđuje upit drugim liderima grupa.

- **Chord protokol:** ne postoji centralni server ili lider (super peer) već su svi čvorovi podjednako važni. Broj koraka da se locira željeni sadržaj raste logaritamski sa brojem čvorova u mreži. Koristi DHT za lociranje željenog objekta. Koristi m-bitne identifikatore raspoređene na prstenu veličine 2^m (chord ring). Identifikator peer čvora se dobija primenom SHA-1 funkcije nad IP adresom čvora – ID=hash(IP). Svaki peer čvor zna identifikator svog neposrednog prethodnika i sledbenika. Ključ za lociranje željenog objekta se dobija primenom SHA-1 funkcije nad imenom fajla – k=hash(name). Fajl kome odgovara ključ k se dodeljuje peer čvoru čiji je ID p tako da važi da je $k \leq p$ i ne postoji peer q za koji važi da je $k \leq q$ i $q<p$, gde je p prvi čvor na prstenu posmatrano u smeru kazalje na satu koji ima ID veći od ključa k.



- **Finger tabele:** svaki čvor sadrži informacije o malom broju drugih čvorova i zna više o čvorovima koji su mu bliži nego o onima koji su dalji. Sukcesivnim pretraživanjem će se pronaći čvor koji ukazuje na naslednika datog ključa. Kada se od čvora traži da pronadje lokaciju za zadati ključ, on će pretražiti svoju finger tabelu da bi pronašao prethodnika sa najvećim identifikatorom u odnosu na dati ključ. Vrsta finger tabele j čvora i sadrži informaciju za sledbenika $i+2^{j-1}$ mod 2^m, gde je m ukupan broj čvorova.

- **Pridruživanje čvora:** neophodno je obezbediti da pointer na sledeći čvor uvek bude validan bez obzira da li se novi čvorovi pridružuju ili odlaze iz mreže. Kada se novi čvor, n, pridružuje mreži, on poziva funkciju join(n^p) gde je n^p bilo koji poznati čvor na chord prstenu. Join funkcija traži od čvora n^p da pronađe neposrednog sledbenika za čvor n. Kada n sazna da je njegov sledbenik čvor n_s preko čvora n^p, n obaveštava n_s da je on novi prethodnik iza njega. Zatim n_s označava da je n njegov prethodnik i saopštava čvor koji mu je bio prethodnik n_p da ima novog prethodnika čima n_p označava n za sledbenika i n označava n_p za prethodnika.

- **Otkaz čvora:** ako čvor n primeti da je njegov neposredni sledbenik otkazao, zamenjuje pointer pointerom prvog živog čvora u svojoj listi iz finger tabele.