

Hajautetun laskennan vakaantuminen häiriöiden kestäessä

Jussi Brunberg

Esitelmä 9.11.2007 kello 11

Helsingin yliopiston tietojenkäsittelytieteen laitos, C221

Kirjoitelman 12.–29.10.2007 vakaa versio

1 Johdanto

Hajautetussa laskennassa on usein tarpeen varautua erilaisiin tilapäisiin toimintahäiriöihin. Itsestabiloiva (self-stabilizing) eli *vakaantuva* hajautettu algoritmi selviää järjestelmän tilan sekoittavista katastrofeista takaamalla, että oli muuttujien alkutila mikä tahansa, riittävän pitkän häiriöttömän kauden aikana laskennan tilojen jatkumo korjaantuu oikeelliseksi. Alkusu-
säyksen vakaantuvien algoritmien tutkimukselle antoi Edsger Dijkstran ratkaisu poissulkemisiongelmaan laskuyksikkörenkaassa [Dij73]. Hänen renkaasaan on laskennan vakaannuttua tasan yksi laskuyksikkösolmuista kerrallaan etuoikeutetussa tilassa. Sanotaan, että solmulla on vuoromerkki (token)¹. Jos toimintahäiriöiden takia renkaaseen ilmestyy ylimääräisiä vuoromerkkejä tai kaikki merkit katoavat, palauttaa laskuyksiköiden noudattama protokolla häiriöiden loputtua järjestelmän vähitellen tilaan, jossa vuoromerkkejä kiertää tasan yksi.

¹Larry Wall kutsuisi etuoikeutettua solmua kurpitsanhaltijaksi —
<http://www.perl.com/doc/manual/html/Porting/pumpkin.html>

Tässä esitelmässä tutustutaan mahdollisuuksiin taata hajautetun järjestelmän häiriöttömästi toimivien yksiköiden vakaa laskenta häiriöllisten yksiköiden rinnalla. Esitys perustuu Shlomi Dolevin kirjan Self-Stabilization [Dol00] lukuun 6. Tarkasteltaviin häiriöihin kuuluvat laskuyksiköiden jumittuminen ja torkahtelu, ja esimerkkiongelmaksi käytetään kellojen synkronointia. Vaikeimmat häiriöt liittyvät virheellisesti toimiviin yksiköihin, joiden ajatellaan tarkoituksellisesti juonittelevan muita yksiköitä vastaan ja jotenkin salaperäisesti onnistuvan keksimään aina tuhoisimmat mahdolliset valheet tai osatotuuksien yhdistelmät. Tällaista häiriökäyttäytymistä kutsutaan *bysanttilaiseksi* muinaisen Bysantin juonittelevan hovin mukaan².

Tarkastellaan esimerkkinä Leslie Lamportin, Robert Shostakin ja Marshall Peasen [LSP82] muotoilemaa bysanttilaisten kenraalien ongelmaa. Olkoot A , B ja C kolme kenraalia, joista yksi juonittelee bysanttilaisesti muita vastaan. Kenraali A ilmoittaa muille hyökkäyssuunnitelman. Olkoot mahdolliset suunnitelmat ”hyökätään aamunkoitteessa” ja ”odotetaan”, ja merkittäköön niitä 1:llä ja 0:lla. Suoraselkäisten kenraalien on selvitettävä mikä on yhteinen suunnitelma. Jos A juonittelee, ilmoittaa se esimerkiksi B :lle suunnitelman 1 ja C :lle suunnitelman 0, jolloin B kuulee A :lta suunnitelman 1 ja C :ltä suunnitelman 0. Jos toisaalta A ilmoittaa sekä B :lle että C :lle suunnitelman 1 ja juonittelija onkin C , kuulee B silloinkin A :lta suunnitelman 1 ja C :ltä suunnitelman 0. B ei tiedä kumpi juonittelee eikä siten osaa päättää mitä suunnitelmaa on noudatettava. Useamman kenraalin tapauksessa kolmannes juonittelevia riittää estämään muiden vakaan päätöksenteon. Michael Fischer, Nancy Lynch ja Michael Merritt [FLM85] osoittavat, että kolmannes bysanttisia solmuja riittää estämään muiden solmujen vakaan toiminnan myös monessa muussa ongelmassa.

²Englannin kirjakeleessä sanaa *Byzantine* käytetään yleisesti merkityksessä ’ovela’, ’juonikas’.

```

(1) upon pulse
(2)   foreach  $j \in Neighbours(i)$ 
(3)     send( $j, clock_i$ )
(4)    $max \leftarrow clock_i$ 
(5)   foreach  $j \in Neighbours(i)$ 
(6)     receive( $clock_j$ )
(7)     if  $clock_j > max$ 
(8)        $max \leftarrow clock_j$ 
(9)    $clock_i \leftarrow max + 1$ 

```

Kuva 1: Vakaantuva algoritmi kellojen synkronointiin rajoittamattomin kelloarvoin laskuyksikölle P_i

2 Kellojen synkronointi

Kellojen synkronointiongelma määritellään n identtisen laskuyksikön järjestelmälle, jossa yksiköt toimivat synkronoidusti globaalin kellosykäyksen aktivoimina. Jokaisella yksiköllä on oma kokonaislukuarvoinen kellomuuttuja. Yksiköt lukevat naapuriyksiköidensä kelloarvot ja laskevat niiden avulla oman uuden kelloarvonsa. Yksiköt toimivat vakaantuvasti oikein, kun ne mielivaltaisesta lähtötilasta saavuttavat tilan, jossa kaikki kelloarvot täsmäävät ja kasvavat yhdellä jokaisella sykäyksellä.

Kuvan 1 algoritmi ratkaisee ongelman, kun $clock_i$ -muuttujien sallitaan kasvaa rajatta. Laskuyksiköt voivat olla identtiset, koska esimerkiksi poissulkemisongelmasta poiketen, jossa symmetria on rikottava, tässä vartavasten pyritään symmetriseen tila-asetelmaan. Järjestelmän vakaantumiseen tarvittavien kellosykäysten määrä on korkeintaan sen läpimitta eli kaukaisimpien solmujen etäisyys.

Rajoittamattomat kelloarvot ovat käytännön sovellusten kannalta ongelma. Vaikka esimerkiksi 64:llä bitillä ilmaistavien kokonaislukujen määrä on normaalissa toiminnassa varmasti riittävä, ei ole takeita, etteikö

```

(1) upon pulse
(2)   foreach  $j \in \text{Neighbours}(i)$ 
(3)     send( $j, \text{clock}_i$ )
(4)    $\text{min} \leftarrow \text{clock}_i$ 
(5)   foreach  $j \in \text{Neighbours}(i)$ 
(6)     receive( $\text{clock}_j$ )
(7)     if  $\text{clock}_j < \text{min}$ 
(8)        $\text{min} \leftarrow \text{clock}_j$ 
(9)    $\text{clock}_i \leftarrow (\text{min} + 1) \bmod M$ 

```

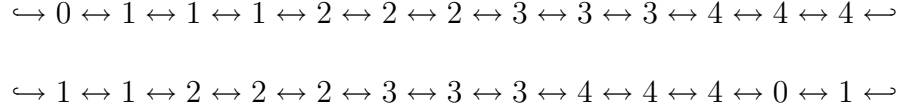
Kuva 2: Nopeahkosti vakaantuva algoritmi kellojen synkronointiin rajoitetuin kelloarvoin

järjestelmä häiriön jälkeen voisi käynnistyä tilasta, jossa kelloarvo 2^{64} hämmöttää, ja ylivuodon aiheuttama ylimääräinen ja turha toimintahäiriö uhkaa.

Huomataan, että jos kellojen ympäripyörähtäminen sallitaan, vakaantunut järjestelmä toimii ylivuodonkin sattuessa vakaasti. Halutaan taata järjestelmän vakaantumisen, kun kelloja kasvatetaan modulo M . Olkoon n verkon solmujen määrä ja d läpimitta. Kuvan 1 algoritmi, jossa kelloa rivillä 9 kasvatetaankin modulo M , on vakaantuva, kun $M > (n + 1)d$. Kelloarvojen avaruudesta löytyy nimittäin nyt millä tahansa kellojen tilojen asetelmalla pituudeltaan vähintään $d + 1$ päivitystä vastaava väli, jolle ei osu yhtään kelloarvoa. Järjestelmän toimiessa häiriöttömästi se saavuttaa joskus tilan, jossa $\max\{\text{clock}_j\} < M - d$, ja vakaantuu tähän kuluva aika mukaanlukien $O(nd)$ kellosykäyksessä.

Kuvan 2 algoritmi vakaantuu $O(d)$ kellosykäyksessä korvaamalla edellisen algoritmin maksimin minimillä. Minimiiä käyttäen $M > 2d$ riittää.

Osoitetaan, että kellojen synkronointialgoritmi, jossa laskuyksiköiden tilojen määrä on rajoitettu verkon läpimitasta riippumattomasti, ei riittävän leveille verkoille ole yleisesti vakaantuva. Tarkastellaan esimerkkinä kuvan 2 algoritmia, jossa $M = 5$. Algoritmin tiedetään vakaantuvan läpimitaltaan



Kuva 3: Liian pienen M -arvon takia vakaantumaton laskenta kolmentoista kuvan 2 algoritmia käyttävän yksikön sykklisessä verkossa

korkeintaan kahden levyisissä verkoissa. Kuva 3 esittää kaksisuuntaisesti linkitetyn kolmentoista solmun sykklisen verkon erään suorituksen kellojen tilojen kaksi peräkkäistä asetelmaa. Jälkimmäisen asetelman tilat vastaavat edellisen tiloja kierrettynä kaksi solmua vasemmalle, joten laskenta ei vakaannu. Kun laskuyksiköiden mahdollisten tilojen joukko S on kiinnitetty, vakaantumaton verkko voidaan yleisesti rakentaa valitsemalla kolme eri tilaa $s_1, s_2, s_3 \in S$, rakentamalla niiden avulla jono

$$s_1, s_2, s_3, s_4 = f(s_1, s_2, s_3), \dots, s_l = f(s_{l-3}, s_{l-2}, s_{l-1}), \dots$$

ja solmimalla jono sopivista kohdista renkaaksi. Tässä $f: S^3 \rightarrow S$ on algoritmista riippuva kaksinaapurisen laskentayksikön siirtymäfunktio. Jos yksikkö on tilassa $x \in S$ ja sen naapurit tiloissa $y, z \in S$, on $f(y, x, z) = f(z, x, y)$ yksikön seuraava tila. Jokin tilakolmikko (s_j, s_{j+1}, s_{j+2}) esiintyy jonossa kahdesti ensimmäisten $O(|S|^3)$ alkion joukossa. Solminta tehdään katkaisemalla jono kahdesti esiintyvän kolmikön alkujen osoittamista kohdista ja yhdistämällä kohtien väliin jäävän jonon viimeinen alkio ensimmäiseen. Saatua tilojen rengasta vastaava laskuyksikkörengas on vakaantumaton.

Kellojen vakaantuva synkronointi verkossa, jonka yksiköiden tilojen määrä on vakio verkon läpimitan suhteen, on mahdollista satunnaistetulla algoritmilla. Jokaisella sykäyksellä yksikkö valitsee silloin satunnaisesti, mitä kelloarvoa se käyttää omansa ja naapureidensa ilmoittamien joukosta. Koska eroavien kelloarvojen määrä kullakin sykäyksellä valinnoista riippuen joko pysyy samana tai pienenee, algoritmi vakaantuu joskus.

3 Torkahtelevat kellot

Siirrytään tutkimaan, millä edellytyksillä järjestelmä voi vakaantua yksittäisten yksiköiden virheellisestä toiminnasta huolimatta. Ensimmäiseksi tarkastellaan tilannetta, jossa virheellisesti toimivat yksiköt eivät suorastaan juonitele bysanttilaisesti, vaan pelkästään lopettavat aika ajoin kellonsa päivittämisen eli torkahtelevat. Torkahtaminen voi myös olla lopullinen kaatuminen. Mitään ei oleteta häiriön kestosta. Koska torkkuva yksikkö tyypillisesti estää sen kautta kulkevan, vakaantumiselle välttämättömän tiedonvälityksen, oletetaan, että yksiköiden verkko on täydellinen, eli kaikki solmut ovat naapureita.

Halutaan algoritmi, jota noudattamalla kukin yksikkö synkronoituu rajatussa määrässä kellosykyä kaikkien muiden vähintään yhtä pitkään oikein toimineiden yksiköiden kanssa. Synkronoiduttuaan yksikkö ei saa korjata kelloaan, vaan sen on kasvatettava sitä tasan yhdellä jokaisella sykäyksellä. Kutsutaan ehdot täyttävää algoritmia *vartoilemattomaksi* (englanniksi wait-free). Synkronoituneet kellot eivät odota synkronoituvia.

Havaitaan heti, että luvussa 2 esitetty kuvan 1 algoritmi on vartoilematon. Algoritmin vakaantumiseksi kelloarvojen on kuitenkin voitava kasvaa rajatta. Kun rajatonta kasvua ei voida taata, voi lähelle ylärajaa pysähtynyt yksikkö pakottaa muut korjaamaan kelloaan toistuvasti.

Ongelma voidaan ratkaista rajoitetuin kelloin käyttämällä apumuuttujia, joiden avulla laskuyksiköt tietävät, kumman kahdesta verrattavasta yksiköstä on epäsynkronoidussa tilanteessa korjattava kelloaan. Jokaisella yksiköllä P_i on apumuuttuja $order_{ij}$ jokaista naapuriaan P_j kohti. Yksiköt pelaavat apumuuttujien avulla erästä kivi, paperi, sakset -leikin determinististä muotoa. Päivittäessään kelloaan yksikkö P_i samalla kasvattaa modulo 3 kaikkia laskureita $order_{ij}$, jotka täsmäävät naapurien ilmoittamien $order_{ji}$ -arvojen kanssa tai ovat näistä jäljessä. Jos siis pelaajaparista kumpikin sanoo ”sakset”, kummatkin ovat kehityksessä mukana ja ollessaan hereillä päivittävät seuraavan kierroksen vastauksensa sakset voittavan kiveen. Jos taas toinen sanoo ”kivi” ja toinen ”paperi”, kiven valinnut on jäljessä ja päi-

```

(1) upon pulse
(2) in( $\{clock_j\}, \{order_{jk}\}$ )
(3)    $\mathcal{NB} \leftarrow \{j \mid \text{millään } k \text{ ei ole } (order_{jk} + 1) \bmod 3 = order_{kj}\}$ 
(4)   if  $\mathcal{NB} \neq \emptyset$ 
(5)      $clock_i \leftarrow \max_{j \in \mathcal{NB}} \{clock_j\} + 1$ 
(6)   foreach  $j \neq i$ 
(7)     if  $order_{ij} \neq (order_{ji} + 1) \bmod 3$ 
(8)        $order_{ij} \leftarrow (order_{ij} + 1) \bmod 3$ 
(9) out( $clock_i, \{order_{ij}\}$ )

```

Kuva 4: Vartoilematta vakaantuva algoritmi kellojen synkronointiin rajoitetuin kelloarvoin yksikölle P_i

vittää herättyään vastauksensa paperiin. Kelloarvon päivityksessä uusi kelloarvo lasketaan kivi, paperi, sakset -peleissä kaikkien yksiköiden suhteen ajan tasalla olevien yksiköiden maksimista.

Dolev selostaa algoritmin ainoastaan sanallisesti. Kuvassa 4 annetaan sen pseudokoodihahmotelma. Algoritmi vakaantuu kahdessa sykäyksessä. Sen oikeellisuudesta vakuuttautumiseksi on huomattava, että kullakin sykäyksellä ja erityisesti vakaantumisen ensimmäisen sykäyksen aikana kaikki hereillä olevat yksiköt näkevät samat $clock_j$ - ja $order_{jk}$ -arvot. Ne laskevat kaikki saman ajan tasalla olevien yksiköiden joukon \mathcal{NB} , ja valitsevat saman kellomaksimin tästä joukosta, jos se ei ole tyhjä. Vakaantumisen ensimmäisen sykäyksen aikana torkkuva yksikkö voi sisältyä joukkoon \mathcal{NB} , mutta toisella sykäyksellä laskettava joukko sisältää vain kaikki edellisellä sykäyksellä toimineet yksiköt.

4 Bysanttilaiset häirikkökellot

Edellisessä luvussa häiriökäyttäytyminen rajoittui häiriintyneiden yksiköiden täydelliseen toimimattomuuteen, ja toimivat yksiköt näkivät samat arvot.

Jos kommunikointi perustuu yksiköiden keskinäiseen viestinvälitykseen, voi bysanttilaisesti häiriköivä yksikkö ilmoittaa ristiriitaisia tietoja naapurisolmuilleen ja vaikeuttaa huomattavasti oikein toimivien yksiköiden laskennan vakaantumista. Halutaan kellojensynkronointialgoritmi, joka vakaantuu, kun alle kaksi kolmannesta yksiköistä toimii väärin. Oletetaan täydellinen verkko.

Olkoon n laskuyksiköiden määrä ja $f < n/3$ algoritmin sietämä virheelisten yksiköitten maksimimäärä. Käytetään kahta perussääntöä uuden kelloarvon määräämiseen. Jos oma kello mukaanlukien vähintään $n - f$ kelloa täsmää yksikön omaan kelloon, kasvatetaan normaalisti kelloa yhdellä modulo M . Muussa tapauksessa kello nollataan.

Tarkastellaan neljän yksikön järjestelmää, jossa on yksi bysanttilaisesti juonitteleva yksikkö. Oikein toimivien yksiköiden P_1 , P_2 ja P_3 kellot ovat järjestyksessä aluksi 0, 0 ja 1. Juonitteleva P_4 ilmoittaa P_1 :lle arvon 0 ja muille 1. Toimivat yksiköt päivittävät arvoikseen 1, 0 ja 0. Seuraavalla sykäyksellä P_4 ilmoittaa P_3 :lle arvon 0 ja muille 1, jolloin toimivat yksiköt päivittävät arvonsa alkutilan mukaiseksi. Havaitaan, että perussäännöt eivät aivan riitä vakaantumiseen.

Tarkistetaan kasvatussääntöä. Otetaan käyttöön apumuuttuja, jonka avulla tiedetään, onko 0-arvoon päädytty kasvatussääntöä vai nollaussääntöä käyttämällä. Jos kello on kasvanut nolnaan ja $n - f$ kelloarvoa täsmää nolnaan, on kasvattamista jatkettava, koska yksikön käytössä olevien tietojen mukaan laskenta on voinut olla vakaata edellisessä sykäyksessä. Jos kuitenkin on käytetty nollaussääntöä ja $n - f$ kelloa täsmää, olisi arvattava, onko riittävän monen oikein toimivan yksikön kello oikeasti 0. Kuvan 5 satunnaisesti algoritmista tehdään juuri näin eli arvotaan kasvatetaanko nollattua kelloa vai ei.

Oikein toimivien yksiköiden $LastIncrement_i$ -arvot vakaantuvat yhdessä sykäyksessä osoittamaan, sovellettiinko edellisessä sykäyksessä kasvatus- vai nollaussääntöä. Tämän jälkeen algoritmi vakaantuu odotusarvoisesti $2^{2(n-f)}$. M sykäyksessä, kun vähintään $n - f$ yksikköä toimii tämän ajan oikein.

```

(1) upon pulse
(2)   foreach  $j \in Neighbours(i)$ 
(3)     send( $j, clock_i$ )
(4)   foreach  $j \in Neighbours(i)$ 
(5)     receive( $clock_j$ ) (* aikarajoitettu *)
(6)   if  $|\{j \mid clock_i = clock_j\}| < n - f$ 
(7)      $clock_i \leftarrow 0$ 
(8)      $LastIncrement_i \leftarrow false$ 
(9)   else if  $clock_i \neq 0$ 
(10)     $clock_i \leftarrow (clock_i + 1) \bmod M$ 
(11)     $LastIncrement_i \leftarrow true$ 
(12)  else if  $LastIncrement_i$ 
(13)     $clock_i \leftarrow 1$ 
(14)  else
(15)     $clock_i \leftarrow random\{0, 1\}$ 
(16)    if  $clock_i = 1$ 
(17)       $LastIncrement_i \leftarrow true$ 

```

Kuva 5: Vakaantuva algoritmi bysanttilaisten kellojen synkronointiin rajoitetuin kelloarvoin yksikölle P_i

Toimiva yksikkö P_i saavuttaa M sykäyksen sisällä tilan, jossa $clock_i = 0$, ja yksikkö näkee $n - f$ eri $clock_j = 0$ -arvoa. Jos oikein toimivia $clock_j = 0$ -yksiköitä on oikeasti vähintään $n - f$, tarvitaan korkeintaan $n - f$ onnekasta valintaa, että $n - f$ oikein toimivaa yksikköä kasvattaa kelloaan. Muutoin alle $n - f$ onnekasta valintaa riittää pitämään $clock_j = 0$:t nolliina, jolloin seuraavalla sykäyksellä nollakelloja on varmasti riittävästi ja $n - f$ onnekkaan valinnan seurauksena $n - f$ toimivaa yksikköä kasvattaa kelloaan. Koska tilaisuuksia vakaantumiseen tarjoutuu ainakin M sykäyksen välein ja jälkimmäisessä vaikeammassa tapauksessa suotuisten valintojen todennäköisyys on $> (1/2^{n-f})^2$, odotusarvoisesti $2^{2(n-f)}M$ sykäystä riittää vakaantumiseen.

Kuvan 5 algoritmi saattaa olla käyttökelpoinen pienillä n ja M . Neljän yksikön järjestelmä vakaantuu odotusarvoisesti $64M$ sykäyksessä. Yksi yksikkö saa olla viallinen. Algoritmin avulla voidaan kuitenkin muodostaa parannettu algoritmi, joka vakaantuu järjellisessä ajassa isoillakin M -arvoilla. Idea on käyttää kuvan 5 algoritmia rinnakkain useaan, pienin M -arvoin rajattuun kelloon. Yksiköt P_i päivittävät yhden $clock_i$ -kelloarvon sijasta useaa toisistaan riippumatonta, pienin eri alkuluvuin rajoitettua muuttujaa $clock_{i1}, clock_{i2}, \dots, clock_{ir}$. Nämä voidaan kuvata yhdeksi kellomuuttujaksi $clock_i$, joka kasvaa yhdellä modulo alkulukujen tulo, kun kutakin $clock_{i_l}$:ää kasvatetaan omien alkulukumoduloaritmetiikkojensa suhteen. Uusi algoritmi vakaantuu kaikkien toimivien yksiköiden P_i kellojen $clock_{i_l}$ suhteen odotusarvoisesti ajassa $2^{2(n-f)}$ kertaa alkulukujen summa. Jos valitaan esimerkiksi $M = 2 \cdot 3 \cdot 5 = 30$, rinnakkaisia kelloja käyttävä neljän yksikön järjestelmä vakaantuu odotusarvoisesti $64 \cdot (2 + 3 + 5) = 640$ sykäyksessä ja rinnakkaistamaton $64 \cdot 30 = 1920$ sykäyksessä. Kuva 6 havainnollistaa kellojen päivittymistä laskennan vakaannuttua tällaisessa järjestelmässä.

5 Yhteenveto

Tarkastellut esimerkit osoittavat, että sopivin oletuksin vakaantuvien algoritmien vikasietoisuusominaisuudet voidaan ulottaa takaamaan toimivien

$(0, 0, 0) \sim 0 \rightarrow$	$(0, 1, 0) \sim 10 \rightarrow$	$(0, 2, 0) \sim 20 \rightarrow$
$(1, 1, 1) \sim 1 \rightarrow$	$(1, 2, 1) \sim 11 \rightarrow$	$(1, 0, 1) \sim 21 \rightarrow$
$(0, 2, 2) \sim 2 \rightarrow$	$(0, 0, 2) \sim 12 \rightarrow$	$(0, 1, 2) \sim 22 \rightarrow$
$(1, 0, 3) \sim 3 \rightarrow$	$(1, 1, 3) \sim 13 \rightarrow$	$(1, 2, 3) \sim 23 \rightarrow$
$(0, 1, 4) \sim 4 \rightarrow$	$(0, 2, 4) \sim 14 \rightarrow$	$(0, 0, 4) \sim 24 \rightarrow$
$(1, 2, 0) \sim 5 \rightarrow$	$(1, 0, 0) \sim 15 \rightarrow$	$(1, 1, 0) \sim 25 \rightarrow$
$(0, 0, 1) \sim 6 \rightarrow$	$(0, 1, 1) \sim 16 \rightarrow$	$(0, 2, 1) \sim 26 \rightarrow$
$(1, 1, 2) \sim 7 \rightarrow$	$(1, 2, 2) \sim 17 \rightarrow$	$(1, 0, 2) \sim 27 \rightarrow$
$(0, 2, 3) \sim 8 \rightarrow$	$(0, 0, 3) \sim 18 \rightarrow$	$(0, 1, 3) \sim 28 \rightarrow$
$(1, 0, 4) \sim 9 \rightarrow$	$(1, 1, 4) \sim 19 \rightarrow$	$(1, 2, 4) \sim 29 \rightarrow$

Kuva 6: Vakaa kellon päivitys kolmen rinnakkaisen kellon avulla koostetussa järjestelmässä

laskuyksiköiden laskennan vakaantuminen ja vakaana pysyminen osan järjestelmästä toimiessa väärin. Tehdyt oletukset ovat kuitenkin vahvoja ja usein käytännön tilanteissa epärealistisia. Erityisesti kaikissa esimerkeissä on oletettu laskennan toimivan synkronoidusti. Asynkronisessa järjestelmässä vakaantuminen häiriöiden kestäessä muuttuu helposti mahdottomaksi. Häiriöiden kestäessä vakaantuvia esimerkkialgoritmeja tarkasteltaessa on oletettu täydellinen verkko. Verkon kahden osan yhdistävän linkkisolmun kaatuminen tekee luonnollisesti kommunikaatiota vaativan vakaantumisen mahdottomaksi.

Viitteet

- [Dij73] Edsger W. Dijkstra. Self-stabilization in spite of distributed control. Kopioina levitetty käsikirjoitus EWD 391, julkaistu painettuna 1982 [Dij82], WWW:ssä <http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD391.PDF>, lokakuu 1973.

- [Dij82] Edsger W. Dijkstra. Self-stabilization in spite of distributed control. Teoksessa *Selected Writings on Computing: A Personal Perspective*, sivut 41–46. Springer-Verlag, New York, New York, 1982. Kirjoitettu 1973 [Dij73].
- [Do100] Shlomi Dolev. *Self-Stabilization*. MIT Press, Cambridge, Massachusetts, 2000.
- [FLM85] Michael J. Fischer, Nancy A. Lynch ja Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5–7, 1985*, sivut 59–70, 1985. Julkaistu uudestaan 1986 [FLM86].
- [FLM86] Michael J. Fischer, Nancy A. Lynch ja Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, maaliskuu 1986. Ilmestynyt alunperin 1985 [FLM85].
- [LSP82] Leslie Lamport, Robert Shostak ja Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, heinäkuu 1982.