

Stabiloivat synkronoijat ja nimeäminen

Mikko Ajoviita

2.11.2007

Synkronoija

- Synkronoija on algoritmi, joka muuntaa synkronoidun algoritmin siten, että se voidaan suorittaa synkronoimattomassa järjestelmässä.
- Algoritmin suunnittelu ja analyysi on usein helpompaa kuin synkronoidussa järjestelmässä.
 - Synkronisessa järjestelmässä prosessorit vaihtavat tilaa samanaikaisesti, joten tietyn alkutilan jälkeinen suoritus voidaan määritellä tarkasti.
 - Synkronoimattomassa järjestelmässä tietyn alkutilan jälkeinen suoritus taas ei ole yksiselitteinen.

- Aina ei kuitenkaan ole järkevää käyttää synkronoijaa, koska tehokkuus kärsii.
 - Kun etenemisnopeutta rajoitetaan synkronoinnilla, niin nopeimmat prosessorit joutuvat suorittamaan toimenpiteitä hitaimpien prosessorien tahtiin.
 - Lisäksi nopeammat prosessorit eivät voi auttaa hitaampia prosessoreja suorituksessa.
- Seuraavaksi tarkastelemme kolmea erilaista synkronoijaa
 - Yksinkertaista itsestabiloivaa alfa-synkronoijan rajoittamatonta versiota.
 - Rajoitettua versiota alfa-synkronoijasta.
 - Sekä beta-synkronoijaa.

Rajoittamaton alfa-synkronoija

- Alfa-synkronoija käyttää apunaan itsestabiloivaa datalink –algoritmia.
- Jokainen sanoma välittyy perille ja samoin kuittaus viestistä takaisin.
- Prosessori P lähettää viestin m prosessorille Q ja jää odottamaan kuittauksia. P ei lähetä uutta viestiä m' ennen kuin on saanut kuittauksen.
- Kuittausviesti voi sisältää tiedon prosessorin Q tilasta.

- Jokaisella prosessorilla on tilamuuttuja *phase_i*.
- Alfa-synkronoijan toiminta on määritelty seuraavasti:
 1. Kahden naapurin tilamuuttujien arvot saavat erota toisistaan korkeintaan yhdellä.
 2. Jokaista tilamuuttujaa kasvatetaan äärettömän monta kertaa.

Itsestabiloiva alfa-synkronoija (rajoittamaton versio)

1. **do** forever
2. **forall** $P_j \in N(i)$ **do** received_j := false
3. **do**
4. **DLsend** (phase_i)
5. **upon** DLreceive(ack_j , phase_j)
6. received_j := true
7. phase_{ji} := phase_j
8. **upon DLreceive**(phase_j)
9. phase_{ji} := phase_j
10. **until** $\forall P_j \in N(i)$ received_j := true
11. **if** $\forall P_j \in N(i)$ phase_i \leq phase_{ji} **then**
12. phase_i := phase_i + 1
13. **od**

- Algoritmin oikeellisuus perustuu kahteen lemmaan.

Lemma 1.

Jokaisen kelvollisen suorituksen jälkeen päädytään sellaiseen alkutilaan, jossa kaikkien naapuriprosessorien tilamuuttujat eroavat toisistaan korkeintaan yhdellä.

Lemma 2.

Jokaisessa kelvollisessa suorituksessa tilamuuttujia kasvatetaan äärettömän monta kertaa

- Rajoittamaton versio ei kuitenkaan toimi käytännössä.

Rajoitettu alfa-synkronija

- Ongelman kuvaus säilyy samana kuin rajoittamattomassakin versiossa.
- Tilamuuttajien arvossa on mukana *modulo* M , missä $M \geq N$ (N on prosessorien lukumäärä.)
- Jokaisella prosessorilla on lisäksi nollausmuuttuja *reset_i*, joka saa arvoja välillä $0-2N$.

- Ideana on selvittää eroaako kahden naapurin tilamuuttuja yli yhdellä toisistaan.
- Tällaisessa tapauksessa kaikkien prosessorien tilamuuttujat nollataan.
- Eli päästään alkutilaan, jossa kaikkien tilat ovat nolliä.
- Stabiloinnin jälkeen laskenta etenee kuin rajoittamattomassakin versiossa.

Nollaaminen

- Kun prosessori huomaa, että sen tilamuuttuja eroaa yli yhdellä naapurista, se asettaa oman nollausmuuttujansa nolaksi.
- Tämän seurauksena naapureiden nollausmuuttujat eivät voi olla suurempia kuin yksi.
- Naapureiden seuraavien naapureiden nollausmuuttuja ei voi olla suurempi kuin kaksi jne.
- Siten prosessori, joka asettaa nollausmuuttujansa nolaksi, aiheuttaa "nollausaallon" koko verkkoon.

Itsestabiloiva alfa-synkronoija (rajoitettu versio)

```
1  do forever
2.  forall  $P_j \in N(i)$  do receivedj = false
3.  PhaseUpdate = false
4.  ResetUpdate = false
5.  do
6.    DLsend(phasej, resetj) //start send to all neighbours
7.    upon DLreceive(ackj, phasej, resetj)
8.      receivedj = true
9.      UpdatePhase( )
10.   upon DLreceive(phasej, resetj)
11.     UpdatePhase( )
12.   until  $\forall P_j \in N(i)$  receivedj = true //all send terminated
13.   If ResetUpdate = false then
14.     if  $\forall P_j \in N(i)$  reseti ≤ resetj then reseti = min(2N, reseti+1)
15.     if PhaseUpdate = false and
16.      $\forall P_j \in N(i)$  phasei ∈ {phasej, (phasej-1)mod M} then
17.       phasei = (phasei +1)mod M
18. od
```

19. UpdatePhase()
20. $\text{phase}_{ji} = \text{phase}_j$
21. $\text{reset}_{ji} = \text{reset}_j$
22. **If** $\text{reset}_i > \text{reset}_{ji}$ **then**
23. $\text{reset}_i = \min(2N, \text{reset}_{ji} + 1)$
24. ResetUpdate = true
25. **If** $\text{phase}_{ji} \notin \{(\text{phase}_i - 1) \bmod M, \text{phase}_i, (\text{phase}_i + 1) \bmod M\}$ **then**
26. $\text{reset}_i = 0$
27. ResetUpdate = true
28. **If** $\text{reset}_i \neq 2N$ **then**
29. $\text{phase}_i = 0$
30. PhaseUpdate = true

- Algoritmin oikeellisuus perustuu kahteen lemmaan.

Lemma 1.

Jokainen kelvollinen suoritus päättyy turvalliseen tilaan, jos mikään prosessori ei aseta nollausmuuttujaansa nolaksi.

Lemma 2.

Jokainen kelvollinen suoritus päättyy turvalliseen tilaan, jos jokin prosessori asettaa nollausmuuttujansa nolaksi.

Beta-synkronoija

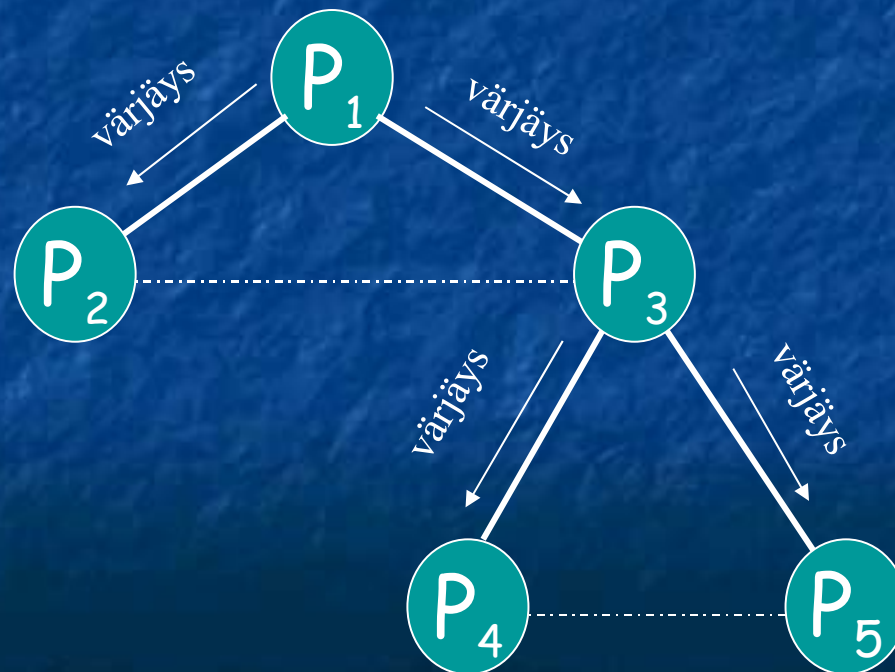
- Käytetään apuna jaettua muistia prosessorien välillä.
- Jokaisella prosessorilla on värimuuttuja.
- Tarkoituksena on värjätä verkko saman väriseksi.

Synkronointi voidaan jakaa kolmeen osaan:

1. Pohjustuksena muodostetaan virittävä puu, jossa on erikseen määritelty juuri. (Leader Election Algorithm and Spanning Tree Construction Algorithm).

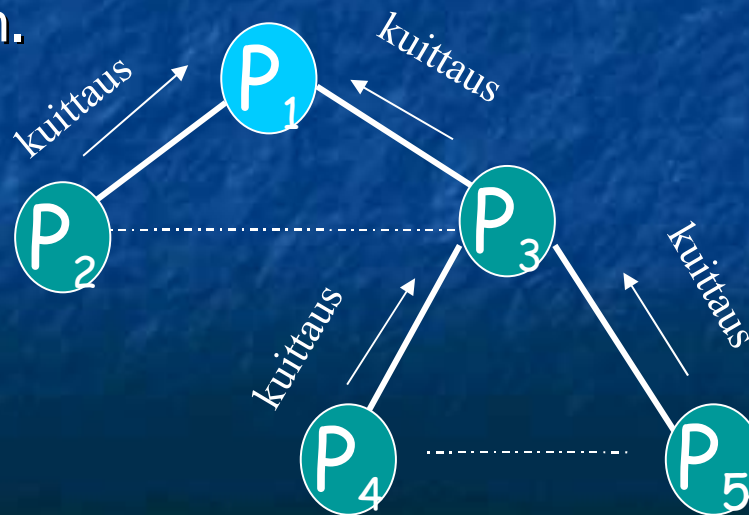
2. Virittävä puu värjätään juuresta lähtien.

- Juuri välittää tiedon omasta väristään lapsilleen puussa.
- Muut prosessorit lukevat vanhempansa värin ja välittävät sen omille lapsilleen puussa.
- Näin koko puu värjäytyy samalla värillä.



3. Kuittaus värjäyksen valmistumisesta välitetään juurelle.

- Kun prosessori on välittänyt värin lapsilleen, se jää odottamaan kuittausta.
- Kun prosessori saa tiedon lapsiltaan, että sen kaikki alipuut on värjätty juuren värillä, se välittää kuittauksen omalle vanhemmalleen.
- Lopulta juuri saa kuittauksen omilta lapsiltaan, että koko puu on värjätty.
- Tällöin juuri valitsee uuden värin ja välittää sen taas lapsilleen.



Itsestabiloiva beta-synkronoija

1. Root: **do** forever
2. **forall** $P_j \in \text{children}(i)$ **do** $lr_{ji} := \text{read}(r_{ji})$
3. **if** $\forall P_j \in \text{children}(i) (lr_{ji}.\text{color} = \text{color}_i)$ **then**
4. $\text{color}_i := (\text{color}_i + 1) \bmod (5n - 3)$
5. **forall** $P_j \in \text{children}(i)$ **do write** $r_{ji}.\text{color} := \text{color}_i$
6. **od**

7. Other: **do** forever
8. **forall** $P_j \in \{\text{children}(i) \cup \text{parent}\}$ **do** $lr_{ji} := \text{read}(r_{ji})$
9. **if** $\text{color}_i \neq lr_{\text{parent},i}.\text{color}$ **then**
10. $\text{color}_i := lr_{\text{parent},i}.\text{color}$
11. **else if** $\forall P_j \in \text{children}(i) (lr_{ji}.\text{color} = \text{color}_i)$ **then**
12. **write** $r_{i,\text{parent}}.\text{color} := \text{color}_i$
13. **forall** $P_j \in \text{children}(i)$ **do write** $r_{ij}.\text{color} := \text{color}_i$
14. **od**

Lemma 1.

Jokaisessa kelvollisessa suorituksessa, juuri vaihtaa väriään vähintään kerran jokaisen $2d+1$ syklin aikana.

Lemma 2.

Alkutilaan, jossa kaikkien prosessorien väri on sama, päästään $O(dn)$ syklin aikana.

- Algoritmi saavuttaa turvallisen tilan $O(dn)$ syklin kuluessa.

Nimeämistekniikka

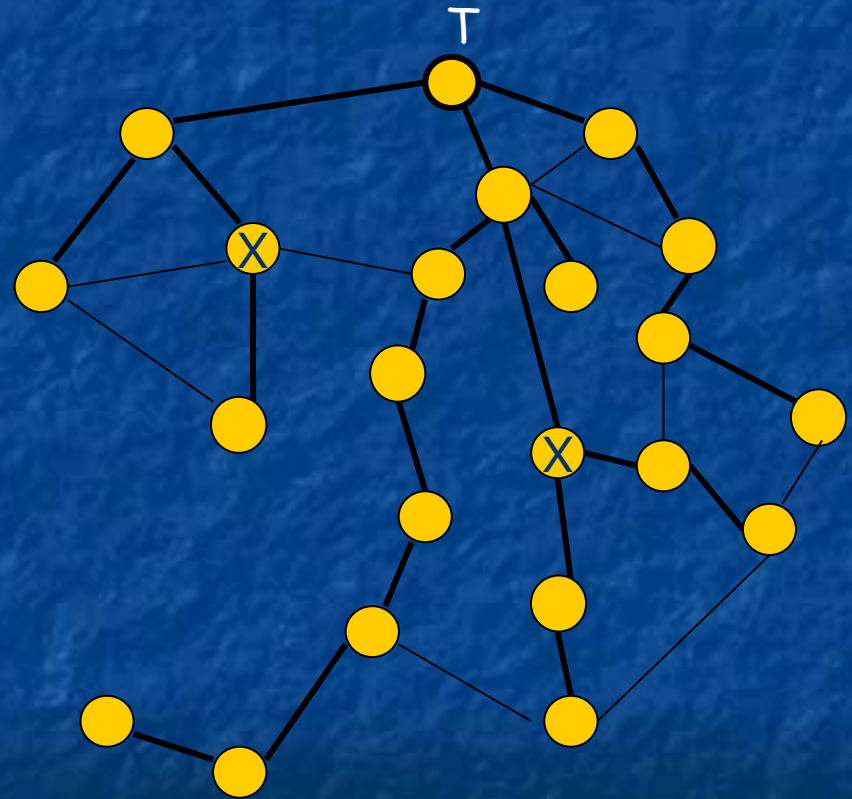
- Tavoitteena on nimetä prosessorit verkossa yksilöivillä tunnuksilla.
- Perustana Update-algoritmin muunnelmä sekä beta-synkronoija.
- Erilaisia tunnisteita suuri määrä.
- Satunnaisen tunnisteiden valinnassa apuna Scheduler-Luck peli.

Perusidea

- Aluksi prosessoreilla annetaan satunnaiset tunnisteet.
- Päivitystekniikan avulla muodostetaan virittävä puu tai puita.
- Värjäystekniikan avulla puut värjätään.
- Jos jokin tunniste esiintyy useampaan otteeseen, valitaan uusi tunniste.

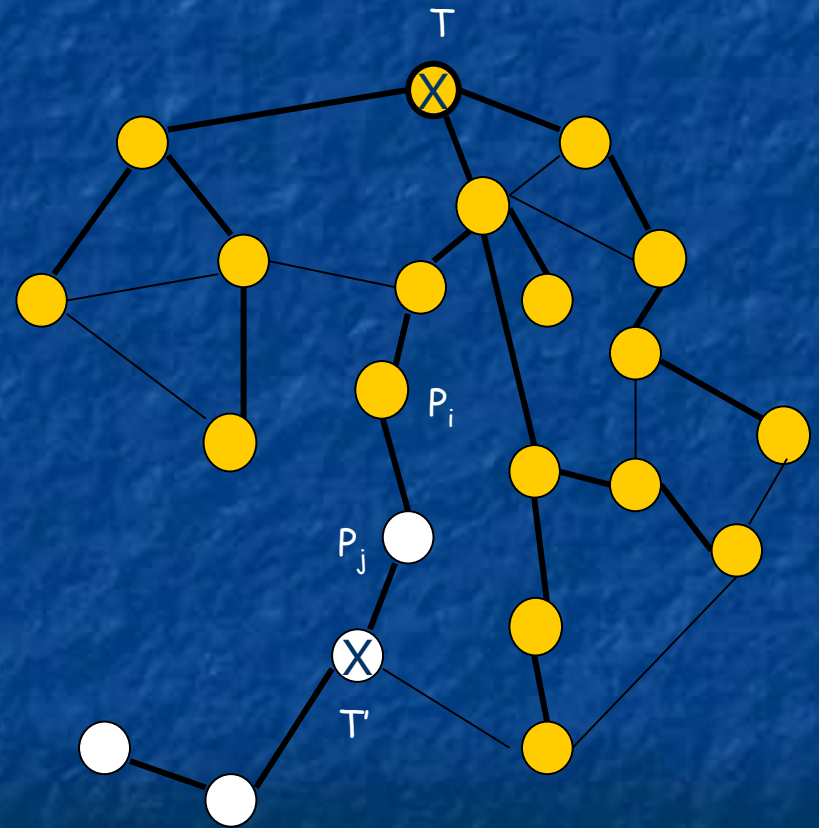
Tuplatunniste

- On muodostettu virittävä puu T.
- Se sisältää kuitenkin kaksi samaa tunnistetta X.
- Prosessorit välittävät vanhemmalleen alipuidensa tunnisteet.
- Juuri saa koko puun tunnisteiden listan.
- Tuplatunnisteet vaihtavat itselleen uuden tunnisteiden



Värjästekniikka

- On muodostunut kaksi erillistä puuta T ja T' .
- Puut erotetaan toisistaan väreillä.
- Prosessorit P_i ja P_j huomaavat kuuluvansa eri puihin.
- Tieto kulkee juurelle, joka vaihtaa tunnistettaan.



Algoritmin toiminta

- Virittävien puiden rakennus ja värjäys suoritetaan yhtä aikaa.
- Ensisijaisesti tutkitaan onko useampia puita.
- Jos näin, niin yksi juuri vaihtaa tunnisteensa ja prosessi aloitetaan alusta.
- Kun on jäljellä vain yksi puu, toissijaisesti tutkitaan onko puussa tuplatunnisteita.

- Mitään tunnistetta ei vaihdeta kuin kerran.
- Jos edelleen löytyy samoja tunnisteita, prosessi aloitetaan alusta.
- Turvallisen tilan odotusarvo on $O(d)$ sykliä.
- Näin kaikille prosessoreille on löydetty yksilöivät tunnisteet.