Modely distribuovaných systémů - základní pojmy a pricipy, synchronní a asynchronní komunikace. Synchronizace. Detekce ukončení. Problém vzájemného vyloučení a problém uváznutí a jejich řešení. Problém volby vedoucího prvku - vliv topologie a její znalosti/neznalosti na složitost řešení problému.

1 Modely

- Nezávislé procesy ktoré komunikujú správami po dopredu danej sieti. Problémy:
  - Procesy sú nespoľahlivé: a) Neznámy execution time (pustí sa GC a čo teraz?) b) Proces môže zlyhať c) Proces môže podvádzať
  - Sieť je nespoľahlivá: a) Správy môžu byť rôzne oneskorené b) Poradie správ sa môže meniť c) Správy môžu byť zdvojené/strácať sa d) Môžu vznikať falošné správy
- Záleží na topológií siete — ring, star, úplný graf, mriežka, torus, strom, hypercube, unknown atd. — často sa na skutočnej topológií vytvorí virtuálna úplná topológia, lebo algoritmy sú potom jednoduššie.
- Komunikácia: Asynchrónna (send and forget), Synchrónna (mám potvrdenie o prijatí)
- Zložitosť: *Komunikačná* (počet správ), časová, priestorová.
- Synchronizácia: No ordering, Causal ordering, Absolute ordering. Causal ordering: Lamport a Vector clock — happens before relácia.

2 Algoritmy

2.1 Mutual Exclusion

- Primitívny prístup: Na ringu koluje token.
- Lamport: Každý proces si drží frontu zoradenú podľa causal clock. Keď chcem ísť do kritickej sekcie, pošlem request všetkým a čakám kým mi všetci odpovedia. Ak som stále na vrchole fronty, môžem ísť do kritickej sekcie. Inak čakám na release. Na konci kritickej pošlem všetkým release, by vedeli že ma môžu vyhodiť z fronty. (lineárny)
- Raymond: Mám kostru na sieti a každý node má rodiča. Vždy si pýtam ktorým smerom je práve token a tým smerom posielam requesty. Každý uzol si pamätá requesty a keď skonči kritická sekcia, pošle token podľa request fronty. Celkovo len posielanie tokenu po strome miesto kruhu. (logaritmický)
- Maekawa: Mám kvóra veľkosti sqrt a vždy sa pýtam len môjho kvóra s tým že musí platiť že všetky kvóra majú pairwise neprázdny prienik.

2.2 Termination Detection

- Dijkstra-Scholten: Prvé requesty vybudujú strom v rámci ktorého ak niekto skončí so všetkým (mám counter na request/ack), tak notifikujem otca (ak sa stanem idle a znovu sa zobudím, môžem dostať nového otca, len ho nemôžem meniť počas aktivity). (lineárne)

- Safra: Každý proces si drží počet odoslané - prijaté správy a flag o tom, či od posledného posunutia tokenu pracoval. Posunutím tokenu sa flag nuluje. Na začiatku posielam čistý token s nulou. Procesy k tomuto pripočítavajú ich counter a ich flag. Ak mám na konci nulu a čistý flag v tokene aj u seba, končím. Ak nie, musím ísť odznova s čiernym/bielym tokenom podľa toho či som od minula pracoval. (counter potrebujem na eleiminaciu sprav v prechode, flag potrebujem kvoli predbiehaniu tokenu)

2.3 Leader Election

- Všetky procesy sú na začiatku rovnaké. Na konci má práve jeden flag leader.

- Pokiaľ procesy nemajú unikátne ID alebo nejaký zdroj náhody, election nemusí byť možná (symetrický systém).

- Stromy: Inicializácia z listov, vyhráva najmenšie ID (nemusím poznať koreň). Algoritmus: Zbieram ponuky (a držím si minimum) or mojich susedov. Pokiaľ mám len jedného suseda od ktorého som ponuku nedostal, pošlem mu ponuku ako môj súčasný stav (tu začínajú listy) a čakám kým mi sused nepošle odpoveď na ponuku (s nejakým ID). Aktualizujem minimum podľa tohoto výsledku a notifikujem všetkých ostatných. (lineárne správ)

- Ring (Chang-Roberts): Pošlem moje ID po kruhu. Forwardujem len IDčka ktoré sú menšie ako ja (a ak ich vidím viem že som prehral). Keď uvidím znova moja ID, som zvoelný a pošlem všetkým na kruhu víťaznú správu aby vedeli že už nemusia čakať. (kvadraticky sprav, priemer je nlogn)

- Ring (Peterson): Algoritmus funguje po kolách pričom v každom kole sa moja skupina porovná so susedmi a pokiaľ nemá menšie ID, tak prehráva a pridáva sa k víťaznej skupine. Problém je že toto vyžaduje obojsmerný kruh. To sa obíde tak, že IDčka v každom kole rotujú o jeden proces ďalej. Takže ja ako proces si zistím ID predchodcu a potom moje ID (alebo ID predchodcu ak je menší) postuniem ďalšiemu procesu, ktorý ho porovná so sebou a osvojí si víťaza. (nlogn zložitosť)

- General: Voľba vlnou — V každom uzle sa iniciuje prehľadávanie siete, ale iba jeden uzol (leader) dostane ack že sieť je prehľadaná, lebo vlny sa budú navzájom rušiť až kým prežije len jedna (Obecne kvadratická zložitosť - N vĺn o veľkosti N)

- General (GHS): Voľba kostrou — Predpokladá váhy na hranách, aby mohol budovať min. kostru. Graf je rozdelený na fragmenty. Fragmenty sa v rámci seba

dohodnú na minimálnej outgoing hrane. Pokiaľ za ňou leží menší/rovný fragment, tak sa k nemu pripojíme. Ak sme väčší, tak čakáme (iniciuje vždy ten menší). (NlogN + E)

- General (KKM): Voľba traversalom — algoritmus je zaujímavý, lebo ukazuje že zložitosť voľby je spojená so zložitosťou traversalu. Princíp: Fungujem podobne ako Peterson, po leveloch. Mám tri typy tokenov: annexing, chasing a waiting. Ak token stretne hocikoho s vyšším levelom, okamžite umiera. Ak stretne niekoho s rovnakým, tak sa pobijú a vznikne nový token ktorý je o level vyššie. Ak token prejde uzlom kde už bol token s lepším ID, tak sa z neho stane chsing token a pokúsi sa dobehnúť annektora a spojiť sa s ním. Ak annektor prejde uzlom kde bol naposledy token s horším ID, tak je waiting, lebo to znamená že ma niekto bude naháňať a ja ho chcem počkať. ((1 + logk) * f(N) - logaritmus krát zložitosť traversalu)