Lokálny algoritmus procesu

Pätica $\left(Z,\, I,\, \rightarrow^{i},\, \rightarrow^{s},\, \rightarrow^{r}\right)$, kde:

- Z - množina stavov
- $I \subseteq Z$ - množina počiatočných stavov
- $\rightarrow^{i} \subseteq Z \times Z$ - prechodová relácia pre internú udalosť
- $\rightarrow^{r} \subseteq Z \times M \times Z$ - prechodová relácia pre udalosť receive, M - množina možných správ
- $\rightarrow^{s} \subseteq Z \times M \times Z$ - prechodová relácia pre udalosť send

Distribuovaný algoritmus

DA pre množinu procesov $\mathbb{P} = \{p_1, ..., p_n\}$ je množina lokálnych algoritmov, jeden pre každý proces z $\mathbb{P}$.

Systém pod asynchrónnou komunikáciou

Systém indukovaný DA-om pre $\mathbb{P} = \{p_1, ..., p_n\}$ pod asynchrónnou komunikáciou je $S = (C, \rightarrow, I)$, kde

- $C = \{(c_{p_1}, ..., c_{p_n}, M) \mid (\forall p \subseteq \mathbb{P}.c_p \in Z_p),\, M \in \mathcal{M}(M)\}$, pozn. wtf znamená $M \in \mathcal{M}(M)$? $M \in \mathcal{P}(M)$?
- $\rightarrow = \bigcup\limits_{p \in \mathbb{P}} \rightarrow_p$, kde $\rightarrow_p$ je množina dvojíc $\left((c_{p_1}, ..., c_{p_i}, ..., c_{p_n},\, M_1),\, (c_{p_1}, ..., c'_{p_i}, ..., c_{p_n},\, M_2)\right)$
  - $\left(c_{p_i},\, c'_{p_i}\right) \in \rightarrow^{i}_{p_i}$, $M_1 = M_2$ (interná udalosť, neposielali sa správy)
  - $\exists m \in M \left(c_{p_i},\, m,\, c'_{p_i}\right) \in \rightarrow^{s}_{p_i}$, $M_2 = M_1 \cup m$ (udalosť send, pribudla správa m)
  - $\exists m \in M \left(c_{p_i},\, m,\, c'_{p_i}\right) \in \rightarrow^{r}_{p_i}$, $M_1 = M_2 \cup m$
- $I = \{(c_{p_1}, ..., c_{p_n},\, M) \mid (\forall p \subseteq \mathbb{P}.c_p \in I_p),\, M = \emptyset\}$

Systém pod synchrónnou komunikáciou

Systém indukovaný DA-om pre $\mathbb{P} = \{p_1, ..., p_n\}$ pod synchrónnou komunikáciou je $S = (C, \rightarrow, I)$, kde

- $C = \{(c_{p_1}, ..., c_{p_n}) \mid (\forall p \subseteq \mathbb{P}.c_p \in Z_p)\}$
- $\rightarrow = \bigcup\limits_{p \in \mathbb{P}} \rightarrow_p \cup \bigcup\limits_{p,q \in \mathbb{P},\, p \neq q} \rightarrow_{pq}$, kde
  - $\rightarrow_p$ je množina dvojíc $\left((c_{p_1}, ..., c_{p_i}, ..., c_{p_n}), (c_{p_1}, ..., c'_{p_i}, ..., c_{p_n})\right)$, $\left(c_{p_i},\, c'_{p_i}\right) \in \rightarrow^{i}_{p_i}$
  - $\rightarrow_{pq}$ je množina dvojíc $\Big((..., c_{p_i}, ..., c_{p_j}, ...), (..., c'_{p_i}, ..., c'_{p_j}, ...)\Big)$, $\exists m.\, \left(c_{p_i},\, m,\, c'_{p_i}\right) \in \rightarrow^{s}_{p_i} \wedge \left(c_{p_j},\, m,\, c'_{p_j}\right) \in \rightarrow^{r}_{p_i}$
- $I = \{(c_{p_1}, ..., c_{p_n}) \mid (\forall p \subseteq \mathbb{P}.c_p \in I_p)\}$

Ak Q je invariant a P je Q-derivát, potom $Q \wedge P$ je invariant

Nech $S = (C, \rightarrow, I)$ a P, Q sú predikáty
$\{P\} \rightarrow \{Q\} \Leftrightarrow (\forall \gamma \rightarrow \delta \in \rightarrow .P(\gamma) \Rightarrow Q(\delta))$

P je invariantom systému S, ak $\forall \gamma \in I.P(\gamma) \wedge \{P\} \rightarrow \{P\}$.

P je Q-derivát, ak

1. $\forall \gamma \in I,\, Q(\gamma) \Rightarrow P(\gamma)$
2. $\{Q \wedge P\} \rightarrow \{Q \Rightarrow P\}$

Q je invariant, preto $\forall\gamma \in I.Q(\gamma)$ a keďže P je Q-derivát, $\forall\gamma \in I.Q(\gamma) \Rightarrow P(\gamma)$, teda $\{P(\gamma)\} \Rightarrow \{Q(\gamma) \wedge P(\gamma)\}$

Nech $\gamma \rightarrow \delta$, $Q(\gamma) \wedge P(\gamma)$. Keďže Q je invariant, platí $Q(\delta)$.
Keďže $\{Q \wedge P\} \rightarrow \{Q \Rightarrow P\}$, $Q(\delta) \Rightarrow P(\delta)$, teda $Q(\delta) \wedge P(\delta)$.

Predikát term, korektnosť, uviaznutie

Nech S je prechodový systém a P je predikát (cieľ).

term je predikát, ktorý je (ne)pravdivý v (ne)terminálnych konfiguráciách.

Systém S končí korektne, ak predikát $(term \Rightarrow P)$ v S vždy platí.

Uviaznutie nastáva v terminálnej konfigurácii, kde sa cieľ P nedosiahol.

Norma

Nech S je prechodový systém a P je predikát. Funkcia f z C do dobre založenej množiny W sa naýva norma (vzhľadom na P), ak $\forall\gamma \rightarrow \delta.\,(f(\gamma) > f(\delta) \vee P(\delta))$.

Lamportove hodiny

Nech $\Theta(a)$ je dĺžka najdlhšej postupnosti $(e_0, e_1, e_2, ..., e_k)$ takej, že $e_0 \prec e_1 \prec e_2 \prec ... \prec e_k = a$.

- Ak b je interná alebo send udalosť, a predchádzajúca udalosť toho istého procesora, potom $\Theta_L(b) = \Theta_L(a)+1$.
- Ak r je receive, i je prdchádzajúca udalosť toho istého procesora a s je odpovedajúca send udalosť, potom $\Theta_L(r) = max\,\{\Theta_L(i), \Theta_L(s)\} + 1$.
- Ak a je prvá udalosť, $\Theta_L(a) = 0$.

Strom najkratších ciest

Pre každý vrchol v grafe existuje strom, kde pre každý vrchol je vzdialenosť od koreňa rovná minimálnej vzdialenosti oboch vrcholov v grafe.

Konštrukcia. Sink-tree. To sa dá vymyslieť „on-the-fly".

Problémy Touegovho algoritmu

- kvôli voľbe pivota vyžadujeme znalosť všetkých vrcholov
- vyžadujeme informáciu $(d(u,w) + d(w,v) < d(u,v))$, ktorá nie je dostupná vo vrchole, ani v susedných vrcholoch.

Chandy-Misra:

$$d(u,v) = \begin{cases} 0, & u = v \\ min_{w \in Neigh_u}\{\omega_{uv} + d(w,u)\}, & inak \end{cases}$$

Netchange

Algoritmus uchováva množinu susedov všetkých vrcholov, pri poruche/oprave kanála patrične upraví množiny a prepočíta podľa nich smerovacie tabuľky. V prípade, že sa vrcholu zmení najmenšia vzdialenosť do nejakého uzla, propaguje uzol túto informáciu jeho susedom (v procedúre $Recompute(v)$).

Premenné

- $Neigh_u$ - Aktuálna množina susedov vrchola u.
- $D_u[v]$ - odhad $d(u, v)$
- $Nb_u[v]$ - prvý vrchol na ceste s $D_u[v]$
- $ndis_u[w, v]$ - odhad $d(w, v)$

Správy

- $\langle mydist, v, d\rangle$ - vrchol oznamuje svoju vzdialenosť d od vrchola v.
  Po prijatí spraví algoritmus odhad pre vrcholy idúce cez v a prepočíta tabuľky.
- $\langle fail, v\rangle$ - správa od w oznamuje poruchu kanála (v, w).
  $Neigh_u \leftarrow Neigh_u - \{w\}$, prepočíta tabuľky pre vrcholy cez v.
- $\langle repair, v\rangle$ - správa od w oznamuje obnovenie kanála (v, w).
  $Neigh_u \leftarrow Neigh_u \cup \{w\}$, prepočíta tabuľky pre vrcholy cez v a pošle ich vrcholu v.

Acyklická orientácia, acyklické pokrytie

Acyklická orientácia grafu G je orientovaný acyklický graf, ktorý vznikne z grafu G zorientovaním hrám. Postupnosť $G_1, ..., G_B$ acyklických orientácií grafu G je acyklické pokrytie veľkosti B pre množinu ciest $\mathcal{P}$, ak $\forall p \in \mathcal{P}$, P možno zapísať ako zreťazenie nanajvýš B ciest $P_1, ..., P_k$, $k \leq B$ takých, že P_i je cesta v G_i.

Ak existuje acyklické pokrytie veľkosti B, na prevenciu uviaznutia stačí vo vrchole nanajvýš B bufferov. Pri generovaní packetu p vo vrchole u sa sa tento umiestni do buffera $b_u[1]$. Pri forwardovaní packetu z vrchola u do vrchola w číslo buffera ostáva, ak môžeme pokračovať v grafe G_i, inak preskočíme do grafu G_{i+1}.

Stabilizácia

Všetky konfigurácie sú počiatočné. Systém S stabilizuje k špecifikácii P, ak existuje množina legitímnych konfigurácií $\mathcal{L}$ taká, že spĺňa

- korektnosť - každá realizácia, ktorá začína v $\mathcal{L}$, spĺňa P.
- konvergencia - každá realizácia obsahuje konfiguráciu z $\mathcal{L}$.

Výhody

- odolnosť voči chybám, spamätanie sa z chýb
- jednoduchšia inicializácia
- robustnosť pre dynamické topológie

Nevýhody

- počiatočná nekonzistencia - kým algoritmus nezačne pracovať korektne, môže dávať nesprávne výstupy
- chýba detekcia stabilizácie - procesy nevedia, kedy pracujú korektne
- veľká zložitosť

Vlnové algoritmy, traverzovanie

Vlnový algoritmus je DA, ktorý spĺňa

- terminácia - $\forall C.\, |C| < \infty$
- rozhodnutie - $\forall C. \exists e \in C$, e je (špecíalna interná) *decide* udalosť
- závislosť - v každom výpočte každej decide udalosti predchádza udalost v každom procese

Traverzovací algoritmus je vlnový algoritmus, ktorý navyše spĺňa

- algoritmus má jediného iniciátora, ktorý jediný rozhoduje
- existuje úplné usporiadanie procesov

Traverzovací algoritmus je $f(x)$-traverzovací, ak po vykonaní x krokov (posunov tokena), je navštívených $max\{f(x), x\}$ vrcholov.

Algoritmus je f-traverzovací pre triedu sietí, ak

- je traverzovací pre túto triedu
- v každom výpočte je po $f(x)$ posunoch tokena navštívených aspoň $min\{N,\, x+1\}$ procesov.

Pre torus (kde vrcholy majú zmysel pre orientáciu) a hyperkocku existujú x-traverzovacie algoritmy.

Detektory chýb

Detektor chýb je algoritmus, ktorý identifikuje procesy, ktoré prestali pracovať.

- T - čas
- $F(\tau)$ - množina procesov, ktoré v čase τ nepracujú.
- $Crash(F) = \bigcup_{\tau \in T} F(\tau)$ - množina chybných procesov
- $Corr(F) = \mathbb{P} - Crash(F)$ - množina korektných procesov.
- $H(p, \tau)$ - množina procesov, o ktorých procesor p predpokladá, že manjú v čase τ fatálnu chybu.

Detektor je úplný vtedy, keď ak nejaký proces prestane pracovať, v konečnom čase ho každý procesor bude považovať za chybný. $\exists \tau \in T. \forall p \in Crash(F). \forall q \in Corr(F). \forall \tau' \in T. \tau' \geq \tau,\, p \in H(q, \tau')$

Detektor je (úplne) presný, ak podozrieva len ozaj chybné procesy. $\forall \tau \in T. \forall p, q \in \mathbb{P}. p, q \notin F(\tau) \Rightarrow p \notin H(q, \tau)$

Konštrukcia úplne presného detektora

- každý proces pošle každých σ časových jednotiek správu *alive*
- existuje horný odhad μ na komunikačné zdržanie

Každý proces, od ktorého neprišla počase $\sigma + \mu$ časových jednotiek správa *alive*, prestal pracovať.

Vzájomné vylúčenie - token v kruhu (Dijkstra)

Procesy $p_0, ..., p_{N-1}$ tvoria jednosmerný kruh, p_i má hodnotu $\sigma_i \in \{0, ..., K\}$, kde $K > N$.

- p_i, $0 < i < N$ je privilegovaný, ak $\sigma_{i-1} \neq \sigma_i$
- p_0 je privilegovaný, ak $\sigma_0 = \sigma_{N-1}$

Každý privilegovaný proces môže zmeniť svoju hodnotu, čo spôsobí stratu privilégia

- $\sigma_i \leftarrow \sigma_{i-1}$, ak $\sigma_i \neq \sigma_{i-1}$, $0 < i < N$
- $\sigma_i \leftarrow (\sigma_{N-1} + 1) \, mod \, K$, keď $\sigma_0 = \sigma_{N-1}$

Algoritmus stabilizuje k vzájomnému vylúčeniu. V každej konfigurácii má privilégium aspoň jeden proces. Krok hodnotu privilegovaných nezvyšuje. $\mathcal{L}$ - množina konfigurácií, v ktorých má privilégium práve jeden proces. Výpočet, ktorý začína v legitímnej konfigurácii, spĺňa vzájomé vylúčenie - token cirkuluje v kruhu. Vzájomné vylúčenie sa dosiahne po $O(N^2)$ udalostiach.

Dohoda

Nech $t \geq \frac{N}{2}$. Potom neexistuje t-f-robustný algoritmus dohody, založený na oneskorene úplne presnom detektore chýb.

Chandra-Touegov algoritmus dohody

t-f-robustný algoritmus dohody s rotujúcim koordinátorom, $t < \frac{N}{3}$ (zjednodušená verzia, pôvodná je funkčná pre $t < \frac{N}{2}$), oneskorene slabo presný detektor chýb .

Kolo k koordinuje proces p_c, pričom $c = (k \, mod \, N) + 1$

- každý proces q pošle $\langle val_q, k \rangle$ procesu p_c
- p_c počká na prijatie $N - t$ správ tohoto kola, nastaví val_{p_c} podľa väčšiny (pri rovnosti 1). Ak sú všetky prijaté správy rovnaké, nastaví agr_{p_c} na T, inak na F.
- p_c spustí broadcast $\langle agr_{p_c}, val_{p_c}, k \rangle$
- každý proces q čaká
  - buď na správu od p_c - vtedy preberie prijatú hodnotu a ak $agr_{p_c} = T$, q rozhodne
  - alebo dovtedy, kým nezačne podozrievať p_c

Voľba šéfa v kruhu s $O(NlogN)$ správami

Franklin-82

- na začiatku sú iniciátori *active*, neiniciátori *passive*; každý aktívny proces pošle svoju identifikáciu susedom
- pasívne vrcholy len posúvajú správy
- nech aktívny vrchol u dostane hodnoty v, w
  - ak $min\{v, w\} < u$, u sa stane *passive*
  - ak $min\{v, w\} > u$, u opäť pošle identifikáciu susedom
  - ak $min\{v, w\} = u$, u sa stane *leader*

Voľba šéfa v jednosmernom kruhu s $O(NlogN)$ správami

Porovnanie troch identifikátorov z vrcholov r, q, p sa realizuje vo vrchole najviac vpravo. $\rightarrow r \rightarrow q \rightarrow p \rightarrow$

- ak q má menší identifikátor ako r a p, p vie, že q je lokálne minimum a bude si pamätať jeho hodnotu
- ak nie, p sa stane pasívnym

Správy

- $\langle one, . \rangle$ - p sa dozvie identifikáciu v q
- $\langle two, . \rangle$ - p sa dozvie identifikáciu v r
- $\langle small, . \rangle$ - identifikácia *leadra*

Gallager-Humblet-Spira

Distribuovaný Kruskal.

- na začiatku je každý vrchol samostatným fragmentom
- vrcholy fragmentu F spoločne hľadajú minimálnu odchádzajúcu hranu e_F
- po nájdení e_F je opačný koniec vyzvaný na spoluprácu pri spájaní týchto framnetov
- algoritmus terminuje, keď ostane už len jediný fragment

Spájanie fragmentov

- meno fragmentu označíme FN
- úroveň fragmentu označíme L, na začiatku majú všetky fragmenty $L = 0$
- Ak spájame fragmenty s rovnakou úrovňou, nový fragment má meno aj úroveň pôvodne väčšieho
- Ak majú fragmenty rovnakú úroveň, úroveň sa zvýši o 1. Novým menom bude váha hrany, nazveme ju *jadro* (*core*).
- Fragmenty $F = (FN, L)$, $F' = (FN', L')$ sa spoja
  - $L < L' \wedge F \stackrel{e_F}{\rightarrow} F'. F \cup F' = (FN', L')$
  - $L < L' \wedge F' \stackrel{e_F}{\rightarrow} F. F \cup F' = (FN, L)$
  - $L = L' \wedge e_F = e_{F'}. F \cup F' = (\omega(e_F), L + 1)$

Keď vrchol u dostane správu $\langle test, L, FN \rangle$, skontroluje, či niektorá z jeho *basic* hrán nie je odchádzajúca

- u pošle $\langle test, L, FN \rangle$ do v
- ak v je vo fragmente FN, odpovedá $\langle reject \rangle$ a následve aj u zaradí v do $reject$
- inak, ak $L \leq level_v$, v odpovie $\langle accept \rangle$
- ak $L > level_v$, v pozdrží spracovanie $\langle test \rangle$ správy
- ak samotné v čaká na reakciu na testovaciu správu, ktorú poslalo u, v neodpovedá

Algoritmus pošle najviac $5NlogN + 2|E|$ správ.

Korach-Kutten-Moran

Iniciátor výberu spustí traverzovací algoritmus, pričom si označí svoj token. Pri stretnutí rôznych traverzovaní niektoré zanikne. K rozhodnutiu dôjde len v jednom traverzovaní, jeho iniciátor sa stane *leadrom*.

Pridáme tokenom informáciu o úrovni. Pri stretnutí dvoch tokenov rovnakej úrovne vznikne token, ktorého úroveň je o 1 väčšia. Ak token s menšou úrovňou vojde do vrchola, v ktorom je, alebo bol token s vyššou úrovňou, zaniká.

Aby sa tokeny rovnakej úrovne mohli porovnať, musia sa stretnúť v jednom procese. Realizácia pomocou stavov

- lev_p - úroveň procesu
- cat_p - (currently active traversal of p,) iniciátor posledného annex tokena forwardovaného procesom p
- $wait_p$ - úroveň tokena, ktorý čaká v p
- $last_p$ - informácia o susedovi, do ktorého p naposledy forwardovalo *annex* token úrovne lev_p
- $token(q, l)$ - q je iniciátor tokena, l je úroveň tokena

Premenovanie v prítomnosti chýb (ABND)

Pre $t \geq \frac{N}{2}$ neexistuje t-f-korektný algoritmus premenovania. ABND je deterministický algoritmus pre $t < \frac{N}{2}$.

- udržiava informáciu o tých, ktorých videl
- pri každej zmene množiny ju zakričí ostatným
- počíta, koľkokrát dostal info o aktuálnej množine; nech pre $V = V_p$ je to $N - t$, potom $V = Vp$ je stabilná množina
- menom je (s, r), kde $s = |V_p|$, r je poradie x_p vo V_p

ABND vyberá mená z množiny veľkosti $K = (N - \frac{t}{2}) \cdot (t + 1)$.

Počítanie na kruhu bez znalosti jeho veľkosti

Neexistuje deterministický procesom terminujúci algoritmus na výpočet nekonštantnej funkcie, ak veľkosť kruhu je neznáma.

Singh

Premenné

- $state_i$ - pasv, candidate, captured, elected
- $level_i$ - počet vrcholov zajatých v 1. fáze
- $step_i$ - počet krokov kandidáta v 2. fáze
- $owner_i$ - pre kandidáta 0, pre zajatca hrana vedúca do leadra
- $phase_i$ - budí sa fázou 1, do 2 prchádza kandidát, alebo po rijatí správy fázy 2

Správy

- $Capture(phase_i, level_i/step_i, i)$ - i sa pokúša zajať nový vrchol
- $Inform(dist)$ - i oznamuje uchvatiteľovi vzdialenosť ownera
- $Accept(Phase_i, Level_i)$ - akceptuje zajatie
- $Owner(id)$ - leader R_i oznamuje svoje id ostatným v R_i
- Ack - potvrdzuje prijatie $Owner$
- $Elected$ - leader oznamuje svoje id ostatným

Dobrev

Správy ($type$, id, r), kde

- $type \in \{target, bullet, die!, OK, alive, dead\}$
- id - číslo procesora
- r - číslo kola

Premenné, procesor si pamätá

- poslednú správu typu $target$, $alive$, $dead$
- najsilnejšiu správu, ktorú videl