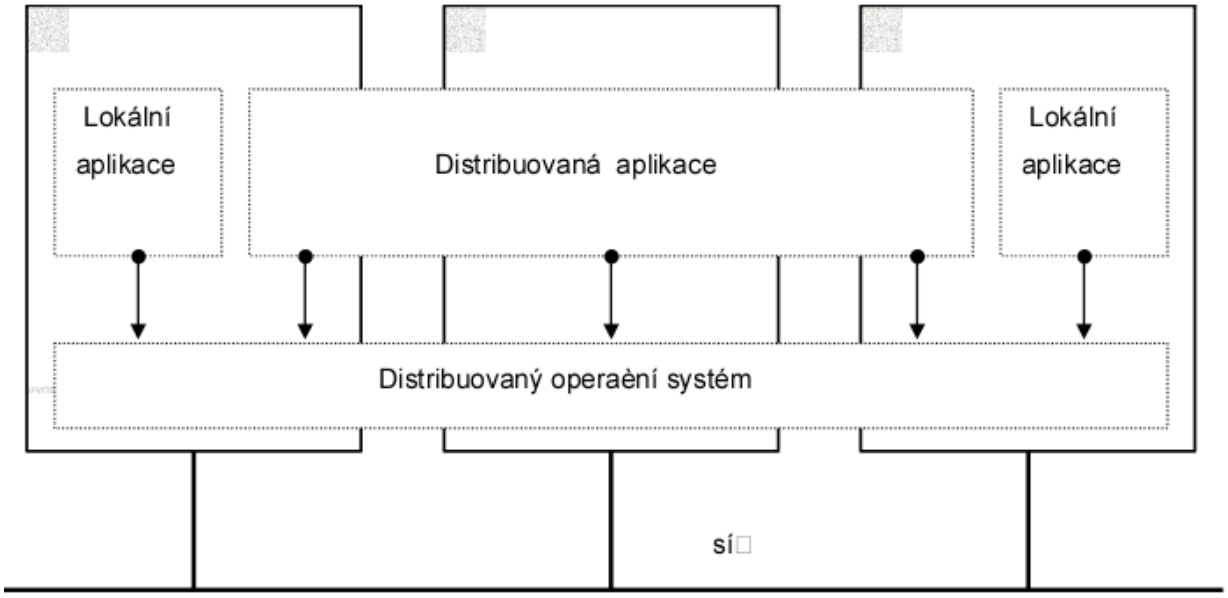
DS (distribuovaný systém)

- je systém propojení množiny nezávislých uzlů, který poskytuje uživateli dojem jednotného systému
- hardwarová nezávislost
    - uzly jsou tvořeny nezávislými “počítači” s vlastním procesorem a pamětí
    - jedinou možností komunikace přes komunikační (síťové) rozhraní
- dojem jednotného systému
    - uživatel (člověk i software) komunikuje se systémem jako s celkem
    - nemusí se starat o počet uzlů, topologii, komunikaci apod.
- celá škála různých řešení
    - multiprocesory na jedné základové desce
    - celosvětový systém pro miliony uživatelů
    - moderní technologie: cluster, cloud, grid, …

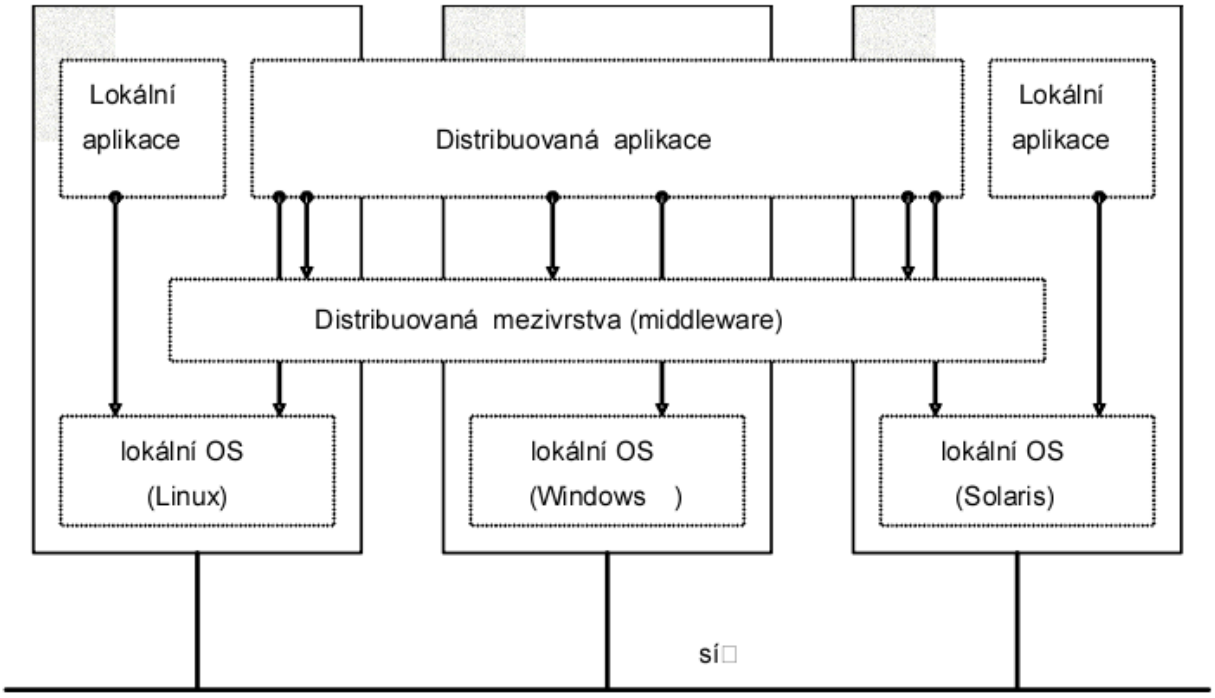
DS v 199x

- operační systém vyvinutý speciálně pro potřeby DS
- “zlatý věk”: 90. léta - Amoeba, Sprite, V, Chorus, Mosix ... T4
- součástí jádra OS podpora distribuovanosti, komunikace, synchronizace
- aplikace: transparentní využití distribuovaných služeb
- neexistuje rozdíl mezi lokální a distribuovanou aplikací
- nesplněná očekávání, žádný systém nedotažen do masově použitelného stavu



DS v 200x

- softwarový průmysl - jiný směr
- dominance MS Windows, UNIX / Linux
- použití některých principů a mechanismů distribuovaných systémů
- libovolný lokální OS, rozšiřující vrstva - distribuované prostředí + další služby
- prostředí pro distribuované aplikace – middleware (OSF DCE, CORBA, DCOM, Globe, …)



DS v 201x

- service-oriented computing
- SaaS, PaaS, IaaS
    - software as a service, platform as a service, infrastructure as a service
- nutná podpora virtualizace
- kompletní infrastruktura připravená k použití
- Windows Azure, Amazon Elastic Compute Cloud, Google App Engine, …

Distributed computing

- obecné principy distribuovaných systémů aplikovatelné na různá prostředí
    - cluster / grid (LHC-CERN) / cloud / sky computing
    - distributed data processing
    - distribuované databáze, NoSQL databáze, Big Data, Hadoop, data mining
    - mobilní systémy
    - bezdrátové sítě, routing, lokalizace, replikace
    - ubiquitní a pervasivní systémy
    - mikrouzly, každodenní sci-fi, samoorganizace, kolektivní znalost a rozhodování
    - distribuovaní agenti, inteligentní brouci
    - mobilita, migrace, task-oriented entities, kooperace, distribuovaná inteligence
    - peer-to-peer sítě
    - sdílení MM obsahu, P2P výpočty, distribuované ukládání, replikace, důvěryhodnost
    - content management
    - publish-subscribe, media distribution, streaming
    - …

Motivace a cíle návrhu DS

- motivace
    - ekonomika
        - síť běžných PC × supercomputer
    - rozšiřitelnost
        - přidání uzlu × upgrade centrálního serveru
    - spolehlivost
        - výpadek jednoho uzlu × výpadek CPU
    - výkon
        - technologické limity
    - distribuovanost
        - ‘inherentní distribuovanost
- cíle návrhu
    - transparentnost – přístupová, lokační, migrační, replikační, …
    - přizpůsobivost – autonomie, decentralizované rozhodování, …
    - spolehlivost
    - výkonnost
    - rozšiřitelnost – rozdílné metody pro 100 a 100.000.000 uzlů

Transparentnost

- na úrovni komunikace s uživatelem, komunikace procesů s jádrem
- stupeň transparentnosti x výkonnost systému
- přístupová
    - proces má stejný přístup k lokálním i vzdáleným prostředkům
    - jednotné služby, reprezentace dat, ...
- lokační
    - uživatel (proces) nemůže říci, kde jsou prostředky umístěny
    - nevyhovuje: počítač:cesta/soubor
- exekuční / migrační
    - procesy mohou běžet na libovolném uzlu / přemisťovat se mezi uzly
- replikační
    - uživatel se nemusí starat počet a aktualizaci kopií objektů
- perzistentní
    - uživatel se nemusí starat o stav objektů, ke kterým přistupuje
- konkurenční
    - prostředky mohou být automaticky využívány více uživateli
- paralelismová
    - různé akce mohou být prováděny paralelně bez vědomí uživatele

Přizpůsobivost

- autonomie
    - každý uzel je schopný samostatné funkčnosti
- decentralizované rozhodování
    - každý uzel vykonává rozhodnutí nezávisle na ostatních
- otevřenost
    - lze zapojit různé komponenty vyhovující rozhraní
    - oddělení interface x implementace
- migrace procesů a prostředků
    - procesy i prostředky mohou být přemístěny na jiný uzel

Rozšiřitelnost

- škálovatelnost, scalability
- rozdílné metody pro 100 uzlů × 100 milionů uzlů
- princip: vyhnout se čemukoliv centralizovanému
    - žádný uzel nemá úplnou informaci o celkovém stavu systému
    - uzly se rozhodují pouze na základě lokálně dostupných informací
    - výpadek jednoho uzlu nesmí způsobit nefunkčnost algoritmu
    - nelze spoléhat na existenci přesných globálních hodin
        - 'každou celou hodinu si všechny uzly zaznamenají poslední zprávu'

Další problémy

- koordinace a synchronizace
- geografická odlehlost, administrace

Techniky škálovatelnosti

- asynchronní komunikace
- distribuce
- caching a replikace

Výkonnost

- teorie: více uzlů vyšší výkon
- praxe: výrazně nižší než lineární nárůst výkonu
- příčiny: komunikace, synchronizace, komplikovaný software
- granularita výpočetní jednotky

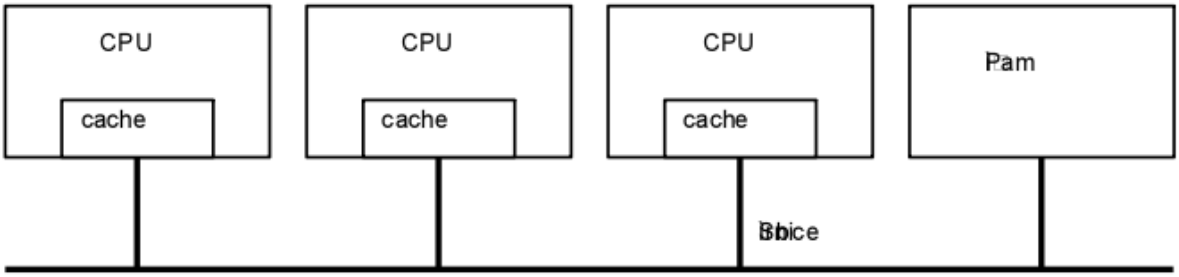
Chyby návrhu distribuovanách systémů

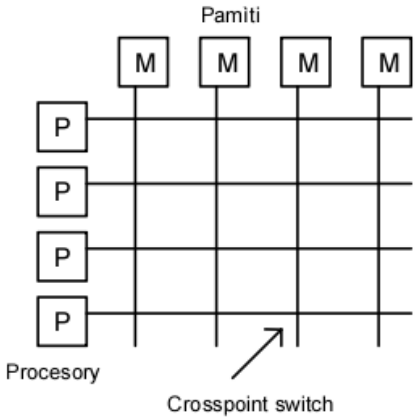
- síť je spolehlivá
- síť je zabezpečená
- síť je homogenní
- topologie se nemění
- nulová latence
- neomezená kapacita sítě
- jeden administrátor

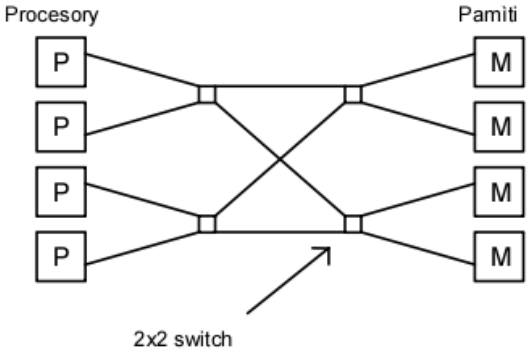
Paralelní architektury

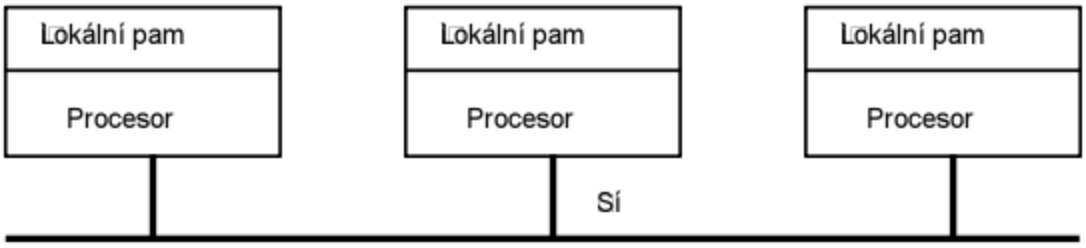
- multiprocesory
    - bus
        - sdílená sběrnice mezi procesory a pamětí
        - levná, běžně dostupná
        - max. desítky uzlů
        - cache – synchronizace
    - switched
        - možné zapojení i většího počtu uzlů
        - mřízka - crosspoint switch
            - sběrnicová mřížka - procesory x paměťové bloky
            - nákladné – kvadratický počet přepínačů
        - omega network
            - postupně přepínaná cesta mezi procesorem a pamětí
            - při větším počtu úrovní pomalé
            - počet přepínačů n * log n

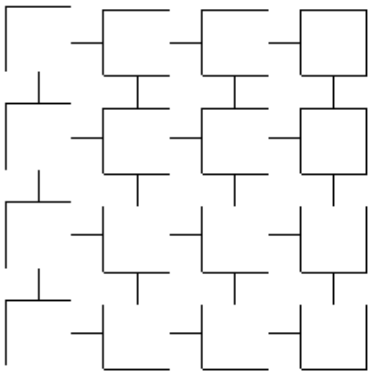
- multicomputery
    - bus
        - vlastní paměť každého uzlu → výrazně nižší komunikace
        - možnost teoreticky neomezeného počtu uzlů
        - levná, běžně dostupná
        - ‘normální’ počítače propojené ‘normální’ sítí
        - lze i jiné provedení: processor poo
            - rack full of cpus in the machine room, which can be dynamically allocated to users on demand
    - switched
        - mřížka
            - relativně jednoduchá (dvourozměrná) implementace
            - vhodné pro řešení ‘dvourozměrných’ problémů – grafy, analýza obrazu, ...
        - hyperkrychle
            - n-rozměrná krychle, častý rozměr 4
            - každý uzel přímo propojen se sousedy ve všech rozměrech
            - supercomputers - tisíce až miliony uzlů
                - 2012 IBM Sequoia Blue Gene - 1.5 mil cores, 1.5 PB RAM, 8 MW ..... Linux



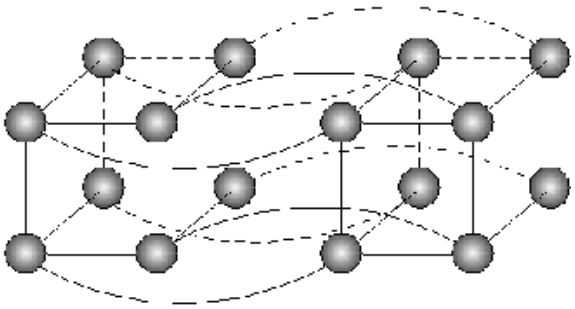






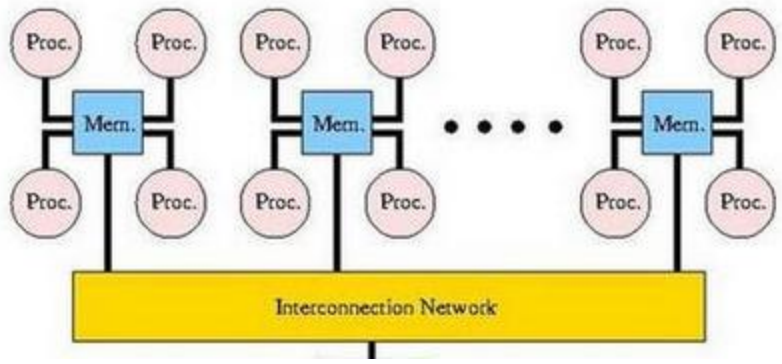


(a)



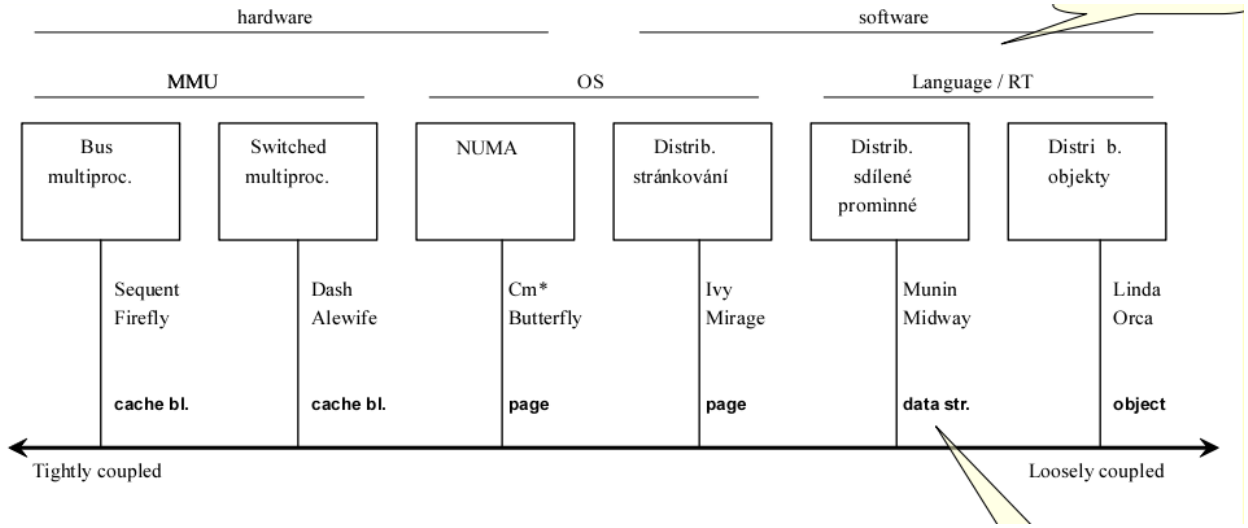
(b)

- NUMA
  - non uniform memory access
  - S-COMA - Simple Cache-Only Memory Architecture
    - požadavek na nelokální data je propagován vyšší vrstvě
  - ccNUMA - Cache Coherent NUMA
    - globální adresace, konzistence cache
  - NUMA-faktor - rozdíl latence lokálních a vzdálených paměťových přístupů



- GPGPU

Mechanismy sdílení paměti

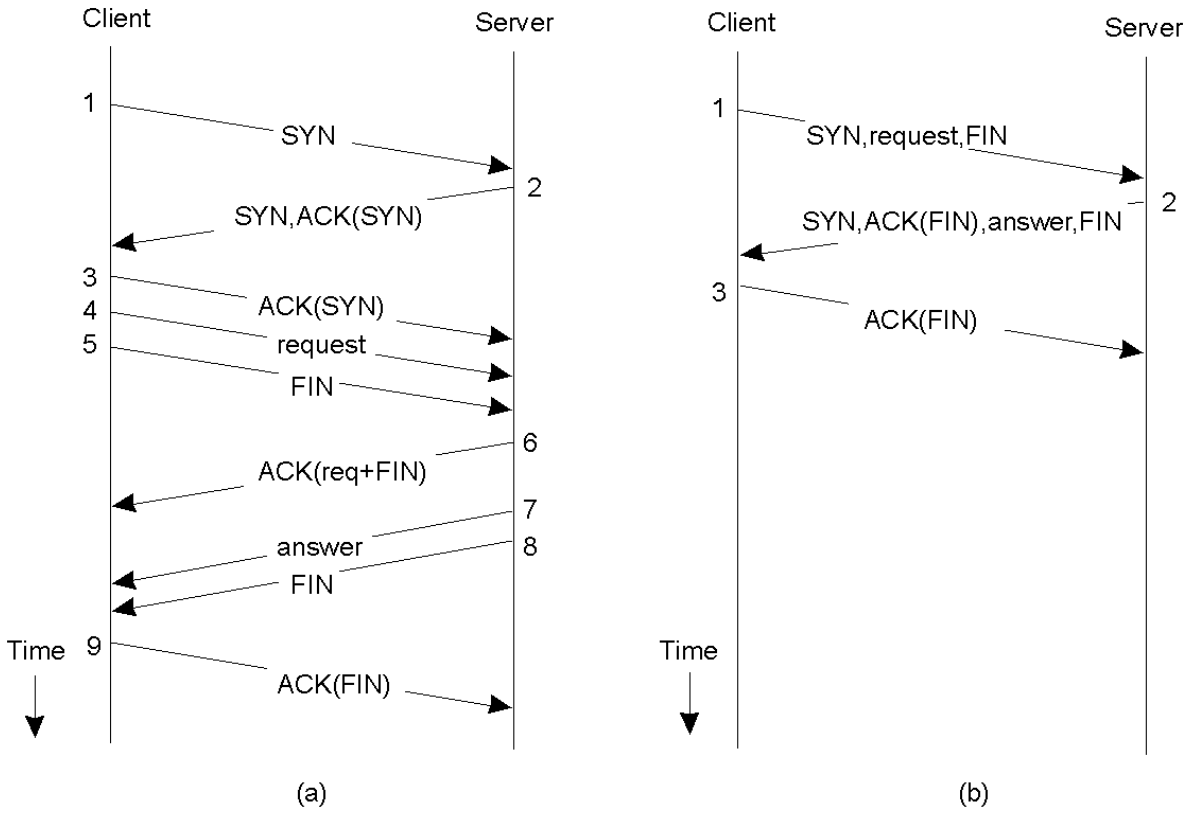


Komunikace

- absence sdílené paměti → zasílání zpráv
  - RPC, doors, RMI, M-queues, ...
- jednotný komunikační mechanismus
  - klient-server model
  - efektivita - lokální / vzdálený přístup

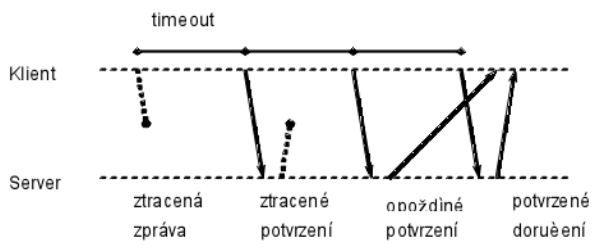
Síťová komunikace

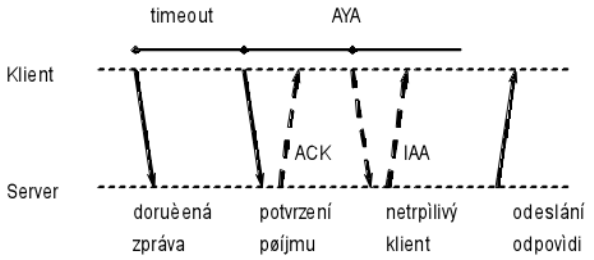
- TCP - standard pro spolehlivý vzdálený přenos
  - ale: většina komunikace je v rychlé a spolehlivé LAN
  - nevýhoda - složitost protokolu, menší výtěžnost sítě
- specializované protokoly pro (spolehlivé) lokální sítě
  - optimistické
  - vysoký výkon za cenu složitějšího zotavení z chyb
  - T/TCP, FLIP, …



Spolehlivost síťové komunikace

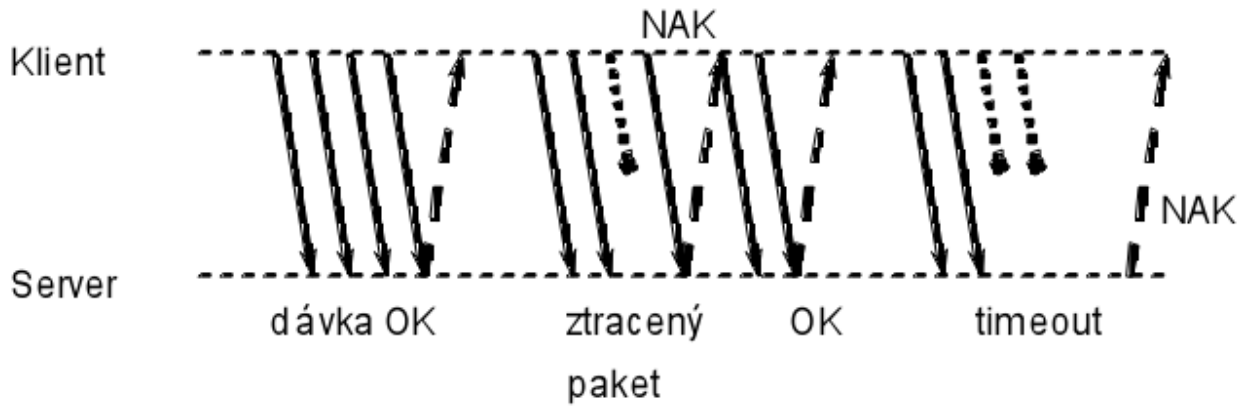
- nespolehlivý unicast – to co nám dává hardware, počítáme s best effort
    - v IPv4 se pakety mohou na routerech fragmentovat po cestě, v IPv6 vyrazí s nějakou délkou a s tou i dorazí (pokud po cestě nějaká část sítě neumí danou délku přenést, tak je zahodí)
- ochrana proti poškození
    - duplikace, checksums, parita, křížová parita, CRC
- ochrana proti ztrátě – potvrzování
- ochrana proti duplikaci – unikátní ID paketů
    - při TCP handshake se dohodne náhodné počáteční a pak roste
- duplicita zpráv - číslování, zahození - poslední zpráva / okno
- předbíhání zpráv - číslování, zpráva o nedoručení - režie, okno
- ztráta zprávy - klient vyslal žádost a neví .....
    - ztráta žádosti / ztráta odpovědi / příliš pomalá komunikace
    - typické řešení: potvrzování, timeout, opakování
- potvrzování → zvýšení režie → odpověď = potvrzení příjmu
- při delším zpracování - timeout serveru nebo klienta
- služební zprávy - ACK, NAK, AYA, IAA, CON, …





Přenos dlouhých zpráv

- jednotka přenosu = zpráva
  - příliš dlouhá → rozdělení na pakety
- co potvrzovat
  - každý paket → režie při správném přenosu
  - celá zpráva → režie při špatném přenosu
- možné řešení - dávky (buřty, blast, burst)
  - potvrzování dávky (definovaný počet paketů)
  - v pořádku celá dávka → ACK
  - předbíhání, výpadek → NAK
    - přenos celé dávky / od místa výpadku
  - ztráta posledního (-ích) paketů timeout
  - pevné dávky - jednodušší
    - při vyšší zátěži sítě velká chybovost
    - při volné síti nízká výtěžnost
  - dynamické dávky - úprava velikosti podle aktuální situace
    - n-krát správný přenos dávky → zvětšení
    - špatný přenos → zmenšení
    - vysoká výtěžnost aniž by jeden přenos omezoval ostatní
    - konsensus o velikosti dávky



Nespolehlivost serveru

- klientovi nepřichází odpověď
  - ztratila se zpráva s žádostí/odpovědí
  - server je zahlcen/je pomalý
  - server spadnul/nefunguje
- obecně nelze zjistit co se stalo
  - i když lze, pro některé služby je to náročné - např. transakce

Sémantika zpracování

- idempotentní služby
  - nevadí opakované provedení služby
  - sečti 1 + 1
- exactly once sémantika
  - pro některé služby nelze (tisk na tiskárně)
- at-least-once sémantika
  - služba se určitě alespoň jednou provede
  - nelze zajistit, aby se neprovedla vícekrát
- at-most-once sémantika
  - služba se určitě neprovede vícekrát
  - nelze zajistit, že se provede

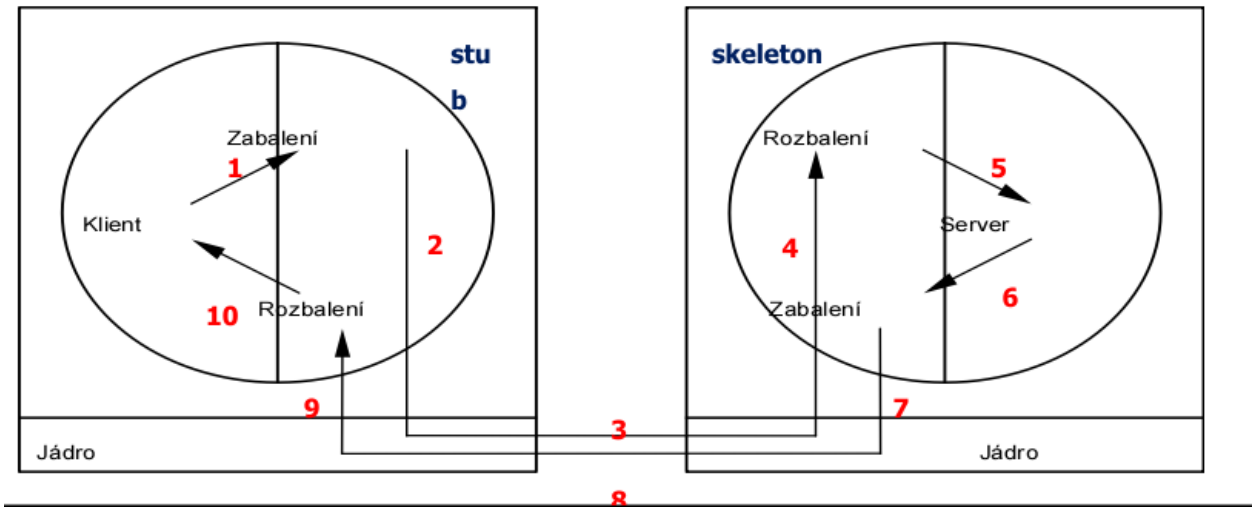
Havárie klienta

- zůstanou sirotci
  - procesy na serveru, ktere čekají na odpověď klienta, který ale spadnul
- typicky se to neřeší
  - specializované dlouhotrvající výpočty
- možnosti řešení:
  - exterminace
    - predtim, nez posle client stub RPC zpravu, zaloguje co bude delat
    - log se udrzuje na disku nebo na nejakem mediu, ktere prezije pad
    - po restartu se podiva do logu a sirotek se explicitne zabije
  - reinkarnace
    - cas se rozdeli do posloupnosti epoch
    - kdyz se klient restartuje, zapocne novou epochu - broadcastuje to zpravou vsem strojum
    - kdyz takova zprava prijde, vsechny vzdalene vypocty daneho klienta jsou zabity
    - mohlo se stat, ze doslo rozdeleni site (partition) a nejaci sirotci, prezili, pokud ale budou odpovidat, bude v odpovedi spatna epocha a tim se prokazou

- expirace
  - kazdy proces ma prideleno kvantum casu K
  - pokud nestiha, musi explicitne pozadat o dalsi - coz je neprijemnost
    - po restartu vsak muze klient pockat K casu a pak si muze byt jist, ze vsichni sirotci budou zabiti
  - vhodne K

Vzdálené volání pomocí zpráv (RPC)

- myšlenka je v tom, že na klientu se zavolá funkce, která se provede na serveru
- meziprocesová komunikace většinou příliš nízkoúrovňová
- idea: přiblížit zažitým mechanismům volání procedur
- RPC - remote procedure call
  1. klientova funkce zavola beznym zpusobem klientsky stub
  2. stub vytvori zpravu, zavola jadro (marshalling)
  3. jadro posle zpravu uzlu se serverem
  4. vzdalene jadro preda zpravu skeletonu - server-side stubu
  5. skeleton rozbali parametry a zavola vykonnou funkci (unmarshalling)
  6. server zpracuje pozadavky a beznym zpusobem se vrati do skeletonu
  7. skeleton zabali vystupni parametry do zpravy a zavola jadro
  8. jadro posle zpravu zpet klientovi
  9. klientovo jadro preda zpravu stubu
  10. stub rozbali vystupni parametry a vrati se ke klientovi



- spojení
    - nejprve se vygeneruje UUID rozhraní
    - programátor dodá definici rozhraní (interface definition file)
    - IDL kompilátor udělá header, co se naincluduje do klienta i serveru
    - IDL kompilátor taky udělá klientskou a serverovou část, která se stará o komunikaci (client stub, server skeleton)
    - programátor dodá implementaci serverové části
    - celé se to zkompiluje a slinkuje
    - server si zaregistruje u nějakého directory serveru to, že dělá tuhle službu
    - klient se rozběhne, když si chce zavolat RPC, tak:
        - normálně zavolá fci
        - vygenerovaný stub si vytvoří zprávu (marshalling), předá jádru jádro, look-upne službu na directory serveru, předá jí jádru serveru, to ji dá skeletonu, ten si ji rozbalí (unmarshalling), a putuje to podobně zpět
    - rozhraní může být i ze signatury v "normálním" jazyce, jen s nějakými doplňujícími informacemi
    - IDL – popisuje rozhraní funkcí, z něj se pak generují skeletony a stuby; pak mapování do jazyků (je to prog. jazyk)
    - varianta s objekty – máme proxy objekty u klienta a servanty na serveru
        - implementace tohohle objektovýho cirkusu je třeba RMI
- problémy
    - reprezentace dat
        - endiany
            - nejde to jen tak, ze reverzujeme poradi bytu, protoze normalne nevime o datech nic a mohli bychom takto reverzovat string, coz nechceme → nutne informace navic
        - floaty
        - ruzna kodovani retezcu
    - pristup ke globalnim promennym
        - neexistene sdilene pameti
        - mozne reseni pomoci DSM (distr. sdileni pameti), ale je to tezkopadne
    - predavani ukazatelu, dynamicke struktury, predavani poli
        - jde to napriklad tak, ze pokud vime ze pointer ukazuje do pole a vime jeho velikost tak udelame jeho kopii a dame ji do zpravy
        - problem, ze obecne nevim kam pointery ukazuji nebo jak jsou ty objekty velke
            - muze se pak napr. zpetne vyzadat po klientovi kopie daneho objektu
                - ztrata transparence, rezie navic
    - skupinova komunikace
    - komunikacni chyby
    - bezpecnost

- RPC je uzitecne a prakticky pouzitelne (a pouzivane), ale neni zcela transparentni, taky nutno davat pozor na omezeni
- dalsi mechanismy: MPI, message queues

Skupinová komunikace

- jeden odesílatel – více příjemců
- atomicita
    - doručení všem členům nebo nikomu
    - evidence a změna členství
- synchronizace
    - doručování od různých odesílatelů
    - příjem vs. doručení zprávy
    - doručovací protokol - sémantika
- uzavřená skupina
    - pouze členové skupiny
    - vhodné pro kooperativní algoritmy
- otevřená skupina
    - zasílat zprávy může kdokoliv
    - distribuované služby, replikované servery
- problém překrývajících se skupin

Synchronizace

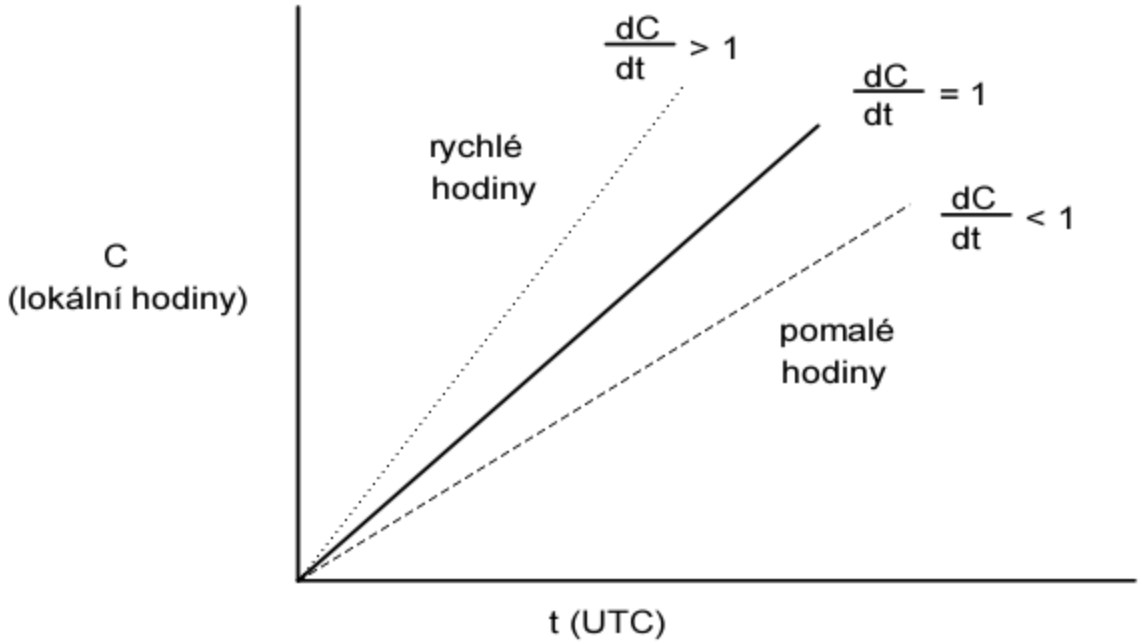
- logické a fyzické hodiny a jejich synchronizace
- distribuované vyloučení procesů
- volba koordinátora
- doručovací protokoly, virtuální synchronie, změny členství ve skupinách
- detekce globálního stavu
- distribuovaný konsensus
- neexistence sdílené paměti → zprávy
- rozprostřená informace mezi uzly
- rozhodování na základě lokálních informací
- vyloučení havarijních komponent
- neexistence společných hodin

Fyzické hodiny

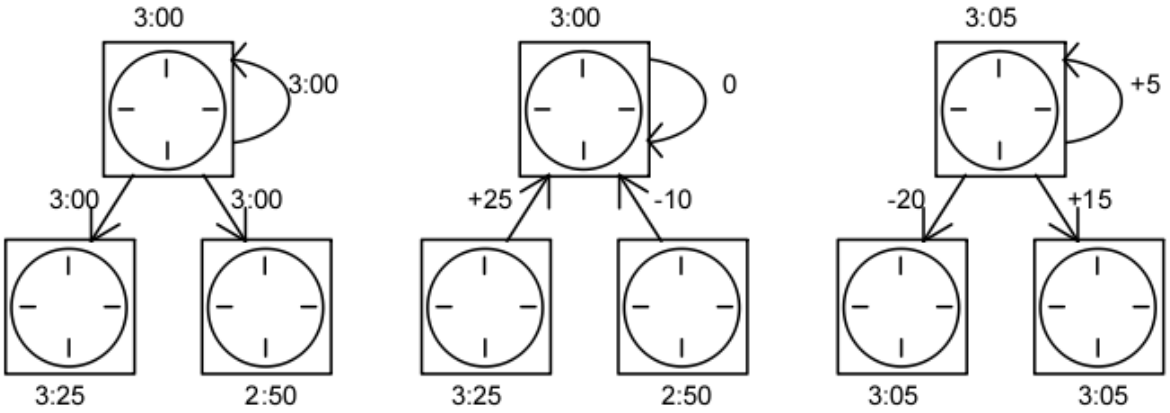
- jak sesynchronizovat lokální hodiny jednotlivých komponent systému
- jak sesynchronizovat jednotlivé hodiny mezi sebou
- astronomické měření → solární sec. = 1/86400 solárního dne
    - avšak slapové tření, atmosferické jevy → nepřesné
- 1948 - atomové hodiny - 1s ≈ 9 mld přechodů atomu cesia 133
    - nezávislé na okolních přírodních jevech, mnohem přesnější
- 1950 - TAI - Bureau International de l'Heure – průměr asi 50 laboratoří
    - international atomic time
    - 1 TAI den je však o 3 ms kratší než solární den
    - řešení: rozdíl >800 ms – vložená sekunda – Universal Coordinated Time UTC
- vysílání UTC (krátké vlny, satelit) – nepřesnost cca 0.1 - 10 ms

Synchronizace fyzických hodin

- jde určit po jaké době je nutné synchronizovat k udržené nějaké maximální odchylky od správného času
    - mira presnosti ρ: 1-ρ <= dC/dt <= 1+ρ, presne hodiny dC/dt = 1



- Cristianův algoritmus
    - jeden pasivní time server (UTC)
    - periodické dotazy v intervalech < δ/2ρ (kde delta je pozadovana snesitelna odchylka)
    - T = T_UTC + (T1 - T0 - I)/2
        - T0 cas zadosti, T1 cas odpovedi
        - pokud zname dobu zpracovani (interruptu) I, muzeme ji zohlednit a zpresnit odhad
    - nikdy nepřeřizovat najednou !
        - (dis)kontinuita
        - postupná změna - serie mereni
        - zrychlování nebo zpomalování - dynamicky
- Berkeley algoritmus
    - mame aktivni time server (time daemon) - treba na jednom ze stroju
    - demon se periodicky pta kazdeho stroje, jaky ma cas a na zaklade vsech odpovedi vypocita prumerny cas
    - kazdemu stroji pak posle, o kolik si maji prenastavit cas



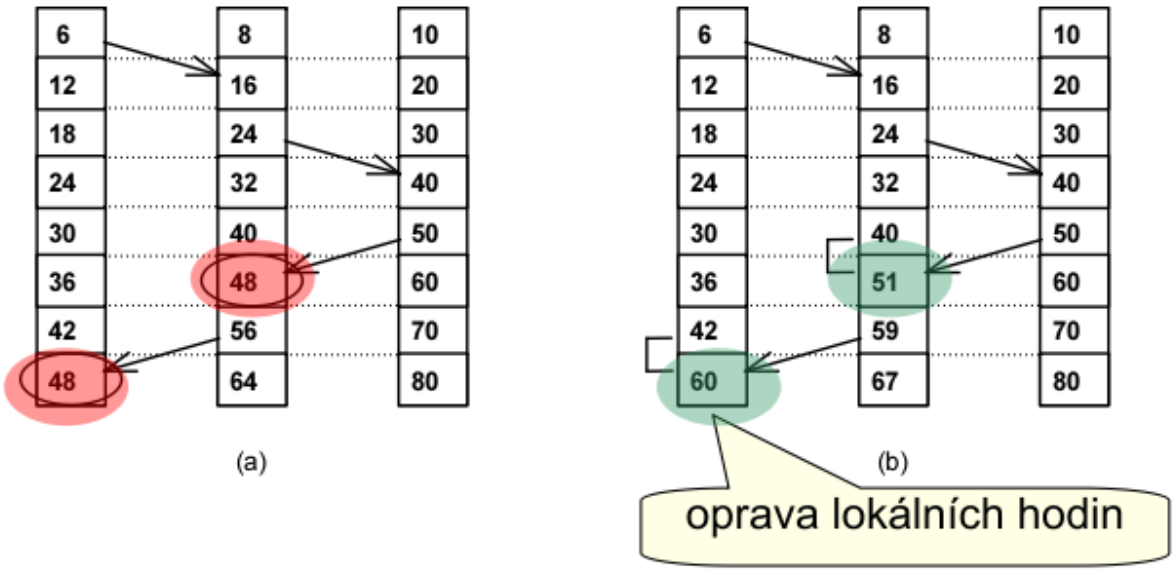
- distribuovany algoritmus (averaging)
    - predesle metody byly centralizovane (se znamymi negativy)
    - zde zadny server ani koordinator
    - rozdeleni casu do resynchronizacnich intervalu pevne delky
        - i-ty zacina na T0 + iR a konci T0+ (i+1)R
            - T0 domluveny moment v minulosti
            - R systemovy parametr
    - na zacatku kazdeho intervalu kazdy stroj broadcastuje svuj cas
    - kazdy stroj po odeslani casu zapne svuj lokalni timer a v nejakem intervalu S ceka na odpovedi od ostatnich
    - kdyz vse dorazi, spocita se prumer (prip. se zahodi extremy)
- intervalovy cas
    - cas neni okamzik, ale interval (nektere udaje jsou neporovnatelne)
    - v DCE (Distributed Computing Environment - DS z 1990s)

Logické hodiny

- problém: fyzické hodiny nelze dostatečně přesně sesynchronizovat
- Lamport: důležité pořadí, nikoliv přesný čas
    - nekomunikující procesy nemusí být sesynchronizovány
- kauzalni zavislost
    - jestliže p: e1 →p e2 potom e1 → e2
        - →p v ramci procesu p
    - forall m: send(m) → rcv(m)
    - jestliže (e1 → e2 & e2 → e3) ⇒ e1 → e3
    - happens-before relace a → b
        - vsechny procesy se shodnou, ze a probehl pred b
    - konkurentni udalosti: e1 -/-> e2 & e2 -/-> e1
        - nic o nich nemuze byt receno
- udalost a, cas C(a): pokud a → b ⇒ C(a) < C(b)

Synchronizace logických hodin

- podle přijímání zpráv
    - čas. značka zprávy m: Tm
    - proces i vysílá v čase Ci(a) zprávu m ⇒ Tm = Ci (a)
    - proces j přijme zprávu m v čase Cj(b) ⇒ Cj = max(Cj(b), Tm+1)
    - zuplneni
        - událost a v procesu i, událost b v procesu j
        - C(a)=C(b) & Pi<Pj ⇒ C'(a)<C'(b)
        - 'byrokratické' uspořádání



- plati, ze jestlize a → b ⇒ C(a) < C(b)
- neplati ale opacna implikace, coz by se vsak hodilo
    - jak tedy zjistit kauzalitu událostí ⇒ vektorové hodiny

Vzájemné vyloučení procesů

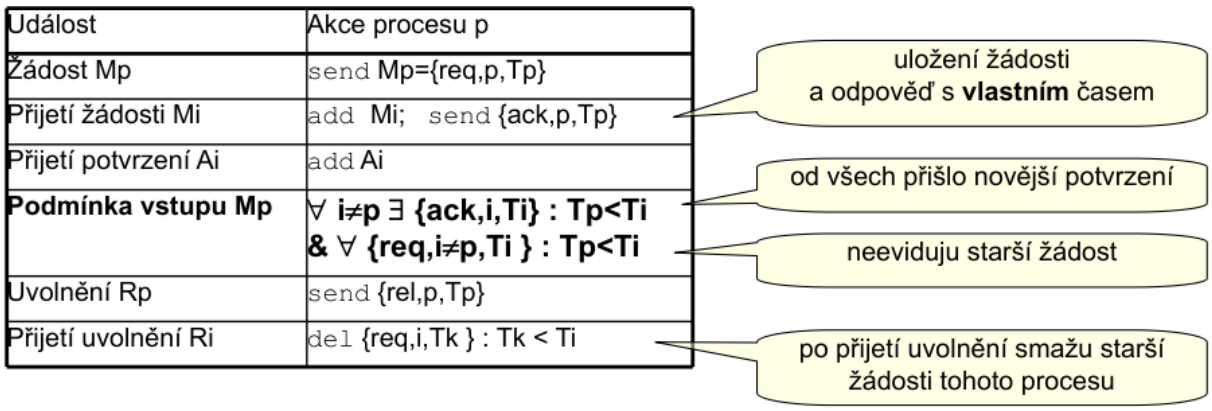
- na jednoprocesorovem systemu jsou kriticke sekce chraneny semafory, monitory a podobnymi konstrukty
- v DS pouze zpravy
- nutno zajistit korektnost
- reseni:
    - centralizovane
    - permission-based
        - casove znacky
            - Lamport - 3n
            - Ricart-Agrawala - 2n
        - volby
            - Maekawa - sqrt n
            - Agrawala-El Abadi - log n
    - token-based
        - token-passing - Suzuki-Kasami
        - strom - Raymond
        - token ring

Centralizovaný algoritmus

- jeden proces vybran jako koordinator
- chce-li nekdo do kriticke sekce → posle koordinatorovi zpravu, pokud tam aktualne nikdo neni, posle mu potvrzeni
    - pokud neni, tak jsou mozne implementace:
        - neposle mu nic - blokuje ho
        - posle mu deny a po opusteni aktualniho mu posle OK
        - kazdopadne si udrzuje frontu, kam si vklada procesy, kteri chteji do dane kriticke sekce
- centralizovana komponenta - ideove nevhodne
    - server muze predstavovat performance bottleneck
    - vypadek serveru - ztrata informace, nutnost volby noveho koord.
    - vypadek klienta - problem vyhladoveni (starvace)

Lamport algoritmus

- idea: proces vyšle žádost a čeká až:
    - dorazí odpovědi od všech procesů a zároveň
    - všechny žádosti v jeho frontě mají větší časovou značku (čili on má nejstarší)
- komunikační složitost 3n



Ricart & Agrawala

- vylepseni Lamporta
- proces chce vstoupit do kritické sekce:
    - zašle žádost s TS a čeká na odpovědi s potvrzením
- proces přijme zprávu se žádostí:
    - jestliže není v kritické sekci a ani do ní nechce vstoupit, pošle potvrzení
    - jestliže je v kritické sekci, neodpovídá, požadavek si zařadí do fronty
    - jestliže není v kritické sekci, avšak chce do ní vstoupit, porovná TS žádosti s vlastní žádostí:
        - vlastní žádost má nižší časovou značku (= starší), neodpovídá a zařadí žádost odesilatele do fronty
        - v opačném případě (žádost odesilatele je starší) pošle zpět potvrzení
- po opuštění kritické sekce pošle proces potvrzení všem procesům, které má ve frontě
- komunikační složitost 2n

Princip voleb

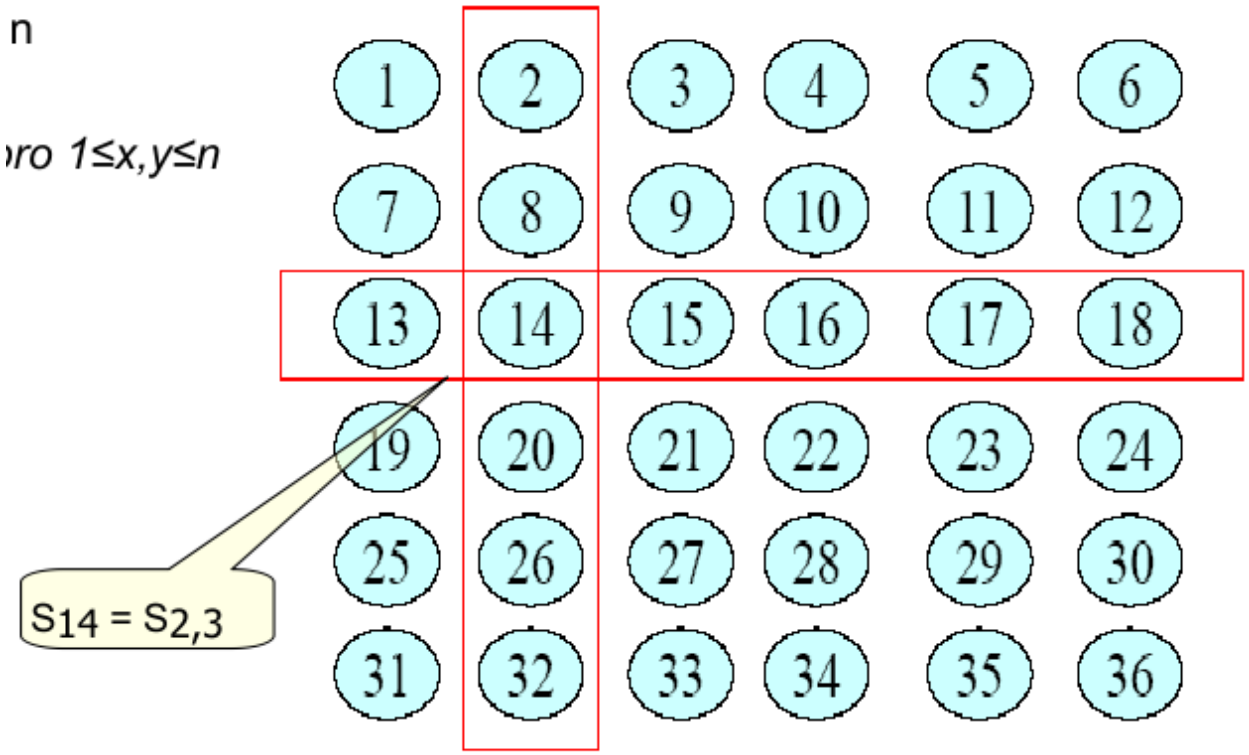
- základní princip:
    - procesy se snaží získat hlasy ostatních procesů
    - každý proces má v jeden okamžik právě jeden hlas
        - může ho dát sobě nebo jinému procesu
    - před vstupem do kritické sekce žádost o hlasy
        - pokud proces dostane víc hlasů než jakýkoliv jiný, může vstoupit
        - pokud dostane méně, čeká dokud nedostane dostatečný počet
    - problém: jak hlasovat a počítat výsledky, kdy už proces ví že vyhrál
- naivní volby
    - každý proces zná všechny ostatní
    - při žádosti jiného procesu dá proces hlas (pokud ještě nehlasoval)
    - relativně odolný proti výpadkům
        - vydrží výpadek až poloviny procesů

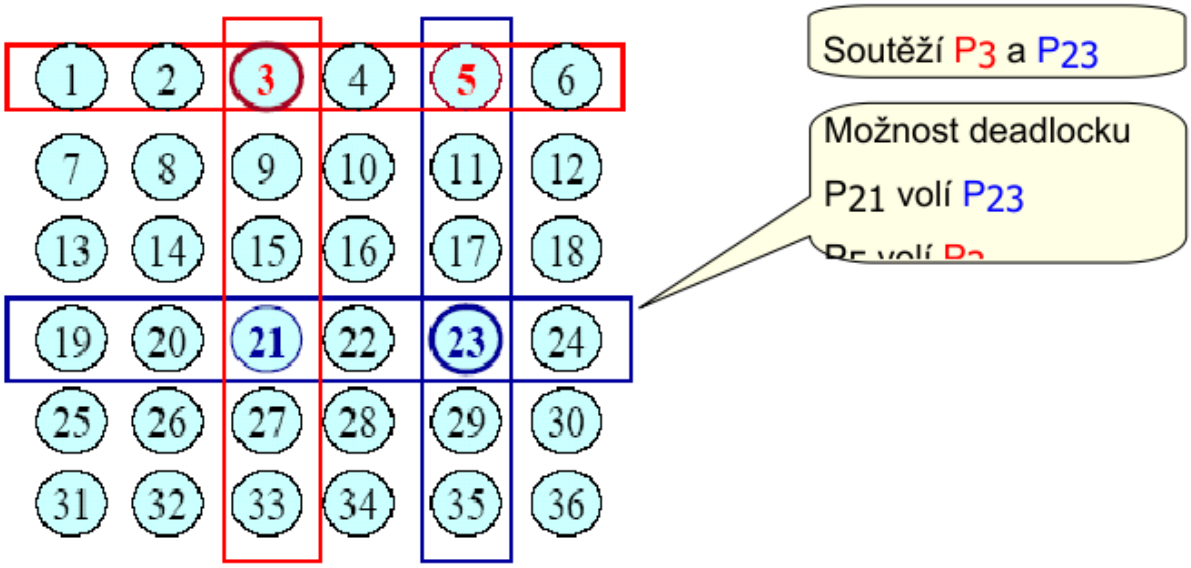
    - složitost O(n)
        - není lepší než TS
    - možnost deadlocku!
        - více procesů může dostat stejný počet hlasů

Maekawa

- organizace procesů pro optimalizaci komunikační složitosti
- každému procesu p je přiřazen volební okrsek Sp (voting district)
- pro vstup do kritické sekce je nutné získat všechny hlasy z vlastního okrsku
- podmínky pro okrsky:
    - forall p, q: Sp ∩ Sq ≠ 0
        - každá dvojice kandidátů má alespoň jednoho společného voliče
            - zaručí korektnost
        - každý proces může volit pouze jednou, nemohou být současně zvoleny dva procesy
    - forall p, q: |Sp| = |Sq| = K
        - velikost volebních okrsků je konstantní, procesy potřebují stejný počet hlasů
            - zaručí spravedlnost
    - forall p, q: |Si: p in Si| = |Sj: q in Sj| = D
        - p je obsažen ve stejném počtu D volebních okrsků
        - každý proces má stejnou zodpovědnost
- důsledek: komunikační složitost O(|Sp|)
    - chceme tedy minimalizovat velikost okrsků
- jak velke maji byt okrsky, v kolika okrscich ma byt kazdy proces ?
    - K velikost okrsku, N # okrsku, M # procesu, D (min) # okrsku, ve kterych je kazdy proces
    - dusledek podminek: N*K = D*M - TODO: Proc?
    - dulezite je uvedomit si, ze:
        - proces ≈ okrsek - protoze proces musi ziskat vsechny hlasy sveho okrsku
        - tedy ∑ procesu ≈ ∑ okrsku → M=N → K=D
        - protoze kazdy q z okrsku Sp je obsazen v dalsich D-1 okrscich, tak pri zachovani podminky pruniku dostavame max pocet okrsku N = (D-1)*K + 1
        - a tedy M = (K-1)*K + 1 → K = O(√M)
    - jak rozdelit procesy do okrsku
        - optimalne je to slozite - vyzaduje restrukturalizaci pri zmene clenstvi
        - suboptimalne (pri zachovani O(sqrt M) jednoduche)
            - $M = n^2$, Pi,j , Si,j , $1 \leq i,j \leq n$
            - Si,j = (∪Pi,x) ∪ (∪Py,j), pro $1 \leq x,y \leq n$
    - muze dojit k deadlocku

- reseni je pouzit Lamportovy hodiny
  - při příjmu žádosti procesu r s TSr procesem p
    - p má volný hlas: potvrzení ACKr
    - p dal již hlas jinému procesu q s TSq < TSr: zařadit do fronty
    - p dal již hlas jinému procesu q s TSq > TSr: pošle zprávu REJECT procesu q
      - race condition?
        - kdyz q jeste neni v kriticke sekci - vrati hlas
          - prideleni procesu s nizsim TS
        - pokud uz je, vrati ho pri opusteni CS





Agrawal & El Abbadi

- volby v O(log n)
- kvorum je cesta od korene k listu
- fault tolerance - nahrazeni havarovaneho uzlu jeho podstromy

Raymond

- token-based, O(log n)
- kostra, nezakoreneny strom
- kazdy uzel si udrzuje referenci na souseda smerem k token holderu
- pri prenosu tokenu se presmeruji reference - rekonfigurace

Token ring

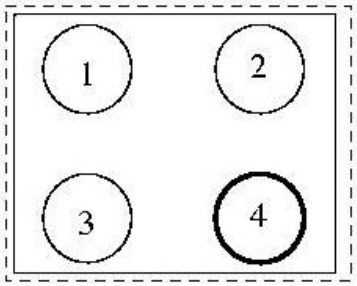
- token-based, komunikační složitost 1 až ∞
- logicky kruh procesu, nejak ocislovane, kazdy vi kdo nasleduje
- přenos zprávy obsahující povolení ke vstupu do kritické sekce
    - dostane token
        - chce vstoupit - provede CS a posle token dal
        - nechce - posle dal
- problemy:
    - dlouhy cekaci cas na token
    - token se ztrati - nutno regenerovat
        - spatne se detekuje i diky minulemu problemu
    - vypadek procesu
        - detekce - soused mu chce dat token a nejde to
        - vymaze se z kruhu, rekonfigurace

Bully algoritmus

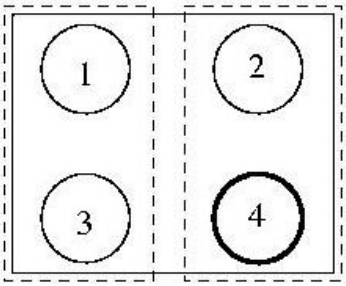
- volba koordinatora
- predpokladem je synchronni spolehlivy system
- Tm ~ message propagation time, Tp ~ message handling time
- omezená doba přenosu zprávy Tm
- omezená doba zpracování zprávy a odpovědi Tp
- detektor havárie v čase: 2Tm+Tp (velmi silný požadavek)
- když se proces rozhodne volit, zašle zprávu všem procesům s vyšší identifikací (číslo procesu)
    - když přijde odpověď, proces končí
    - když nepřijde nic, proces vyhrál, je novým koordinátorem, pošle zprávu všem ostatním
- když proces přijme zprávu o volbě, vrátí zpět odpověď a vyšle žádosti všem vyšším procesům
- volba ve dvou kolech
    - prijde vic odpovedi od procesů s vyssim PID a ty pak maji mezi sebou vyresit kdo je koordinator ve druhem kole
- pri prekroceni casovych limitu moznost vice koordinatoru !

Invitation algoritmus

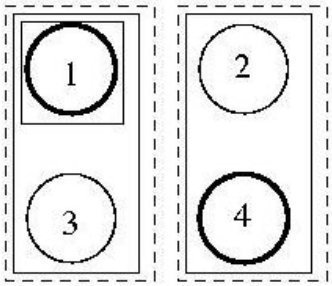
- volba koordinatora
- asynchronni system - nelze predpokladat nic o delce udalosti
- pouziti v “realnem” prostredi
    - procesor: může havarovat (fail-stop), neomezená doba reakce
    - komunikace: rozpojení skupiny na segmenty, ztráta zpráv
    - nelze spolehlivě detekovat havárii: absence odpovědi neznamená havárii
- koordinátor je vázán na skupinu, skupiny lze štěpit
    - všichni členové stejné verze skupiny (pohled, view) vidí stejného koordinátora
- koordinátor: pravidelná výzva AreYouCoordinator
    - pokud dojde k příjmu AYC koordinátorem ⇒ sjednocení do skupiny koordinátora s vyssim ID
- pokud člen skupiny neobdrží AYC svého koordinátora (dostatečně dlouho)
    - prohlásí se za koordinátora vlastní nové skupiny
    - posílá pozvání ke sjednocení ostatním novým skupinám
- konzistence relativní vzhledem ke skupině
    - procesy se shodují na členství ve skupině a na nějaké hodnotě
    - koordinátor vyzve ostatní k připojení
    - pokud se nepřipojí, jsou konzistentní samy se sebou



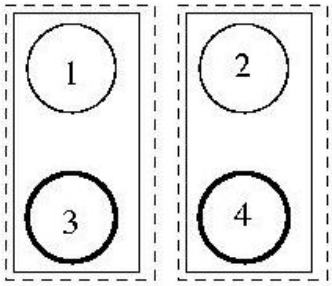
One partition. One group
Coordinator = Processor 4



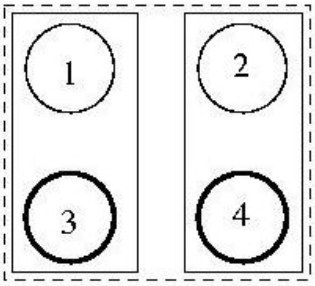
The network is partitioned, but no one notices



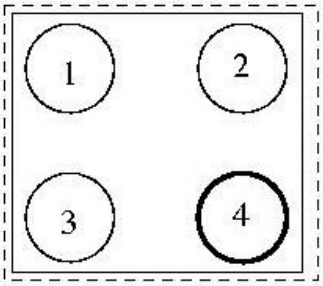
Processor 1 notices the partitioning. and declares itself a coordinator. It calls out and reaches processor 3.



Processor 3 realizes the partitioning and becomes a coodinator. Processor 1 retires from this role.



The network partition is repaired,. but none of the processors notice



Either processor 3 or processor 4. discover that the network has been repaired. Processor 4 responds by announcing that it is coordinator and restoring global consistency.

Kruhový algoritmus

- volba koordinatora
- proces se rozhodne volit (koordinator je nedostupný), pošle zprávu následníkovi
- zpráva obsahuje čísla procesů - kandidáti na koordinátora
    - každý při obdržení zprávy přidá sebe na seznam
- po návratu k iniciátorovi obsahuje zpráva nového koordinátora
- následuje fáze oznámení nového koordinátora
    - změní se typ zprávy
    - koordinátor je ten s nejvyšším id
    - zároveň zpráva obsahuje seznam žijících - tak jak se zapsali na seznam
- stačí znát následníky a mít možnost zjistit následníka nedostupného uzlu
- složitost $O(n^2)$
    - optimalizace - koordinator jen sveho okoli $2^k \rightarrow O(n \log n)$
        - implementacne narocnejsi
        - vyhodne pro velke skupiny a vysoke frekvenci konkurencnich voleb - caste vypadky

Doručovací protokoly skupinové komunikace

- globální uspořádání
  - zprávy jsou doručovány v pořadí odeslání
  - nelze – neexistence globálních hodin
- sekvenční uspořádání
  - všechny uzly doručí zprávu ve stejném pořadí
  - doručení nezávisí nutně na času odeslání
  - sekvencer, dvoufázový distribuovaný alg.
- atomický multicast
  - spolehlivé sekvenční doručení
- kauzální uspořádání
  - kauzálně vázané zprávy ve správném pořadí
  - konkuretní zprávy v libovolném pořadí
- kauzálně vázané zprávy
  - obsah vázané zprávy může být ovlivněn
  - není podstatné, jestli obsah byl ovlivněn
- konkurentní zprávy
  - ty, které nejsou kauzálně vázané

Kauzální doručování

- kauzalni zavislost - viz logicke hodiny (zpet)
- kauzalni usporadani dorucovanych zprav
  - dest(m) množina procesů, kterým je zaslána zpráva m
  - $deliver_p(m)$ je událost doručení zprávy m procesu p
  - $m1 \rightarrow m2 \Rightarrow \forall p \in (dest(m1) \cap dest(m2)) : deliver_p(m1) \rightarrow_p deliver_p(m2)$
    - jestlize je m2 kauzalne zavisla na m1, potom na vsech procesech, kterym se doruci obe tyto zpravy, se zachova poradi tohoto doruceni

Vektorové hodiny

- mame vektor delky n, kde n je # procesu ve skupine
- uvazujeme casovou znacku procesu VT(p) a znacku zpravy VT(m)
- pravidla aktualizaci casovych znacek
  - start $\rightarrow VT(p_i) = 0^n$
  - dojde-li k nejake interni udalosti v procesu p_i, inkrementuje si vlastni logicke hodiny - tedy: $++VT(p_i)[i]$
  - vzdy, kdyz proces p_i pripravuje poslani zpravy, tak si inkrementuje vlastni hodiny a cely svuj vektor zabali do zpravy: send $p_i(m) \sim VT(m) = ++VT(p_i)[i]$
  - p_j si pri doruceni zpravy m upravi $VT(p_j)$:
    - po slozkach maxima z prijateho a sveho vektoru
      - $\forall k: VT(pj)[k] = \max(VT(pj)[k], VT(m)[k])$
    - obecna pravidla nerikaji KDY k doruceni dojde !
  - libovolne dve zpravy nemaji stejne vektory:
    - $\forall mi,mj \ (i \neq j): VT(mi) \neq VT(mj)$

- porovnavani casovych znacek
    - ne kazda dvojice VT musi byt porovnatelna - tj mohou byt konkurentni
    - VT1 <= VT2 ⇔ forall i: VT1[i] <= VT2[i]
    - VT1 < VT2 ⇔ VT1 <= VT2 && exist i: VT1[i] < VT2[i]

Doručovací protokol pro jednu skupinu

- využití VT
- odeslání zprávy m procesem pi
    - klasicky
    - VT(pi)[i] ++
    - VT(m) = VT(pi)
- prijeti zpravy m od procesu pi procesem pj
    - proces pj ≠ pi pozdrží doručení m a posle jej dale, pokud dojde ke splneni nasledujicich podminek:
        - VT(m)[i] = VT(pj)[i] + 1 a
        - VT(m)[k] ≤ VT(pj)[k] pro k ≠ i
        - tj. toto jsou podminky doruceni
            - napr. proces pv poslal zpravu s clankem ~ a a proces pj posila reakci na tuto zpravu ~ m
            - prvni podminka ~ m je nasledujici zprava, kterou proces pj ceka od pi
            - druha podminka ~ pj nevidel zadne zpravy, ktere nevidel pi ve chvili posilani zpravy m, tzn pj jiz videl zpravu a (videl jiz vse, co videl pi pri odesilani zpravy m)
- doruceni zpravy
    - po doručení si proces pj upraví VT
        - klasicky
        - forall k: VT(pj)[k] = max(VT(pj)[k], VT(m)[k])

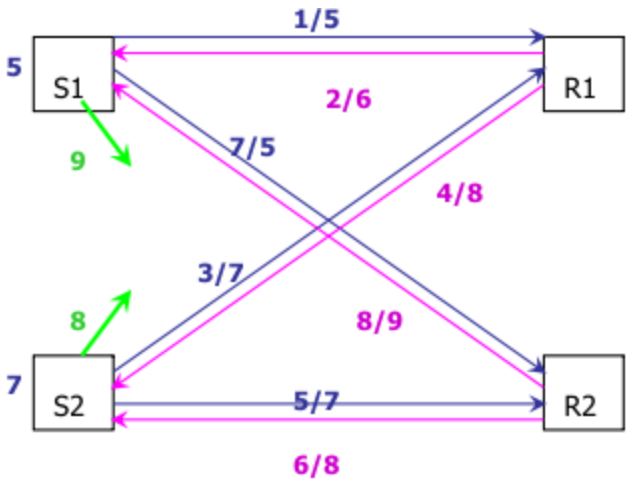
Doručovací protokol pro překrývající se skupiny

- maticovy protokol
    - vektory vektorovych hodin
- VTa - časová značka skupiny ga
- VTa[i] - počet multicastů procesu pi do ga (zprav, ktere vysle do skupiny)
- s odesílanou zprávou posílá proces VT všech skupin, kterých je členem
- odeslání zprávy m procesem pi do skupiny ga
    - VTa(pi)[i]++
    - VT(m) = ∪ VTb(pi), ze pi in gb (vsechny skupiny, do kterych pi patri)
- prijeti zpravy m od procesu pi procesem pj ze skupiny ga
    - proces pj ≠ pi pozdrží doručení m a posle jej dale, pokud dojde ke splneni nasledujicich podminek:

        - VTa(m)[i] = VTa(pj)[i] + 1
        - forall k (pk in ga & k ≠ i): VTa(m)[k] <= VTa(pj)[k]
            - kauzalita vzhledem ke skupina ga, kam byla zprava odeslana
        - forall b (pj in gb): VTb(m) <= VTb(pj)
            - kauzalita vzhledem k ostatnim skupinam prijemce (pj)
- doruceni zpravy
    - po doruceni si proces pj upravi VT
        - klasicky

Distribuovany total-order protokol

- totalni nebo sekvencni dorucovani
    - sekvencni dorucovani formou centralizovane komponenty - sekvencer
        - efektivnejsi
        - komunikacne jednodussi
        - nutnost vyberu koordinatora
        - serializace/usporadani dle prijmu sekvencera
        - moznost prioritniho dorucovani
- bez ohledu na typu dorucovaciho protokolu pouziteho pro zpravu je navic nutne, aby byly zpravy pri doruceni doruceny vsem clenum skupiny ve stejnem poradi
- stačí skalární hodiny (časové značky) - logicke hodiny
- při příjmu zprávy potvrzení odesilateli TS(Ri)
- odesilatel po příjmu všech potvrzení odešle finalizační zprávu s TSF = max(TS(Ri))
- po příjmu finalizační zprávy doručí příjemce zprávy podle TSF



Spolehlivé doručování

- naivni definice - vsem clenum skupiny bude dorucena zprava
- naivni implementace - poslu spolehlivym kanalem vsem clenum skupiny zpravu
    - problemy: vypadek prijemce/odesilatele
- transakce
    - doručení pouze po commitu
    - funguje, ale příliš striktní
    - nepříjemné důsledky:
        - odesílatel neobdrží potvrzení - abort celeho procesu
            - i když by stačilo něco méně drastického, např. redukce skupiny
    - havárie odesílatele po odeslání všech zpráv

Flooding algoritmus

- při příjmu každé doposud nepřijaté zprávy ji každý uzel přepošle **všem** ostatním
- nakonec všichni zbývající členové zprávu přijmou
- spolehlivý, neefektivní

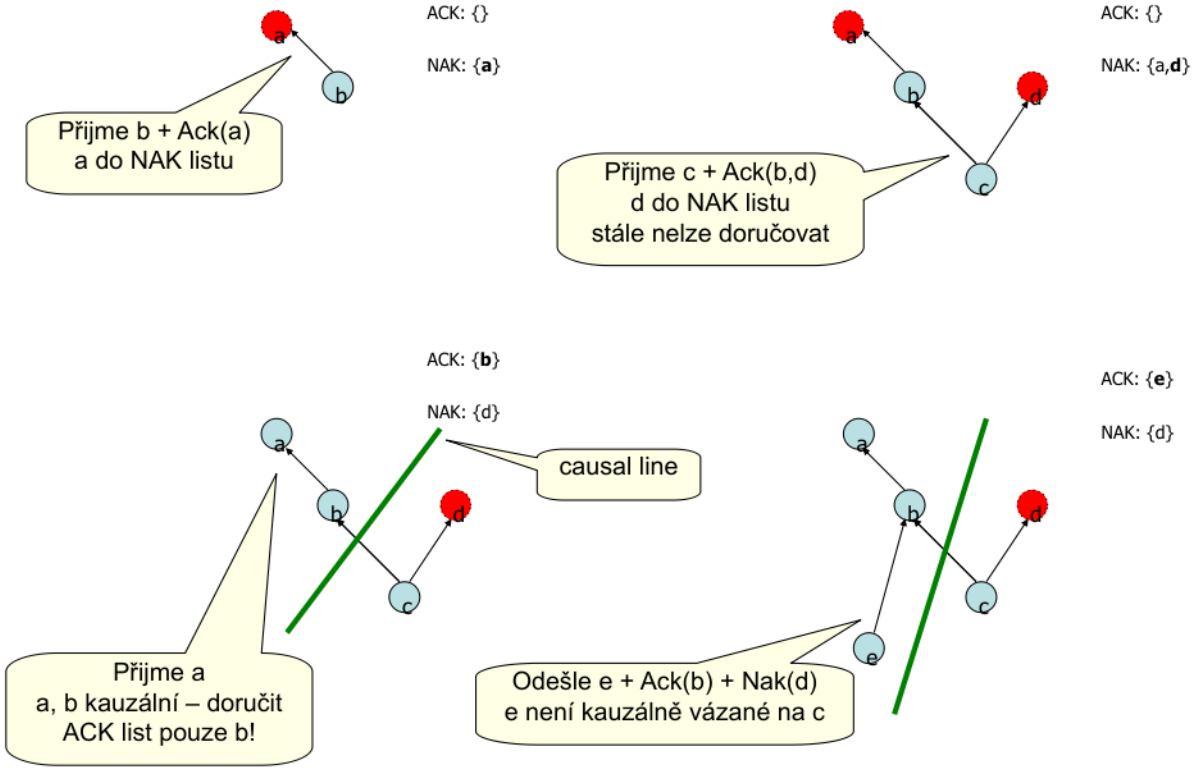
Algoritmus s potvrzováním

- jak to udelat:
    - pi odesílatel zprávy, pj in L příjemce zprávy, px havarovaný uzel
    - pi odešle zprávu forall p in L, zprávu si uchová až do obdržení Ack(p) forall p in L nebo do zjištění že px havaroval
    - pj po příjmu zprávy odešle Ack(pj) uzlu pi, zprávu si uchová až do zjištění že zprávu přijaly all p in L
    - jestliže pj zjistí, že pi havaroval, odešle zprávu forall p in L o kterých neví, že zprávu přijaly
- problem s tim, jak zjistit, ktere uzly zpravu prijaly a ktere ne

Trans algoritmus

- protokol pro spolehlive kauzalni dorucovani
- pro potvrzovani se pouziva graf zavislosti - DAG
- pouziva se invariant:
    - p rozesílá Ack(m) když přijal m a všechny kauzálně předcházející zprávy
        - transitive acknowledgements
    - není nutné potvrzovat zprávy kauzálně předcházející m
- stabilni zprava je ta, kterou prijali vsichni ze skupiny
- DAG G bude graf kauzality skupiny
- Gp - všechny zprávy které p přijal a ještě nejsou stabilní (ještě je nemají všichni)
- jestliže m potvrzuje zprávu m' pak (m, m') in G
    - hrany orientovany ve smeru potvrzeni (ne ve smeru kauzality)
    - vypocitavame si graf na zaklade doslych potvrzeni
- zdrojem informací pro negativní potvrzování je G (nepřijaté zprávy)
- jak pj zjistí které uzly zprávu přijaly

    - možnosti:
        - rozsilani Ack(p) nejen odesilateli, ale vsem - neefektivni, mnoho zbytecnych potvrzeni
        - vyuziti kauzality zprav, piggbacking potvrzení
            - D muze po prijmu a, b+Ack(a), c+Ack(b) odvodit, ze B prijal a, C prijal a i b
            - D muze po prijmu b+Ack(a), c+Ack(b) odvodit, ze B prijal a, C prijal a i b, A odeslal a -- vyzadame si zaslani
            - pri korektnim kauzalnim dorucovani jsou potvrzeni zprav tranzitivni (viz invariant)
- implementace:
    - ack_list, nak_list - seznam zprav pro doruceni a seznam neprijatych zprav
    - undelivered_list - seznam prijatych, ale jeste nedorucenych zprav
    - zpravu preposilame vcetne ack_listu, nak_listu

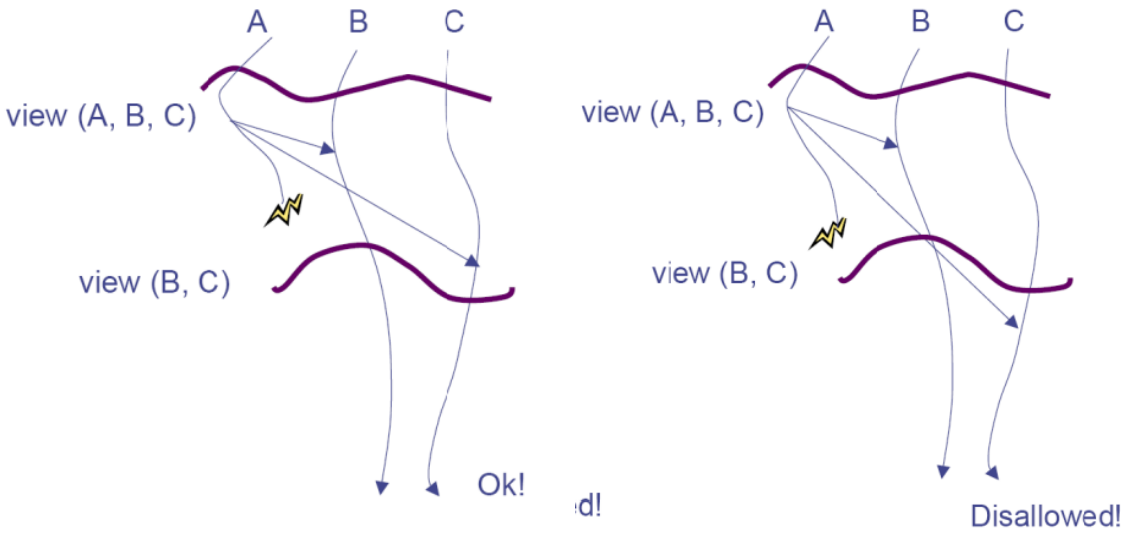


- pohled na protokol - posouvame stable line a causal line grafem
    - nove prichozi zprava vne causal line
    - pokud vsechny kauzalne predchazejici zpravy jsou uvnitr, pak i tuto zpravu lze dat dovnitr

- pozorovani:
    - je-li zpráva doručena spolehlivému procesu, pak někdy každý nehavarující proces zprávu dostane
    - zprávy jsou doručovány v kauzálním uspořádání
    - G reprezentuje graf závislostí v kauzálním uspořádání
    - všechny procesy vytvářejí stejný graf závislostí
        - pořadí přidávání uzlů však může být různé
    - jestliže procesor havaruje, paměťová náročnost je neomezená!
        - slabina - pro praktické použití musí být Trans protokol doplněn nějakým protokolem pro změnu členství ve skupinách

Virtuální synchronie

- group view / delivery list / ... - mnozina uzlu ve skupine
    - mnozina procesu nalezici skupine, kterou mel odesilatel pri posilani zpravy m
- znaceni: L, L_i, L^x, L_i^x - globalni pohled, lokalni pohled procesu i, verze pohledu x, verze lokalniho pohledu x pro proces i
- mame p,q in Lx, pohled se meni na Lx+1
- install Lpx < deliverp(m) < install Lpx+1 ⇒ install Lqx < deliverq(m) < install Lqx+1
    - pokud je zprava m odeslana skupine s pohledem Lx pred zmenou na Lx+1
        - vsechny uzly z Lx doruci m pred provedenim zmeny na Lx+1
        - nebo zadny uzel z Lx, ktery provede zmenu na Lx+1 zpravu m nedoruci
- podminka vzajemne konzistence
    - p in Lq ⇒ q in Lp
    - vsechny uzly ve skupine udrzuji stejny pohled L
    - instaluji pohledy ve stejnem poradi



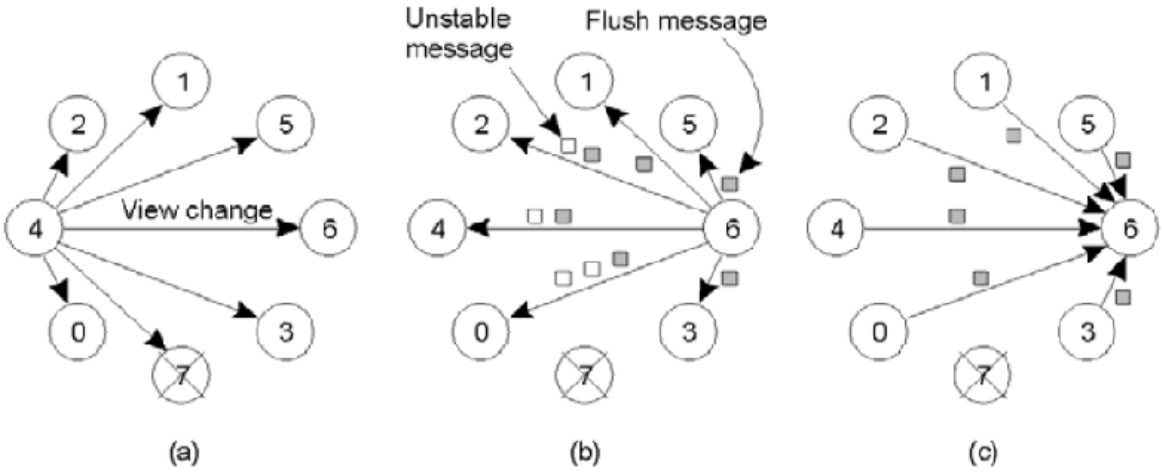
Transis algoritmus

- spolehlivy kauzalni multicast + clenstvi ve skupinach
- protokol je monotonni
    - nemožnost dosažení distribovaného konsensu (později)
    - paranoia - jedno podezření stačí k vyloučení
        - k vyloučení stačí větší zpoždění - nerozeznatelné od havárie
    - jednosměrnost - vyloučené procesy se již do algoritmu nevrací
        - lze se ale explicitně znovu připojit
- idea - konzistentni zmeny pohledů, dorucovani v ramci pohledů
    - při detekci havárie zpráva FAULT(q), při přijetí její přeposlání
    - kauzální hranice pohledu
        - vynutí doručení předcházejících zpráv
        - pozdržení zpráv kauzálně následujících
    - konkurentní detekce různých havárií
        - stanou-li se dvě havárie zároveň → společná hranice
        - doručení obou zpráv
        - doručení pozdržených zpráv
    - kauzální doručení zpráv havarovaného procesu vzhledem ke změně pohledu
        - zprávy odeslané kauzálně před zjištěním havárie se doručí v Lx
        - zprávy odeslané konkurentně se zjištěním havárie se zahodí
        - zprávy odeslané kauzálně po zjištění havárie se zahodí

Protokol ISIS

- spolehlive kauzalni dorucovani
- základem doručovacích protokolů je způsob přenosu informace o doručení
    - trans alg. ~ kauzalni potvrzeni
    - ISIS ~ maticove hodiny
- idea - každý proces zná VT[] všech procesů - udržuje matici MTpi[j][k]
    - co proces pi ví o doručení zpráv procesu pj od pk
    - tj proces si uklada:
        - vektor s odeslanymi zpravami
            - pri prijeti nejake dalsi zpravy od prijemce cas odeslani posledni zpravy od nej, kterou prijemce dostal
        - a zároveň vektory co mu došly od všech ostatních
- pj si při příjmu zprávy od pi aktualizuje MTpj[j] (co vi prijemce o sobe) a MTpj[i] (co vi prijemce o odesilateli)
    - MTpj[j][i]=MTm[i]
    - MTpj[i][*]=MTm[*]
- stabilní zprávy (přijaté všemi členy skupiny) - byly doručeny MT od všech členů skupiny
- srovnání s trans
    - trans potřebuje potvrdit max. M jiných zpráv - lze nahradit MT
    - trans lze povazovat za formu komprimace MT

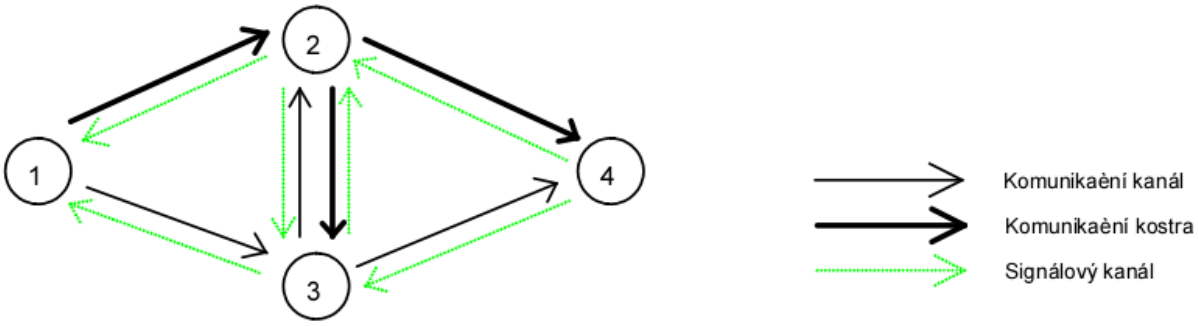
- optimistický algoritmus
    - relativně nízká režie při správném přenosu, vysoká režie při chybách
- cílem je, že všechny zprávy zaslané do pohledu L jsou doručeny všem žijícím procesům v L před instalací nového pohledu
- každý proces v L udržuje zprávu dokud není stabilní - detekce pomocí MT
- po příjmu zprávy o instalaci nového pohledu proces
    - přepošle všechny nestabilní zprávy
    - pošle flush message - potvrzení instalace
    - zatím pohled neinstaluje
- po příjmu flush od každého procesu může instalovat nový pohled
- zprávy od havarovaných procesů
    - každý proces udržuje seznam 'havarovaných' procesů aktuálního pohledu
    - s každou zprávou se rozesílá seznam, při příjmu se sjednotí s vlastním
    - zprávy od havarovaných procesů se zahazují
- reálné nasazení
    - New York Stock Exchange, Swiss Stock Exchange
    - French Air Traffic Control System
    - Reuters, Bloomberg



Ukončení distribuovaných systémů

- orientovaný graf uzlů se vstupními / výstupními kanály + signální kanály pro signalizaci ukončení
- iniciační proces začíná výpočet, vyšle podél svých výstupních hran zprávu
- proces se při příjmu zprávy dostane do stavu výpočet
    - může dále posílat zprávy podél výstupních hran
    - může přijímat zprávy ze vstupních hran
    - když proces skončí výpočet, dostane se do stavu ukončen, zprávy neposílá, může přijímat
    - při přijetí zprávy se proces zrestartuje do stavu výpočet

- algoritmus ukončení
    - všechny procesy jsou ve stavu ukončen, pak se v konečném čase toto všechny procesy dozví
    - stačí, když se to dozví iniciační uzel (může ostatním uzlům poslat zprávu)



Dijkstra-Scholten algoritmus

- pro strom
    - každý listový proces při přechodu do stavu ukončen pošle signál svému otci
    - proces dostane signály od všech svých synů, pošle signál svému otci
    - iniciační proces dostane všechny signály - konec výpočtu
- pro DAG
    - s každou hranou je asociován čítač - tzv. deficit
        - rozdíl mezi počtem zpráv došlých tímto datovým kanálem a počtem signálů poslaných signálním kanálem zpět
    - končící proces vyšle každým signálním kanálem tolik signálů, aby deficit = 0
- obecný graf
    - problém: neexistují listy, které by mohly rozhodnout o ukončení výpočtu
    - řešení: výpočet vytvoří dynamicky kostru grafu
        - otec = uzel, od kterého přišla první zpráva
    - algoritmus ukončení pro jeden proces:
        - 1. poslat signál podél všech vstupních hran kromě hrany k otci
        - 2. čekat na signál od všech výstupních hran
        - 3. poslat signál otci
    - iniciační proces dostane všechny signály - konec výpočtu

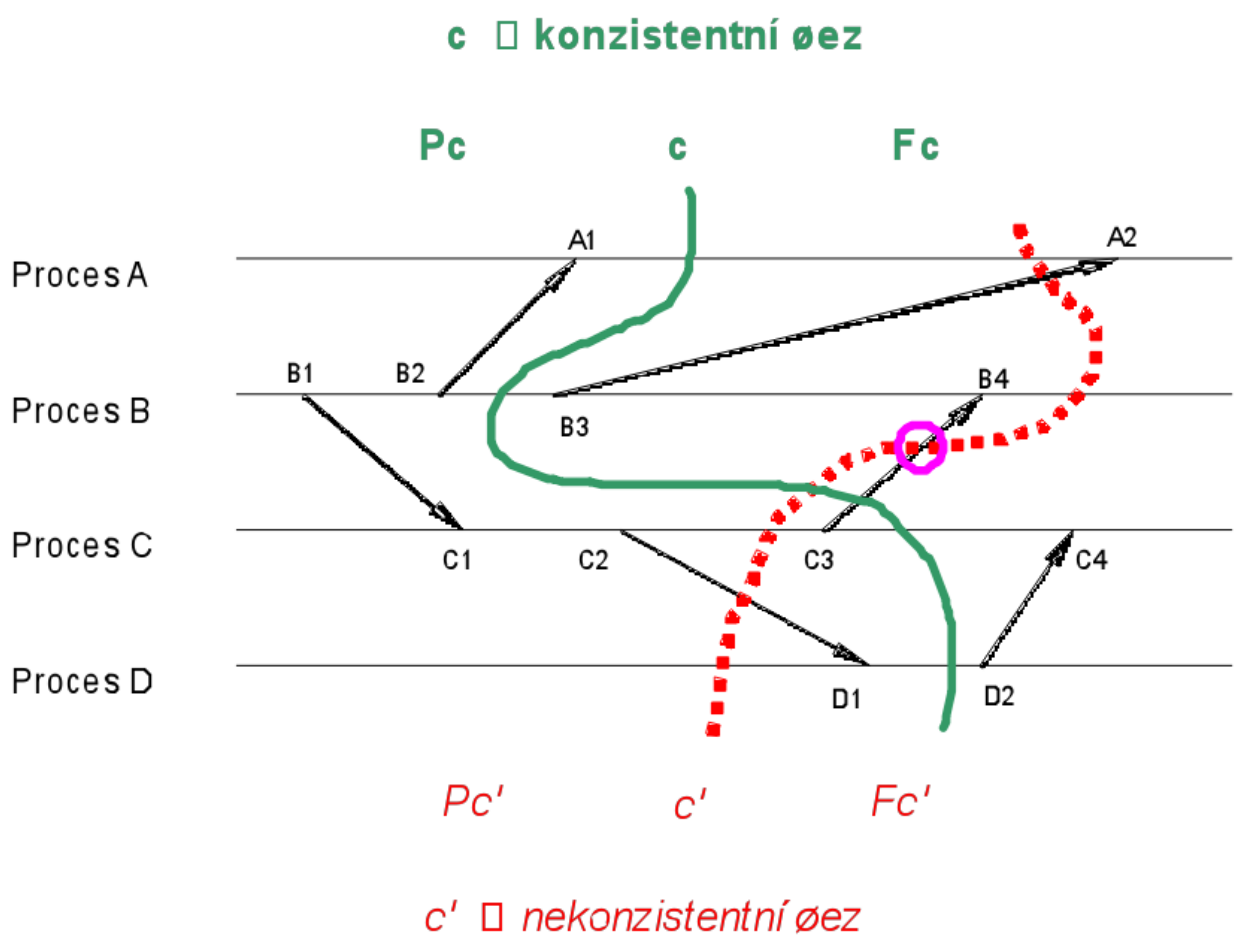
Značkový algoritmus

- značka je speciální zpráva odlišitelná od běžných zpráv
- iniciační uzel vyšle žádost o ukončení (značku, že je kanál prázdný) všem výstupním hranám
- uzel přijme první značku:
    - připraven - propagace značky výstupním hranám
    - nepřipraven - vysílá negativní signál (zamítnutí)

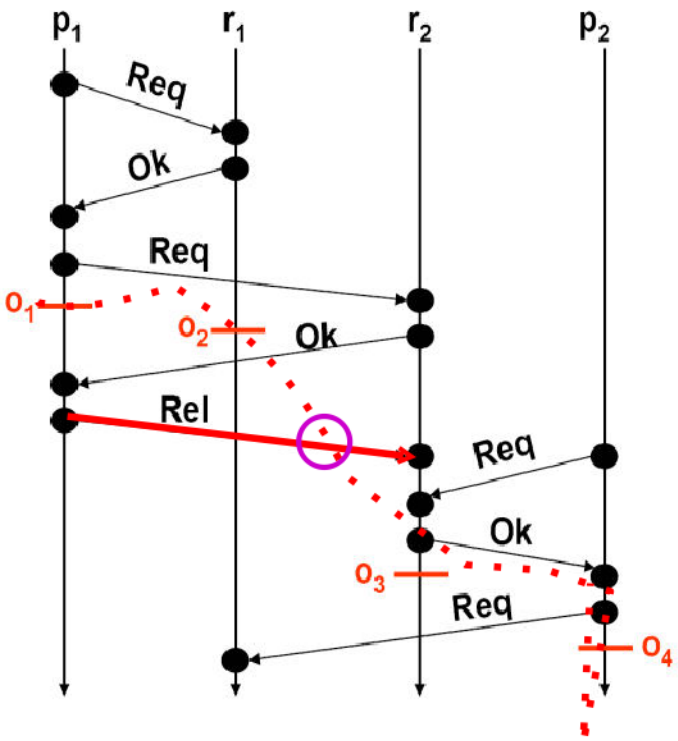
- příjem další značky: signál / zamítnutí
- příjem zamítnutí: zamítnutí první žádosti
- příjem všech potvrzení: signál první žádosti
- speciální případ obecnějšího algoritmu - detekce globálního stavu

Konzistentní stav

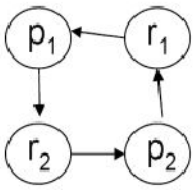
- máme množinu událostí v systému E
- řez c je disjunktní rozdělení E na Pc a Fc (past/future): Pc u Fc = E & Pc n Fc = 0
- konzistentní řez c: (a → b & a in Fc) ⇒ b in Fc
    - minulost nelze ovlivnit udalostmi z budoucnosti
- stav dist. procesu je mnozina udalosti, ktere se v procesu udaly
- konzistentni stav S = Pc, kde c je konzistentni rez
- mame-li S konzistentni stav a hranu e ⇒ S' = S u e je konzistentni stav (je dosazitelny z S pomoci e)
- posloupnost udalosti s = (e1...en) se nazyva rozvrh, pokud je Sn dosazitelne z S pres udalosti (hrany) e1...en



Detekce falešného deadlocku



Inference from
o_1: p_1 waits for r_2
o_2: r_1 waits for p_1
o_3: r_2 waits for p_2
o_4: p_2 waits for r_1



From $O=\{o_1,o_2,o_3,o_4\}$ the deadlock detector concludes there is a deadlock!

Detekce globálního stavu

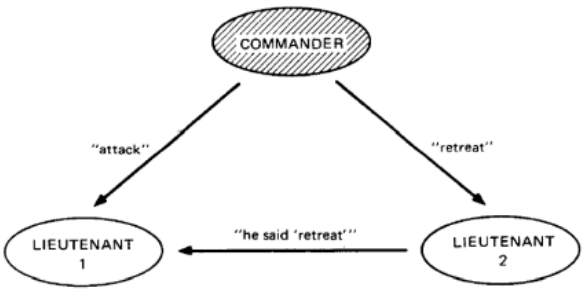
- udrzujeme globalni stav DS
  - zahrnuje lokalni stav vsech proc. + zpravy, ktere jsou aktualne v obehu (nedorucene)
- uzitecne napr. pro detekci ukonceni behu/deadlocku, garbage collection, obecne detekce glob. vlastnosti
- jak? - udrzovat distribuovany snapshot
  - stav, ve kterem system mohl byt
  - odrazi konzistentni glob. stav
  - pokud jsme zaregistrovali, ze P obdrzel zpravu od Q, meli bychom take zaznamenat, ze Q zpravu poslal
    - je ale povoleno, ze Q zpravu poslal a obdrzeni od P jsme jeste nezaregistrovali
    - konzistentni/nekonzistentni rezy
      - posledni udalosti, ktere byly u procesů zaznamenány
  - predpoklad: DS je kolekce procesu navzajem propojenych
    - takze napr. proces musi nejprve iniciovat TCP spojeni, pokud chce komunikovat
  - jakykoliv proces muze iniciovat algoritmus - necht je to P
  - P zacne zaznamenanim sveho lok. stavu

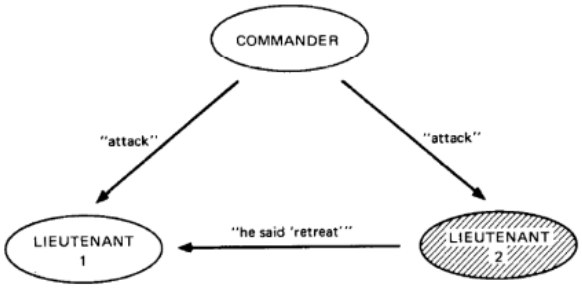
    - nasledne posila znacku kazdym svym vystupnim kanalem, ktera znaci, ze prijemce se ma podilet na zaznamenavani glob. stavu
    - kdyz proces Q obdrzi takovou znacku, tak pokracuje na zaklade toho, zda ma zaznamenany svuj lok. stav ci ne
        - nema - posle znacku do vystupnich kanalu a zaznamena svuj lok. stav
        - ma - mezi prvni znackou co dostal a tou co ted dosla si zaznamenaval prichozi zpravy a ted kdyz mu dosla druha znacka, tak si svoji frontu spolecne se svojim lokalnim stavem zaznamenal, jako akt. stav kanalu
    - proces skoncil svou ulohu, pokud obdrzel znacku vsemi prichozimi kanaly a kazdou zpracoval
    - v tuto chvili muze cely snapshot sjednotit a poslat iniciatorovi jako odpoved

Distribuovaný konsensus

- *organizace replikovanych procesu do skupin pomaha zlepsovat odolnost proti chybam*
- klasifikace problemu
    - byzantsky konsensus (1 → 1)
        - iniciator vysila jedinecnou hodnotu, vsechny nechybove procesy se na ni musi shodnout
    - konsensus (n → 1)
        - iniciujicich uzlu je nekolik a maji svoje inicialni hodnoty, vsechny nechybove procesy se musi shodnout na jedne spolecne hodnote
    - interaktivni konzistence (n → n)
        - iniciujicich uzlu je nekolik a maji svoje inicialni hodnoty, vsechny nechybove procesy se musi shodnout na mnozine/vektoru spolecnych hodnot
- problem dvou armad
    - pocetnejsi armada je rozdelena
    - úspěch pouze při synchronizovaném útoku, jinak drtivá porážka
    - obě části musí mít jistotu, že druhá část začne útok také
    - komunikace pouze nespolehlivým kurýrem - možnost zajetí
    - striktní řešení neexistuje
        - A1 → A2: attack -- A1 nevi jestli A2 dostal zpravu
        - A2 → A1: A2 nevi, jestli A1 dostal potvrzeni
        - A1 → A2: A1 nevi, jestli A2 dostal potvrzeni
        - …
    - praktická řešení
        - agresivní
            - první generál vyšle větší množství zpráv oznamující čas útoku, zaútočí tak jako tak
            - předpokládá se, že při vysokém počtu poslaných zpráv alespoň jedna projde
            - druhý generál zde ani nemusí odpovídat

        - pravdepodobnostni
            - první generál vyšle větší množství (N0) zpráv
            - kromě času útoku přidá i to, kolik jich poslal
            - druhý generál odpoví na každou, kterou obdržel (N1)
            - počet odpovědí vrácených prvnímu generálovi je N2
            - oba generálové znají přibližnou míru úspěšnosti, že posel zprávu pronese
- problem byzantskych generalu
    - předpoklady:
        - uzly mohou libovolně havarovat
        - havarovaný uzel se může chovat zákeřně!
        - spolehlivá komunikace (časově neohraničená)
    - úkol:
        - někteří generálové jsou zrádci
        - všichni loajální generálové se musejí rozhodnout shodně
        - každý generál se rozhoduje na základě informací obdržených od ostatních generálů
    - budeme uvažovat 1 generál, ostatní důstojníci (búno)
        - jak generál tak důstojníci mohou být zrádci
        - generál vydá rozkaz, důstojnící ho předají ostatním
        - rozkaz bude vydán na základě většiny
    - cíl:
        - C1: všichni loajální důstojníci vydají stejný rozkaz
        - C2: je-li generál loajální, pak každý loajální důstojník vydá rozkaz generála
    - 3 uzly, 1 zradce
        - reseni neexistuje
        - obecne je reseni jen pro: n > 3m uzlu, kde m je # zradcu





    - 4 uzly, 1 zradce
        - zradce general
            - vyda 3 stejne rozkazy - nelze, to by nebyl zradce
            - vyda 2 stejne rozkazy
                - dustojnici si vzajemne preposlou vetsinovy rozkaz
                    - dosahneme C2

                - vyda 3 ruzne rozkazy
                    - dustojnici se shodnou na tom, ze je zradce
                        - dosahneme C1
            - zradce dustojnik
                - nejhorsi pripad: preda falesny rozkaz vsem ostatnim
                    - loajalni dustojnici dostanou vetsinu spravnych rozkazu (C2)

Distribuovaná sdílená paměť

- paralelní výpočty
    - multiprocesory
        - malý počet procesorů
        - příliš komplikované a drahé
    - multicomputery
        - snadno dostupné
        - programování a synchronizace není prakticky (inženýrsky) dobře zvládnutá
        - RPC – problémy
        - pokus o řešení – distribuovaná sdílená paměť (DSM)
            - 1986 Li & Hudak – DSM - množina uzlů sdílející adresový prostor

Konzistenční modely DSM bez SP

- specifikace co implementace musí splňovat vzhledem k operacím čtení a zápis
- klasifikace od nejvíc pedantského po nejvolnější
- striktní konzistence
    - jakékoliv čtení z paměti z adresy x vrátí hodnotu uloženou při posledním zápisu na adresu x
    - zajišťuje absolutní časové uspořádání
    - nejsilnější, na jednoprocesorových systémech je tradičně zajištěna
    - všechny zápisy okamžitě všude viditelné
    - podmínka: musí existovat přesný globální čas
        - ideální pro programování, v DS nedosažitelné
- sekvenční konzistence
    - výsledek výpočtu je stejný, jako kdyby všechny operace všech procesorů byly vykonávány v nějakém sekvenčním uspořádání a operace každého procesoru jsou vykonávány v pořadí specifikovaném programem
    - snadno implementovatelná
    - příjemná pro programování
    - povoleno libovolné prokládání instrukcí na různých procesorech
    - všechny procesy vidí stejné pořadí změn paměti
    - změny nejsou propagovány okamžitě, není zaručena velikost zpoždění
    - výkonnost není příliš velká
        - platí že čas čtení a zápisu dohromady musí trvat alespoň tak jako přenos jednoho paketu → pomalé

- kauzální konzistence
    - kauzálně vázané zápisy musí být viděny všemi procesy ve stejném pořadí, konkurenční zápisy mohou být viděny v různém pořadí
    - rozlišuje události, které jsou potenciálně závislé a které ne
    - kauzálně závislé zápisy
        - pokud W(x)a a (druhý proces) R(x) W(y)b, pak W(y)b je kauzálně závislý na W(x)a
    - analogie k zasílání zpráv: zápis ~ odeslání, čtení ~ přijetí
    - složitější implementace
        - vyžaduje udržování grafu závislostí zápisů a čtení
- PRAM konzistence (FIFO konzistence)
    - pipelined RAM
        - zapisy jednoho procesu jsou pipelinovane, nemusi se cekat vzdy na na dokonceni kazdeho nez se zacne novy
    - zápisy prováděné jedním procesem jsou viděny ostatními procesy v tom pořadí, ve kterém byly prováděny, zápisy různých procesů mohou být viděny různými procesy různě
    - srovnání se sekvenční konzistencí:
        - v PRAM neexistuje jednotný pohled na rozvrh
    - snadná na implementaci
        - nezáleží na pořadí, v němž různé procesy vidí přístupy k paměti
        - je nutné dodržet pořadí zápisů z jednoho zdroje
- slow memory
    - zápisy jedním procesem do jednoho místa musí být viděny ve stejném pořadí
    - nejslabší model - lokální zápis, pomalá nesynchronizovaná propagace
    - neposkytuje žádnou synchronizaci

Konzistenční modely DSM s SP

- doposud uvedené konzistenční modely velmi restriktivní
    - vyžadují propagaci všech zápisů všem procesům
    - málo efektivní
    - ne všechny aplikace vyžadují sledování všech zápisů, natož pak jejich pořadí
    - typická situace:
        - proces v kritické sekci ve smyčce čte a zapisuje data
        - ostatní procesy nemusí jednotlivé zápisy vidět, není nutné, aby byly propagovány
- paměť ale neví, že proces je v kritické sekci, musí propagovat všechny zápisy
    - řešení: nechat proces ukončit kritickou sekci a poté rozeslat změny ostatním
        - používá se synchronizační proměnná
- slabá konzistence
    - přístup k synchronizačním proměnným je sekvenčně konzistentní
        - všechny procesy vidí všechny přístupy k SP ve stejném pořadí

    - přístup k SP není povolen, dokud neskončí všechny předchozí zápisy
        - před přístupem k SP budou dokončeny všechny předchozí zápisy
    - přístup k datům není povolen, dokud nebyly dokončeny všechny předchozí přístupy k SP
        - při přístupu k obyčejným proměnným jsou dokončeny všechny předchozí přístupy k SP
    - provedením synchronizace před čtením se zajistí aktuální verze dat
    - odpadá nutnost propagace všech zápisů, ale paměť při přístupu k SP nerozezná vstup/výstup z CS → musí se vždy vykonat akce pro oba případy
- výstupní konzistence
    - před přístupem ke sdílené proměnné musí být úspěšně ukončeny předchozí Acq() procesu.
    - před provedením Rel() musí být ukončeny všechny předchozí zápisy i čtení prováděné procesem.
    - Acq() a Rel() musí být PRAM konzistentní.
    - po Acq() jsou všechny lokální kopie aktuální
    - po Rel() jsou propagovány změny ostatním procesům
    - při správném párování Acq() a Rel() je výsledek jakéhokoliv výpočtu ekvivalentní sekvenčně konzistentní paměti
    - implementace:
        - eager release consistency (horlivá)
            - po Rel() se propaguje všem procesům
            - optimalizace přístupové doby
        - lazy release consistency (líná)
            - po Rel() se nic nepropaguje, až po Acq() jiného procesu
            - optimalizace síťového přenosu
- vstupní konzistence
    - Acq k SP není povolen, dokud nebyly provedeny všechny aktualizace chráněných sdílených dat procesu
    - exkluzivní přístup procesu k SP (= zápis) je povolen pouze v případě, že žádný jiný proces nepřistupuje k SP, a to ani neexkluzivně (= čtení)
    - po exkluzivním přístupu k SP si příští neexkluzivní přístup libovolného procesu k SP musí vyžádat aktuální kopii dat od vlastníka SP
    - sdílená data jsou vázána na SP, při přístupu se synchronizují pouze tato data
    - přístup k datům a SP může být exkluzivní (RW) nebo neexkluzivní (RO)
    - každá SP má vlastníka - proces, který k ní naposledy přistupoval
    - vlastník může opakovaně vstupovat a vystupovat z k.sekce bez nutnosti komunikace
    - proces, který není vlastníkem, musí požádat o vlastnictví

Distribuované stránkování

- obdoba virtuální paměti - přístup k nenamapované stránce → přerušení, obsluha, načtení
- adresovy prostor rozdelen na casti (chunks - pages), ktere jsou rozprostreny mezi vsechny procesory v systemu
- kdyz procesor referencuje adresu, ktera neni lokalni, nastane vyjimka, DSM software nafetchuje prislusnou cast a pokracuje se dal
- problémy
    - replikace → konzistence
        - jak udržovat data konzistentní
            - namapování read-only, při zápisu synchronizační akce
            - replikovat vsechny casti
                - pokud dochazi pouze ke cteni je to OK a muzou se replikovat i RW kusy
                - pokud je ale replikovana kopie nahle zmenena, dochazi k nekonzistenci
                    - resi se aplikaci nektereho konzistencniho protokolu
            - invalidace vs. aktualizace
    - nalezení stránky
        - jak nalézt distribuovaná data, vlastníka stránky
            - centralizovaný manager
            - replikovaný manager (indexace spodními n bity nebo hash)
            - broadcast
    - správa kopií
        - co dělat s kopiemi při čtení/zápisu
            - jak je najit, kdyz musi byt invalidovány
            - broadcast page number a rict vsem procesorum, ktere ji maji, aby ji invalidovali
                - musi byt spolehlive
            - copyset
                - vlastník stránky udržuje množinu lokací kopií
                    - pokud musi byt stranka invalidovana, posila zpravu procesorum, ktere ji maji a ceka na ACK
    - uvolňování stránek
        - kterou stránku uvolnit
            - nevlastněná read-only kopie
            - vlastněná replikovaná kopie - přenos vlastnictví
            - lokální heuristika - LRU / …
    - falešné sdílení
        - nezávislá data na jedné stránce

Sekvenčně-konzistentní distribuované stránkování

- využívá protokolu sekvenční konzistence pro zachování konzistentních dat
- stránky mají vlastníka, který na ně může zapisovat, ostatní mají kopie ke čtení

Kauzálně-konzistentní distribuované stránkování

- základní idea – graf závislostí
- vektorové hodiny:
    - VTS ~ jednotka granularity (stránky)
        - na kterých stránkách závisí obsah
    - VTP ~ procesy
        - z kterých stránek znám data
- VT[i] ≈ stránka i
- výpadek stránky – přenos dat, VTP = max(VTS , VTP)
- zápis do stránky – VTS = inc(VTP)
    - jak OS pozná zápis? - změnou mapování
- aktualizace VTP – zneplatnění stránek i: VTSi [i] < VTP [i]
- porovnání kauzality - obdobné jako dříve - srovnání vektorů, pokud jsou porovnatelné
- problémy:
    - velká prostorová režie (1 MB = 256 stránek = 512 B / stránku)
    - propagace konkurentních zápisů
        - zámky, bariéry, timeouty

Distribuované sdílené proměnné

- implementace na úrovni knihoven
    - potenciálně replikovaná distribuovaná databáze
    - typicky konzistenční model se synchronizačními proměnnými
- výhody
    - potenciálně lepší výkonnost
    - eliminace false sharing
- nevýhody
    - nepodporováno přímo operačním systémem
    - nutnost implementace pro různé jazyky
    - nutnost rekompilace
- příklad implementace: Munin 1990 Benett & spol.
    - pouziti ordinary / shared / synchronization promennych
        - shared promenne: read-only, migratory, write-shared, conventional
            - read-only
                - při výpadku je v adresáři nalezen vlastník, žádost o read-only kopii dat
            - migratory
                - acquire/release protokol implementující eager release consistency
                - při opuštění kritické sekce se propagují změny
                - data chráněná SP migrují na uzel v kritické sekci

        - write-shared
            - stránky iniciálně r/o, při zápisu do lokální kopie s původním obsahem jsou r/w, označí se dirty
            - po release se porovná stránka s původní, změny se propagují
            - propagace na ne-dirty stránku akceptace, jinak word-po-wordu porovnání
            - sjednocení dat / konflikt - runtime-error
        - conventional
            - jako distribuované stránkování - single writer/many readers
            - sekvenční konzistence
- distribuované objekty
    - díky zapouzdření flexibilnější - komunikace a synchronizace v metodách
    - distribuovaná data potomkem základní distribuované třídy
    - Class Definition Language - automatické generování hlaviček a kódu
    - základní "distribuovaná" třída, dědění vlastností, CORBA, Java RMI, …

Identifikace a správa prostředků

- který objekt má být použit (identifikace)
- kde je tento objekt umístěn (adresa)
- jak se k němu můžeme dostat (cesta)
- trvanlivost identifikaci
    - dynamicka jmena - docasna, transient
        - typicky vazana na zivotnost procesu
        - deskriptory otevrenych objektu, porty, adresy DSM
    - staticka jmena - trvala, persistent
        - nezavisla na procesech
        - jmena souboru, kapability
- prostor jmen
    - separatni prostory - file system, registry, URL, id procesu, …
    - jednotny - distribuovany name server
    - lokacni a replikacni transparentnost
- nestrukturovaná jména (UUID) versus strukturovaná (ms.mff.cuni.cz)
- systemova jmena
    - interní identifikace objektů, optimalizace pro strojové zpracování
    - typicky binární řetězce pevné délky i struktury
    - chráněné objekty jádra - porty
    - access control matrix - user x resource → effective right
        - tak jednoduche to neni, takova matice by mohla byt obrovska
        - capabilities - uživatel drží oprávnění k prostředkům
        - access control list - prostředek (objekt) má seznam uprávněných uživatelů

Kapability

- datová struktura umožňující jednoznačnou identifikaci objektu
- obsahuje navíc přístupová práva pro držitele kapability
- ochrana objektů, šifrování, redundance
- k jednomu objektu typicky několik různých kapabilit
    - vlastník, písař, čtenář, ... další druhy služeb
- uživatelským procesům znemožněno vlastní generování kapabilit i změny práv
- výhody
    - snadný test oprávěnosti přístupu
    - flexibilita - každý správce prostředků může nadefinovat vlastní druhy práv
- nevýhody
    - problematická kontrola propagace
        - možnost neautorizovaného přístupu
        - copy bit, čítač - chráněný způsob přenosu
    - review - seznam oprávněných uživatelů
    - revocation - odejmutí práva
        - zrušení objektu a vytvoření nového, notifikace ostatních uživatelů
    - garbage collection
- varianty:
    - s podpisem
        - hlavní část - identifikace objektu a přístupových práv
        - platná kapabilita je rozšířena o podpis, který je vypočítán z obsahu hlavní části
        - celá kapabilita zašifrována tajným klíčem
            - ochrana proti odvození podpisové funkce uživatelskými procesy
        - řídkost: ze všech binárních čísel velikosti kapability jsou platné jen ty, jejichž podpis odpovídá zbytku kapability
    - s redundantní kontrolou
        - port serveru a identifikátor objektu jsou volně přístupné, ale část s přístupovými právy zašifrována
            - přístupová práva a náhodně vygenerované binární číslo pevné délky - zakódováno
                - číslo má u sebe i server, který kapabilitu vytvořil → verifikace
        - ochrana serveru
            - port, dostatečně velké náhodně generované číslo
        - ochrana přístupových práv
            - zašifrování pole s přístupovými právy spolu s redundantní kontrolou
        - uživatelskému procesu znemožněna změna přístupových práv
            - proces nemůže svá práva změnit, nezná dešifrovací funkci, ani ji nemůže odvodit
            - náhodně generovaná redundance
        - služba serveru - vygenerování kapability s menšími právy

- klient kapabilitu pošle serveru, pomocí čísla check, které si uložil při generování kapability a masky práv, kterou mu klient předloží vygeneruje nový check a vytvoří restringovanou kapabilitu

Distribuovaná správa jmen

- name servers, directory servers, lookup servers
- adresare jsou mnozina polozek <jmeno,hodnota>, kde hodnota muze byt:
  - primitivní (čísl, řetězce, binární data)
  - perzistentní reference (trvalé odkazy na objekty, kapability)
  - tranzientní reference (na živé objekty, porty, kanály)
  - odkazy na jiné adresáře
  - operace jako SET, LOOKUP(jmeno) - změna kontextu, LOOKUP(složené jméno - hierarchie v URL napr.)
  - vyhledávání typicky přes server, ten publikuje jména a vrací reference na objekty (to může být třeba nějaká kapabilita nebo něco jako handle do FS)
- klientsky proces ma nekolik zakladnich “adresaru”
  - file system, objekty, sluzby a servery, registry
- server spravuje objekty - jejich identifikace a operace nad nimi
  - kazdy objekt ma lokalni id spravovane serverem
    - na nem definovany operace / sluzby
  - server publikuje ridici port a porty pro otevrene objekty
  - proces:
    - server publikuje kapability na name server
    - zaregistruje sve sluzby u registration serveru a vytvori kanal
    - klient pozada NS o kapabilitu
    - klient pozada RS o open(cap)
    - pozadavek se preposle serveru
    - mezi klientem a serverem se vytvori komunikacni kanal
  - poskytovane sluzby
    - porty ~ objekty, sluzby ~ metody
    - vytvoreni objektu, transient reference - service_port->create_obj(...)
    - operace nad objektem - obj_port->op()
    - freeze - vytvoreni kapability (perzistentniho odkazu)
      - cap = obj_port->freeze()
    - zruseni transient reference (ne objektu) - obj_port->drop()
    - ulozeni kapability - ns->add(“path/to/cap.obj”, cap)
    - vytvoreni restring. kapability z dosavadni kap.
      - cap = service_port->cap_restrict(cap,0101)

Distribuovaná správa prostředků

- centralizované řešení - resource manager
    - pokusy o distribuovanou správu
        - prakticky nepoužitelné - distribuované vyloučení procesů
- migrace - identifikace, komunikace
- replikované servery – aktualizační protokol
- problém deadlocku
    - časté řešení - pštrosí algoritmus
    - detekce - větší problém než u centralizovaných systémů
    - WFG – Wait-For-Graph
        - orientovaný graf (procesů, transakcí)
        - P1 → P2 - proces P1 je blokován procesem P2
        - orientovaná kružnice – uváznutí

Algoritmy detekce deadlocků

- korektnost
    - každý existující deadlock je v konečném čase detekován
    - detekovaný deadlock musí existovat
        - nutné dávat pozor na vnořené deadlocky - phantom deadlock
- modely - single, and, or m-out-of-n, and-or
    - and - proces muze naraz pozadat o vice prostredku, je zasekly, dokud nema vsechny
    - or - jak vyse, ale staci mu jeden, aby se rozjel
    - single unit - muze zadat pouze o jeden prostredek, ve WFG vystupni hrany stupne 1, cyklus znaci deadlock
    - and-or - pozadavky o prostredky reprezentovany predikatem, kde jsou atomy prostredky, kdyz je formule splnitelna, proces se rozjede
    - m-out-of-n - muze zadat o n, staci kdyz bude mit m

Centralizovaný algoritmus na DD

- jeden uzel vybran jako ten, co detekuje, vsechny zavislosti, ktere v systemu vyvstanou, se posilaji jemu
- na centralnim uzlu se analyzuje a spravuje WFG
- prenasime informace bud po kazde zmene, po danem intervalu nebo na expl. vyzadani
- pro prevenci falesneho uvaznuti kvuli zpozdeni zprav se pouzivaji logicke hodiny + kauzalni dorucovani
- rozsireni na hierarchicky algoritmus: koordinatori a podrizeni - koord. resi deadlocky podrizenych, uzel resi deadlocky lokalne

Path-pushing algoritmus na DD

- zakladni idea je, ze kazdy uzel periodicky shromazduje informace o lokalnich zavislostech a stavi si svuj lokalni WFG
- sousednim uzlum posilaji externi zadosti, z kusu WFG, ktere jim nepatri
- tyto vysledky pak zahrnou do sve casti
- toto se opakuje dokud nejaky uzel nema dostatecne info o tom prohlasit zda doslo k deadlocku ci nikoliv

Edge-chasing algoritmus na DD

- zasilani spec. zprav (probes) podel WFG
    - proces odesle zpravu vsem, na ktere ceka
    - pokud se zprava vrati, je to deadlock (existuje orient. kruznice)
    - mezitim se to vsak mohlo odblokovat
        - resenim je aging - postupne zvysovani priority tech, kteri jiz dlouho cekaji

Diffusing computation na DD

- těm na které čekám se posílají pingy, oni je vrací pokud jsou taky zablokováni, pokud dostanu všechny své pingy zpět, mám deadlock
- vhodne u slozitejsich modelu (OR, m-out-of-n)

Detekce globálního stavu pro DD

- jestlize WFG' nasleduje WFG a proces je ve WFG v deadlocku, bude i ve WFG' (existuje-li deadlock, je i v konzistentnim rezu)
- pri prijmu znacky - tj. v okamziku rezu - uzel zaznamena lokalni WFG
- dochazi k lokalni kontrakci WFG, externi zavislosti se posilaji iniciatorovi

Distribuované procesy

- správa procesů v distribuovaných systémech
- sdílet výpočetní sílu systému
- rozdělovat zátěž na jednotlivé procesory
- provádět operace na vzdálených procesech
- synchronizovat procesy a vést evidenci stavu

Vzdálené spouštění procesů

- nalezení volného uzlu
    - alokační algoritmy
- znalost vlastní zátěže
    - procento využití procesoru
    - periodické měření - vyhlazení (nějaký klouzavý průměr)
- spuštění vzdáleného procesu
    - transparentnost
    - přenesení kódu a dat – nezajímavé
        - heterogenní prostředí
    - vytvoření prostředí odpovídající domovskému uzlu
        - kontexty (fs, naming, environment)
        - systémová volání – vzdálená / přesměrování
- problém - hostitelský uzel přestane být volný (uživatel se vrátil z oběda, potřeba restartu, ...)
    - doběhnutí / zabití / čas na uložení / uzavření / migrace

Klasifikace alokačních algoritmů

- zda jsou předem známy údaje o procesech, podle kterých se algoritmus rozhoduje
    - deterministické / heuristické
- do jaké míry se provádí optimalizace
    - optimální / suboptimální
- co se snaží optimalizovat
    - využití CPU / čas odpovědi / přidělování prostředků / komunikační složitost, ...
- zda umožňují přemístění již rozběhnutého procesu
    - migrační / nemigrační
- rozdíly mezi HW uzlů, speciální požadavky procesů na hw vlastnosti
    - homogenní / heterogenní
- podle charakteru algoritmu
    - centralizované / distribuované
- podle jakých informací se provádí rozhodnutí o vzdáleném spuštění procesu
    - lokální / globální
- kdo iniciuje vzdálené spuštění procesu
    - odesilatel / příjemce / symetrický
- *podtržené - co se používá v praxi*

Alokace procesorů

- hierarchický algoritmus
    - manažeři skupin, při neúspěchu žádost vyšším místům
- distribuovaný heuristický algoritmus
    - k náhodných výběrů cíle
    - server / receiver initiated
- deterministický grafový
    - minimalizace komunikace
    - nutnost znalosti komunikační složitosti
    - optimální deterministický algoritmus – tok v sítích
- bidding (obchodní, nabídkový) algoritmus
    - procesy kupují výpočetní sílu, procesory ji nabízejí
- up-down algoritmus
    - optimalizace na stejnoměrné sdílení výkonu
    - koordinátor má tabulku se záznamem pro každý uzel obsahující “trestné body”
    - při každé významné akci (vytvoření procesu, ukončení procesu, tik hodin):
        - zpráva koordinátoru, který provede změny:
            - každý proces běžící na jiném uzlu - plus trestné body
            - každý neuspokojený požadavek - mínus trestné body
            - jestliže nic z tohoto - směrem k nule
    - v okamžiku uvolnění uzlu se vezme ten proces z fronty (neuspokojených požadavků), jehož vysílající uzel má nejméně trestných bodů
    - vlastnosti:
        - homogenní, nemigrační, spravedlivý, heuristický, centralizovaný, suboptimální, globální, symetrický

Migrace procesů

- jde nám o korektní a transparentní přenesení procesu během výpočtu
- motivace
    - vyvažování zátěže; optimalizace - I/O, komunikační; přemístění; shutdown
- korektnost
    - ostatní procesy nejsou migrací podstatně ovlivněny
    - po ukončení stav odpovídá stavu bez migrace
- transparentnost
    - proces o migraci neví a nemusí spolupracovat
    - zůstanou zachovány vazby na komunikující procesy
    - není narušena komunikace
        - ... v konečném důsledku

- problémy k řešení
    - přenesení rozpracovaného stavu …
    - přenesení adresového prostoru …
    - reinicializace vazeb na ostatní procesy
    - komunikace s ostatními procesy
    - reziduální dependence - žádná, domácí, průběžná, dočasná
    - vícenásobná migrace - viskozita

Implementace migrace - přenos procesu

- vyjmutí ze stavu, zmražení, speciální stav
- oznámení příjemci o migraci, alokace procesu
- přenos stavu - registry, zásobník, stav procesu
- přenos kódu / adresového prostoru
- přesměrování / doručení zpráv
- dealokace procesu, vyčištění
- vazby na nové jádro, nastartování procesu
    - přesunutí části stavu spolu s procesem (obsah VM)
    - forwardování (některých) požadavků (komunikace s konzolí)
    - použití odpovídajícího prostředku na cílové stanici (fyzická paměť)
- dokončení přenosu vazeb, dočištění
    - dočasná reziduální dependence, přesměrování zpráv
- co s virtuální pamětí
    - přenesení celé VM při migraci
        - výhody - eliminace reziduálních dependencí
        - nevýhody - prodloužení doby zmražení procesu, mnohdy zbytečné přesouvat celý obsah virtuálního adresového prostoru
    - pre-copying
        - výhoda - proces je zmražen pouze po dobu přesunu malého množství dat
        - nevýhoda - některé části se kopírují vícekrát - prodloužení celkové doby migrace
    - copy-on-reference
        - nejprve se přenese stav procesu potřebný pro běh (registry, kanály, ...)
        - přesun adresového prostoru odložen
        - stránky na cílové stanici označeny jako neprezentní (jako by byly odloženy na disk)
        - při přístupu na stránku se vyvolá vyjímka, obsluha obsah přenese
        - nutnost evidence různých zdrojů dat pro každou stránku
            - swap, vícenásobná migrace, DSM, …
- co se zprávami
    - přesměrování dočasné/trvalé
    - migrace kanálů, spojení front
        - vytvoření nových kanálů / přepojení / zrušení starých

- implementace v praxi
    - DEMOS/MP
        - IPC pomocí linků spojených s procesy, migrace v jádře
        - max transparence, jednotne komunikacni rozhrani bez ohledu na umisteni procesu
    - Charlotte
        - IPC pomocí linků nezávislých na umístění procesů, možnost přesouvat linky
        - migracni politika v uziv. procesu, migrace v jadru
        - snaha o max. fault-tolerance
        - sber statistiky o procesu, pocitaci
    - V
        - transparence, minimalizace doby zmrazeni (precopying)
    - MOSIX
        - jeden z mala realne pouzivanych
        - rozsahle vyvazovani zateze
            - rozsireni struktury procesu o ruzne statistiky
                - cas od posledni migrace, cas na procesoru, ...
        - UNIX pro distr. prostredi (pouzivany 1988 az do 2004!)
    - Sprite
        - rychlost a transparence
        - predpoklad - hodne nevyuzitych uzlu behem nejake pauzy, nutno pak zase odmigrovat po navratu vlastnika
        - VM: kombinace presunu vseho + copy-on-reference
    - T4
        - byl by efektivni pro pouziti jako konvencni OS
        - silne prostredi pro pouziti jako DS
        - meziproc. komunikace, prenosove protokoly, vzdalena komunikace, name services, podpora pro high-level distrib. sluzby
        - splnil ucel pro vyukove/vyzkumne ucely, nesplnil ucel pro kazdodenni vyuziti (chybely aplikace)

Load balancing

- nutno porovnávat zatížení procesorů (nějak konzistentně), pak vybrat co bude migrovat a kam
- rozhodnutí o okamžiku migrace
    - jak porovnávat zatížení uzlů
    - udržování konzistence údajů
    - volba migrujícího procesu
    - volba příjemce
    - přenos a běh procesu
- algoritmy

- párový algoritmus
  - vytvářejí se páry, které se vzájemně vyvažují
  - A pošle B žádost o vytvoření páru se seznamem procesů
  - B odmítne / vytvoří pár / migruje
  - zatíženější uzel vybere proces podle míry vylepšení
    - ki = Ai / (Bi + ABi)
  - významné zlepšení stavu – migrace a další proces, jinak konec
- vektorový algoritmus
  - pouziva se pevny vektor zateze, prvni slozka L(0) vlastni zatez, na ostatnich slozkach ostatni uzly
  - periodicky kazdy uzel zjisti vlastni zatez a polovinu vektoru posle nahodnemu uzlu
  - pri prijmu prolozi hodnoty s vlastnim vektorem + pricte se komunikacni rezie
  - pouzito v MOSIXu
- bidding algoritmus
  - “inteligentni” alg.
  - pouziva se vyhodnocovaci bunka - excitatory, inhibitory, vystup
  - procesy se pravidelne vyhodnocuji, pri vystupu pod urcity prah se migruje:
    - broadcastuje zadost o nabidku (RFB) do nejake max vzdalenosti d
    - odpovedi s nabidkou se zkoriguji o rezii
    - bereme nejlepsi nabidku, pokud zadna nedorazi, zvetsime d
  - obtizna kvantifikace vlastnosti procesu
- UI/math-based
  - zpetne uceni, bayesian decisions -- neprosadily se - moc komplikovane
- centralizovaný / hierarchický algoritmus
  - centralizovaný manažer, zná zátěž všech uzlů, velí
  - hierarchicky organizované skupiny, nadřazení manažeři
- lokalni algoritmus
  - znama jen lokalni zatez
  - nastaven prah - pokud ji proces preleze, posila n uzlum zadost o prijeti, vybere si nejlepsi
  - velmi jednoduche, velmi suboptimalni, velmi naprd

Replikace

- replikace souborů - udržování více kopií na více fileserverech
  - spolehlivost (reliability) - při havárii serveru nejsou ztracena data
  - dostupnost (availability) - k souborům lze přistupovat i při výpadku serveru
  - výkon (performance) - lze přistupovat k nejbližším datům, rozdělení výkonu
- typy:
  - explicitní replikace
    - 'uživatel' se sám stará o udržování konzistence
  - odložená replikace
    - zápis do primární repliky, aktualizace sekundárních
  - skupinová komunikace
    - zápisy simultánně zasílány všem dostupným replikám
- aktualizacni protokoly
  - problem aktualizaci kopii
    - uvazujeme, ze soubor je replikovan na N serverech
    - moznosti:
      - primarni kopie vitezi
      - vetsinove/vazene hlasovani
        - pro aktualizaci souboru musi klient pozadat vetsinu serveru (pres polovinu) a ziskat od nich schvaleni
        - pokud chce klient cist soubor musi shromazdit read quorum N_R, podobne pri zapise N_W, kde plati:
          - $N_R+N_W > N$ &
            - zabranuje read-write konfliktum (unrepeatable reads)
          - $N_W > N/2$
            - zabranuje write-write konfliktum (overwriting uncommitted data)
          - toto je <u>vazene hlasovani</u>, vetsinove je kdyz $N_R = N_W$
        - napr. ROWA schema (Read One, Write All) je korektni
      - hlasovani s duchy
        - bezdatový (dummy, ghost) server, obsahuje pouze verze, žádná data
        - neúčastní se hlasování o čtení, pouze o zápisu
      - dynamická kvóra

Klientocentrické konzistenční modely

- u konzistenčních modelů DSM byl pohled na sdílená data různými procesy
- replikovaná databáze
    - zápisy málo časté, masivně paralelní čtení
    - není potřeba vzájemná synchronizace klientů
    - WWW + cache, mobile computing, News, ...
- u klientocentrického modelu je pohled na replikovaná data jedním procesem
- eventuální konzistence
    - po ukončení všech zápisů budou všechny repliky v konečném čase aktualizovány
    - tolerace pomerne vysoke urovne nekonzistence
    - problem: muze se stat, ze při připojení k jedné replice uživatel vidí zprávy, po připojení k jiné replice některé zprávy (které již viděl) 'ještě' nevidí
- monotonní čtení
    - po přečtení hodnoty x všechna další čtení vrátí stejnou nebo novější hodnotu
    - priklad: při připojení k jiné replice uživatel vidí všechny dosud přečtené zprávy
- monotonní zápis
    - zápis proměnné je proveden před jakýmkoliv následným zápisem této proměnné
    - nekdy je dulezite propagovat vsechny write operace ve spravnem poradi na vsechny kopie
    - priklad: CVS commit na různých replikách
- čtení vlastních zápisů
    - zápis proměnné je proveden před jakýmkoliv následným čtením této proměnné
    - priklad: aktualizovani webovych stranek, nechceme si prohlizet kopie z cache
- zápisy následují čtení
    - zápis proměnné po předchozím čtení této proměnné je proveden na stejné nebo novější hodnotě
    - priklad: zapis odpovedi do news groups se provede tam, kde je i prectena hodnota
- naivní implementace
    - každému zápisu je přiřazen globálně jednoznačný identifikátor WID
        - přiřazuje replika kde byl zápis proveden klientem: WID = repl_id + loc_id
    - pro každého klienta udržujeme dvě množiny identifikátorů WID:
        - read-set
            - WID relevantni pro read operace vykonane klientem
        - write-set
            - WID relevantni write operacim vykonanym klientem
    - monotónní čtení
        - při čtení replika serveru ověří podle read-set aktuálnost svých zápisů
            - při chybějících zápisech provede synchronizaci nebo forwarduje čtení
        - po čtení si klient aktualizuje read-set podle repliky ze které četl

    - monotónní zápis
        - při zápisu replika ověří podle write-set aktuálnost svých zápisů
            - chybějí-li nějaké, zapíše je
        - po zápisu si klient aktualizuje write-set
        - problem monotonního cteni a zapisu - neomezeny rust read/write setu
    - čtení vlastních zápisů
        - při čtení replika ověří podle write-set aktuálnost svých zápisů
            - jiné možné řešení - forward čtení na aktuální repliku
    - zápisy následují čtení
        - aktualizace repliky podle read-set
        - aktualizace read-set i write-set klienta
- efektivější implementace
    - seskupení do relací / sessions
        - typicky vázané na aplikaci nebo modul
        - při ukončení / restartu smazání množin
        - neřeší problém trvale běžících aplikací
    - nebo reprezentace množin pomoci vektorovych hodin
        - umoznuji efektivni operace nad mnozinami
        - klient posila sve aktualni VT pri read/write pozadavku
        - replika serveru při zápisu přiřadí WID a lokální TS(WID)
        - každá replika Si udržuje RCV(i)[j] - TS poslední operace <u>zápisu</u> přijatá replikou Si od Sj
        - při přijetí žádosti o čtení nebo zápis replika vrátí aktuální RCV(i)
        - read-set i write-set jsou reprezentovány vektorovými hodinami
        - obecná pravidla:
            - VT(A)[i] = maximální TS operací z A iniciovaných na replice Si
            - sjednocení: VT(A+B)[i] = max(VT(A)[i], VT(B)[i])
            - test podmnožiny: A in B $\Leftrightarrow$ VT(A) $\leq$ VT(B)
        - po přijetí TS si klient aktualizuje read-set nebo write-set
            - čtení: VT(RS)[j] = max(VT(RS)[j], RCV(i)[j]) forall j
            - zápis: VT(WS)[j] = max(VT(WS)[j], RCV(i)[j]) forall j
        - read/write-set reprezentuje poslední operace zápisu, které klient viděl/zapisoval

Epidemické protokoly

- implementace eventuální konzistence
- optimalizace komunikace ve VELMI rozsahlych systemech, neresi konflikty
- teorie epidemii - infekcni nakaza
    - rozsireni infekce co nejrychleji, na co nejvetsi pocet uzlu
- servery jsou: infected (rozšiřují "epidemii"), removed (data mají ale nerozšiřují), susceptible (data nemají)
- antientropie
    - server P vybere nahodne server Q k vymene dat
    - mozne vymeny:
        - push
            - server P pushne pouze svoje zmeny na Q
            - pri velke infikaci mala pst, ze se zmeny rozsiri
        - pull
            - server P pullne zmeny z Q
            - zpocatku se zmeny rozsiruji pomalu, nakonec cela mnozina
        - push/pull
            - vyhody obou postupu
- gossiping
    - pokud byl P aktualizovan, kontaktuje nějaký další server Q, aby šířil update; jestliže Q už update má, P se s pravděpodobností 1/k nastaví jako removed
        - nezarucuje rozsireni vsem
    - kombinuje se s periodickym pullem
- problem mazani dat
    - naivne: smazani dat → smazani vsech info o datech
        - dusledek: entropie jinym kanalem data zase obnovi
    - reseni: certifikaty smrti
        - zaznam o smazani
        - problem: neomezeny narust certifikatu
            - reseni: v konecnem case certifikat zanikne, podminkou je konecny cas rozsireni infekce
        - asi neco jako kdyz si vymazu bookmark v chrome a na jinem PC, kde jeste je se sesynchronizuje a je zpatky
- aplikace - agregace dat
    - podproblém: spočítejte průměr
        - uzel i: xi = iniciální (libovolná) hodnota
        - epidemicky: (xi, xk) = (xi + xk) / 2
        - xi → avg forall n (xn)
    - iniciátor xi = 1, ostatní xi = 0
        - xi → 1 / N