

Hyväksymispäivä

arvosana

arvostelija

Itsestabiloivat algoritmit: Paikallinen stabilointi

Aila Koponen

Helsinki, lokakuu 2007

Seminaarityö (Hajautetut Algoritmit)

Helsingin Yliopisto

Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion - Faculty		Laitos – Institution - Department	
Luonnontieteellinen tiedekunta		Tietojenkäsittelytieteen laitos	
Tekijä – Författare - Author			
Koponen, Aila Helena			
Työn nimi – Arbetets titel - Title			
Itsestabiloivat algoritmit: Paikallinen stabilointi			
Oppiaine – Läroämne – Subject			
Tietojenkäsittelytiede			
Työn laji – Arbetets art – Level		Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Seminaarityö		Lokakuu 2007	14 sivua
Tiivistelmä – Referat			
<p>Hajautetun järjestelmän tilasiirtymät eivät ole aina järkevässä mittakaavassa deterministisesti pääteltävissä systeemin hetkellisestä tilasta lähtien. Esimerkiksi tapahtumien (events) vastaanottojärjestys sekä ympäristön suoritusaskeleet, esim. viestin tms. hävittäminen, vaikuttavat prosessoreiden (ks. haj.järj. mallinnus/yleistys) tilasiirtymiin muuttaen niiden ennalta arvattavissa olevaa käyttäytymistä. Mallinnuksessa tämä saattaa aiheuttaa "tilarajähdyksen". Itsestabiloituva järjestelmä voi aloittaa mistä tahansa alkutilasta ja päättyä lopulta haluttuun kelvolliseen käyttäytymiseen. Koska itsestabiloivat algoritmit ovat luonteeltaan päättymättömiä (non-terminating), ne sopivat hyvin hajautettuihin järjestelmiin ja protokollaongelmien ratkaisuihin.</p> <p>Paikallista itsestabiloivaa algoritmia, joka palauttaa turvallisen konfiguraation hallitulla (graceful) tavalla rajatun virhemäärän sattuessa, kuvataan termeillä virheitä eristävä (fault containment), aikamukautuva (time-adaptive) ja superstabiloiva (superstabilizing). Seminaarityön tarkoituksena on kuvata näitä itsestabiloivien algoritmien ominaisuuksia.</p> <p>ACM Computing Classification System (CCS): A.1 [Introductory and Survey], C.2.4 [Distributed Systems], C.4 [Performance of Systems], G.4 [Mathematical Software]</p>			
Avainsanat – Nyckelord – Keywords			
Hajautetut algoritmit, itsestabiloiva, superstabiloiva			
Säilytyspaikka – Förvaringställe – Where deposited			
Muita tietoja – Övriga uppgifter – Additional information			

Sisältö

1	JOHDANTO.....	1
2	PAIKALLISTEN ALGORITMIEN KÄSITTEITÄ	2
3	SUPERSTABILOIVUUS.....	4
4	VIRHEITÄ ERISTÄVÄT ITSESTABILOIVAT ALGORITMIT.....	8
5	VIRHEENHAKUKOODIT JA KORJAUS	11
6	YHTEENVETO	13
7	VIITTEET	14

1 Johdanto

Itsestabiloivien algoritmien yksi haluttava ominaisuus on paikallisuus. On toivottavaa, että pieni määrä paikallisia virhetilanteita ei johtaisi laajamittaiseen toipumisalgoritmeista johtuvaan häiriötilaan. Tällöin verkko saattaa käyttäytyä epätoivotulla tavalla sekä ennalta arvaamattomasti. Pyrkimyksenä on, että paikalliset virheet eivät sotke tai jopa sulje koko verkkoa. Paikallisten itsestabiloivien algoritmien tavoitteena on palauttaa turvallinen konfiguraatio hallitulla (graceful) tavalla riittävän lyhyessä ajassa rajatun virhemäärän sattuessa.

Työn tavoitteena on käsitellä paikallisia stabiloivia algoritmeja ja niihin liittyviä käsitteitä kirjan ”Self-Stabilization” [Dol00] ja pääosin sen luvun ”Local Stabilization” mukaan.

Luvussa 2 ”Paikallisten algoritmien käsitteitä” esitellään lyhyesti luvuissa 3, 4 ja 5 käytettäviä käsitteitä. Luvussa 3 ”Superstabiloivuus” käsitellään superstabiloivuuden perusominaisuuksia sekä muunnetaan luvussa esitelty itsestabiloiva väritysalgoritmi superstabiloivaksi. Luvussa 4 ”Virheitä estävät itsestabiloivat algoritmit” eritellään käytetty virhemalli (fault model) ja lisätään esitellyn synkronisen algoritmin virheideneristävyyttä. Luku 5 ”Virheenhakukoodit ja korjaus” esittelee virheenhakukoodeja sekä pyramididatarakennetta käyttävän korjausprosessin sekä tarkentaa mihin virhemalliin ne sopivat. Viimeisessä luvussa suoritetaan yhteenveto.

2 Paikallisten algoritmien käsitteitä

Esitellään seuraaviin lukuihin käsitteitä - hajautetun järjestelmän pelkistetty malli viestinvälitysmallissa ja jaetun muistin mallissa, järjestelmän konfiguraatio ja turvallinen konfiguraatio itsestabiloivien algoritmien tapauksessa, lomitukset, asynkroninen kierros, asynkroninen sykli, porttiehto, pyramididatarakenne, superstabilisuus, virheeneristysalgoritmi ja virheenhakukoodi.

Hajautetun järjestelmän pelkistetty malli on n kappaleen joukko tilakoneita eli prosessoreita P_i jotka kommunikoivat keskenään. Viestinvälitysmallissa (message passing distributed system) kommunikointia mallinnetaan FIFO-viestijonoilla $q_{i,j}$ jotka sisältävät prosessorilta P_i prosessorille P_j lähetetyt mutta vielä vastaanottamattomat viestit. Jaetun muistin mallissa (shared memory distributed system) tiedonvälitys käy prosessoreille yhteisten rekisterien r kautta.

Järjestelmän **konfiguraatio** viestinvälitysmallissa on $c = (s_1, s_2, \dots, s_n, q_{1,2}, q_{1,3}, \dots, q_{i,j}, \dots, q_{n,n-1})$ missä s_i on välillä $1 \leq i \leq n$ on prosessorin P_i tila ja $q_{i,j}$ kun $i \neq j$ on P_i :n P_j :lle lähettämien viestien jono. Konfiguraation määritelmä jaetun muistin mallissa on $c = (s_1, s_2, \dots, s_n, r_1, r_2, \dots, r_m)$ jossa s_i välillä $1 \leq i \leq n$ on prosessorin P_i tila ja r_m välillä $1 \leq j \leq m$ on yhteysrekisterin sisältö [Dol00, s.6]. Järjestelmän konfiguraatio on **turvallinen** suhteessa tehtävään (task) sekä algoritmiin, jos jokainen tästä konfiguraatiosta alkanut suoritus kuuluu tehtävään. Algoritmi on tehtävälle itsestabiloiva, jos jokainen algoritmin reilu suoritus saavuttaa edellä mainitun turvallisen konfiguraation suhteessa tähän tehtävään [Dol00, s. 9].

Lomitus tarkoittaa, että annettuna ajanhetkenä vain yksi prosessori suorittaa laskenta-askeleen a eli tässä mallissa koosteaskeleen (aggregate step) tai atomisen suorituksen (computational step, atomic step) eli tästälähtien lyhyesti askeleen, joka koostuu sisäisestä laskennasta ja yhdestä viestitysoperaatiosta joka voi olla joko luku tai kirjoitus rekisteriin tai viestijonoon. [Dol00, s. 7].

Asynkroninen kierros on pienin laskennan (execution) osasuoritus jossa jokainen prosessori saa suorittaa ainakin yhden askeleen [Dol00, s. 9].

Asynkroninen sykli on pienin laskennan osasuoritus, jossa jokainen prosessori saa suorittaa ainakin yhden ”do forever”-silmukan [Dol00, ss. 10-11].

Porttiehto (passage predicate) on vahvin ehto joka pitää, kun laillinen (legitimate) tila läpikäy topologiamuutoksen [Dol00, s. 159].

Pyramiditilatietorakenne prosessorille P_i [Dol00, sivut 123-133] $\Delta_i = V_i[0], V_i[1], V_i[2], \dots, V_i[d]$ koostuu ”snapshot”-näkyistä $V_i[l]$ missä l -säteinen näkymä kuvaa konfiguraation topologian määrittelemällä alueella eli k.o. prosessorista lähtien l :n prosessorin päähän. Näkymän muodostava algoritmi kerää, häiritsemättä muuta toimintaa, prosessoreiden tilat ja viestipuskureiden sisällöt. Erityisesti pyramidi Δ_i sisältää P_i -keskiset näkymät $V_i[l]$ joissa näkymään kuuluvan prosessorin etäisyys keskustasta on korkeintaan l ja ”snapshot”-näkymä on otettu l aikayksikköä sitten. Pyramidin pohjalla on näkymä koko järjestelmästä, jonka säde on siis d .

Algoritmi on **superstabiloiva** jos se on stabiloiva ja jos turvallisesta konfiguraatiosta lähtien topologiamuutoksen sattuessa algoritmi pysyy palauttaamaan hallitusti (gracefully) uuden turvallisen konfiguraation. Lisäksi algoritmin on suoririuduttava em. uuteen konfiguraatioon muuntautumisesta ripeästi ja säilytettävä samalla porttiehto.

Virheeneristysalgoritmi (fault-containing algorithm) pyrkii, lähtien turvallisesta konfiguraatiosta, palauttamaan turvallisen konfiguraation $O(f)$ syklissä, kun f :n prosessorin tilat ovat korruptoituneet tilapäisten vikojen (transient fault) seurauksena.

Virheenhakukoodi laskettuna yksittäisen prosessorin nykytilasta talletetaan prosessorikohtaisesti. Kun aloitetaan suoritus, tarkistetaan sopiiko nykytila viimeeksi talletettuun virheenhakukoodiin. Jos sopii, suoritusta voidaan jatkaa ja päätteksi tilanvaihdon yhteydessä koodi lasketaan ja edellinen koodi korvataan.

3 Superstabiloivuus

Tässä luvussa esitellään itsestabiloiva solmunväritysalgoritmi ja siitä muunnettu superstabiloiva versio sekä muuntaminen. Lisäksi havainnollistetaan termit porttiehto, superstabilointiaika (superstabilizing time) ja muutosarvo (adjustment measure).

Virhemalli ja määritelmä. Superstabiloivat algoritmit perivät ominaisuuksia dynaamisista sekä itsestabiloivista algoritmeista. Dynaamiset järjestelmät toipuvat tavanomaisesti kommunikaatiolinkkien sekä prosessoreiden aiheuttamista topologiamuutoksista paikallisesti. Virhemalli (failure model) olettaa että vikoja sattuu vain rajallinen määrä ja dynaaminen algoritmi selviää niistä mahdollisimman vähillä muutoksilla. Itsestabiloiva algoritmi takaa, että aloittamalla mistä tahansa konfiguraatiosta tai tilasta järjestelmä alkaa jossain vaiheessa käyttäytyä laillisesti mutta mitään takeita eikä myöskään rajoitteita virhemallin rajoissa anneta käyttäytymisestä muutosprosessin aikana. Yhdistämällä nämä kaksi tyyppiä saadaan superstabiloivia algoritmeja jotka saavuttavat mistä tahansa tilasta turvallisen konfiguraation nopeasti ja hallitusti säilyttäen porttiehdon. Superstabiloivat algoritmit ovat määritelmän mukaan aina myös itsestabiloivia.

Superstabiloivuuden määreitä. Superstabiloivia algoritmien vaativuutta kuvaillaan määreillä porttiehto, superstabilointiaika (superstabilizing time) ja muutosarvo (adjustment measure). Koska topologiamuutos yleensä haastaa laillisuuden, porttiehto määritelläänkin suhteessa topologiamuutostyypeihin. Sen on oltava riittävän voimakas mutta heikompi kuin laillisuusehto – se on vahvin predikaatti joka on voimassa laillisesta tilasta seuranneen topologiamuutoksen jälkeen. Esimerkiksi ehto ”vain yksi vuoromerkki” (”only one token”) ei vaaranna kun prosessorin kaatuminen voi hävittää vuoromerkkin mutta ei toisintaa sitä. Superstabilointiaika määrittää algoritmin suorituskierrokset joka kuluvat turvallisesta konfiguraatiosta taas turvalliseen konfiguraatioon siirryttäessä ja muutosarvo samassa kuluksa tilaansa muuttavien prosessorien määrän kun välissä tapahtuu tasan yksi topologiamuutos.

Itsestabiloiva solmunväritysalgoritmi. Tässä Käytettävän itsestabiloivan solmunväritysalgoritmin ”do forever” –luoppi on kuvassa 1 ”**Itsestabiloiva väritysalgoritmi prosessori P_i lle**”.

Solmunväritysalgoritmin ideana on värittää solmut eli prosessorit niin, että yksikään ei ole saman värinen kuin naapurinsa. Itsestabiloivuus takaisi loppujen lopuksi tämän, sekä sen, että kaikki värit

ovat samat kuin mitä ne olivat ennen ”katastrofin” sattumista laillisessa tilassa. Algoritmissä käytetään jaetun muistin mallia naapuriprosessoreiden välillä, mutta ei oteta kantaa siihen, miten värien määrän voisi minimoida vaan oletetaan että värejä on $\Delta + 1$ jossa Δ on yksittäisen solmun naapureiden maksimimäärä.

```

01  do forever
02      GColors :=  $\emptyset$ 
03      for m := 1 to  $\delta$  do
04          lrm := read (rm)
05          If ID(m) > I then GColors := GColors  $\cup$  lrm.color
06      Od
07      if colori  $\in$  GColors then
08          colori := choose( $\setminus$  GColors)
09      Write ri.color := color
10  Od

```

Kuva 1 Itsestabiloiva väritysalgoritmi prosessori P_i :lle

Luuppi prosessorille P_i aloitetaan resetoimalla prosessorin omista muuttujista sen naapureiden värit (*GColors*). Seuraavaksi luetaan kaikkien naapurien rekisterit mutta kerätään värit vain niistä joilla on suurempi prosessoritunniste (ID) (rivit 3-6). Prosessori valitsee uuden värin vain, jos sen käyttämä väri on edellä kerätyssä joukossa (rivit 7-9).

Vaikka dynaamisen muutoksen sattuminen säilyttäisi ehdon ”naapureita korkeintaan Δ kappaletta” voi käydä niin että erivärisyysehto ei toteudu. Tällöin voi pahimmassa tapauksessa käydä niin että turvalliseen konfiguraatioon kyllä päästään mutta liki kaikki prosessorit muuttavat matkalla väriänsä. Näin kävisi esimerkiksi prosessoriketjussa ($\Delta = 2$) jossa viimeinen yhteyskatkosta toipumisen jälkeen olisikin saanut naapurinsa värin ja prosessoritunnisteet koko ketjussa olisivat suurenevassa järjestyksessä.

Muunnos superstabiloivaksi. Muunnetaan edellä käsitellystä itsestabiloivasta solmunväritysalgoritmista superstabiloiva versio jonka ”do forever” –luuppi ja keskeytyksäsittely ovat kuvassa 2 ”**Superstabiloiva väritysalgoritmi prosessori P_i :lle**”. Muutokset näkyvät kuvassa valkoisella pohjalla.

```

01  do forever
02      AColors :=  $\emptyset$ 
03      GColors :=  $\emptyset$ 
04      for m := 1 to  $\delta$  do
05          lrm := read (rm)
06          AColors := AColors  $\cup$  lrm.color
07          If ID(m) > I then GColors := GColors  $\cup$  lrm.color
08      Od
09      if colori =  $\perp$  or colori  $\in$  GColors then
10          colori := choose( $\setminus$  AColors)
11      Write ri.color := color
12  Od
13  interrupt section
14      if recoverij and j > i then
15          colori :=  $\perp$ 
16          write ri.color :=  $\perp$ 

```

Kuva 2 Superstabiloiva väritysalgoritmi prosessori P_i :lle.

Itsestabiloivan algorimin esittelyn lopussa kuvatun ongelmatilanteen varalta ketjun viimeisen ja toiseksiviimeisen prosessorin täytyy käsitellä topologiamuutosta erityismerkkinä jotta ne voisivat suunnitella värinvaihtoa harkiten. Superstabiloivan algoritmin tapauksessa prosessoreiden P_i ja P_j välisen toipuvan linkin päät saavat signaalin *recover_{ij}*. Tällöin keskeytyskäsitelyssä topologiamuutoksen merkiksi pienemmän tunnisteiden omaava prosessori ottaa väriksen \perp , joka ei kuulu verkon normaaliin väripalettiin (rivit 13-16).

Luuppi prosessorille P_i aloitetaan kuten itsestabiloivassakin versiossa alustamalla prosessorin omista muuttujista sen naapureiden värit (*GColors*) joilla on suurempi prosessoritunniste. Nyt lisäksi alustetaan myös uusi muuttuja johon on talletettu *kaikkien* k.o. prosessorin naapureiden värit (*AColors*). Seuraavaksi luetaan kaikkien naapureiden rekisterit ja populoidaan molemmat edellä mainitut joukot (rivit 4-8). Nyt prosessori valitsee uuden värin vain, jos sen käyttämä väri on suurempi-identiteettisten prosessoreiden värien joukossa *GColors* tai topologiamuutos on tapahtunut (rivit 9-11). Lisäksi väri on valittava niin, että se ei ole minkään sen naapureiden käyttämien värien joukossa *AColors*. Nyt on huomattava, että itsestabiloivan algoritmin kuvauksessa esimerkkinä mainitun prosessoriketjun ongelmatilanteessa väriä muuttaisi vain toiseksiviimeinen prosessori koska se osaisi valita värinsä niin että ”värinvaihtoaalto” ei lähtisi liikkelle ja turvallinen konfiguraatio säilyisi.

Tämä algoritmi säilyttää itsestabiloivuutensa. Jos topologiamuutosta ei ole tapahtunut, ensimmäisen asynkronisen syklin jälkeen jokaisen prosessorin ja prosessorirekisterin värit kuuluvat väripalettiin eikä keskeytyskäsitelyä tarvita. Lisäksi ensimmäisen asynkronisen syklin jälkeen suurimman tunnisteensa omaavan prosessorin väri on kiinteä siten että se ei seuraavilla sykleillä enää vaihdu. Tästä siirryttäessä pienempään päin tunnistenumeroissa värit kiinteistyvät aina yksi prosessori syklistä joten turvalliseen konfiguraatioon päästään.

4 Virheitä eristävät itsestabiloivat algoritmit

Luvussa esitellään synkroninen algoritmi joka tuottaa kiinteän tuloksen ja samaan tehtävänasetteluun muodostettu itsestabiloiva virheitä eristävä algoritmi sekä virheen korjausta.

Topologiamuutokset dynaamisessa järjestelmässä voivat vaatia yhden tai jopa jokaisen järjestelmän prosessorin vaihtamaan tilansa riippumatta muutoksen tapahtumapaikasta tai sen fyysisestä etäisyydestä. Esimerkkinä tästä voisi olla turvallisen konfiguraation renkaan muodostava juurellinen viritävä puu (rooted spanning-tree) jossa juuressa sijaitseva prosessori keskustelee sekä renkaan seuraavaan että viimeisen prosessorin kanssa. Kaikki solmut ovat siis valinneet itselleen vanhemman naapureistaan ja määritelmän mukaan yksi kaari ei ole merkinnyt itseään puun kaareksi. Nyt jos toinen juureen liittyvistä kaarista vikaantuu rikkoen renkaan yhteysverkon, noin puolet prosessoreista vaihtaa saman tien vanhempansa.

Virhemalli ja määritelmä. Edellä kuvattuun vakavaan virhetilanteeseen verrattuna tässä luvussa käytetään mallia jossa oletetaan tapahtuvaksi vain tilapäisiä vikoja (transient faults). Tällöin konfiguraation voi korjata turvallisesti f :n tilapäisen vian jälkeen palauttamalla f :n viasta kärsineen prosessorin tilat ennalleen. Itsestabiloiva virheitä eristävä algoritmi pyrkii, lähtien turvallisesta konfiguraatiosta, palauttamaan turvallisen konfiguraation $O(f)$ syklissä, kun f :n prosessorin tilat ovat korruptoituneet väliaikaisten vikojen seurauksena. Lisäksi vaaditaan, että mistä tahansa konfiguraatiosta lähtien voidaan saavuttaa turvallinen konfiguraatio.

Itsestabiloiva päivitysalgoritmi. Muunnettavana algoritmina käytetään päivitysalgoritmia [DoI00, luku 4.4]. Algoritmin päätehtävä on tulvittaa kyseisen prosessorin ympäristö yhtenäisen komponenttinsa (connected component) prosessorien tunnuksilla. Liitoskomponenttiin kuuluvat kaikki prosessorit jotka voivat otta suoraan tai välillisesti kommunikoida. Tulvituksen tuloksena järjestelmän prosessorit tietävät suurimman tunnuksen ja pystyvät näin yksiselitteisesti valitsemaan johtajan. Kuvassa 3 ”**Kiinteän tuloksen tuottava synkroninen algoritmi prosessori P_i :lle**” näkyy kohteena olevan itsestabiloivan kiinteän syötteen tarvitsevan algoritmin pulssin aikana suoritettava ohjelma.

```

01  upon a pulse
02    ReadSeti := ∅
03    forall Pj ∈ N(i) do
04      ReadSeti := ReadSeti ∪ read(Processorsj)
05      ReadSeti := ReadSeti \\i := ReadSeti ++ <*, 1, *>
07      ReadSeti := ReadSeti ∪ {<i, 0, Ii>}
08    forall Pj ∈ processors(ReadSeti) do
09      ReadSeti := ReadSeti \\j, ReadSeti)>
10    write Processorsi := ConPrefix(ReadSeti)
11    write Oi := ComputeOutputi(Inputs(Processorsi))

```

Kuva 3 Kiinteän tuloksen tuottava synkroninen algoritmi prosessori P_i:lle

Jokainen prosessori P_i lukee monikot $\langle id, dist, I_{id} \rangle$, missä id on prosessorin identiteetti, $dist$ on etäisyys P_i:stä ja I_{id} on prosessorin id kiinteä syöte, naapureidensa *Processors*-muuttujista (rivit 3 ja 4) ja poistaa sitten monikkojoukosta (*ReadSet_i*) itseensä viittaavat. Sen jälkeen muiden luettujen monikoiden etäisyyskenttiä kasvatetaan yhdellä ja lisätään prosessoriin itseensä viittava monikko takaisin etäisyytenään nolla. Tässä korjataan muista prosessoreista luetut etäisyydet sopiviksi prosessorille P_i. Seuraavaksi (rivit 8 ja 9) siivotaan joukosta monikot joita on useampi eri etäisyyksillä ja jätetään näistä aina pienimmällä etäisyydellä varustettu.

ConPrefix(ReadSet_i) poistaa monikkojoukosta kaikki monikot joilla on etäisyytenä suurempi luku kuin ensimmäinen monikkojoukosta puuttuva etäisyyysluku. Pois joutavat siis sellaiset, joita ei voi saavuttaa niiden lähimmästä prosessorista. Nyt voidaan lopuksi laskea prosessorijoukon syötearvoista osatuloste O_i prosessorille P_i.

Jos prosessoreiden tilat muuttuvat mutta syötteet eivät, on algoritmilla virheitä eristävä ominaisuus koska tilamuutosten jälkeisen turvallisen konfiguraation tuloste on pakostakin sama kun ennen virhetilannetta. Jos taas syötteet ovat muuttuneet, usea prosessori ehtii tallentaa väärää tietoa tulostemuuttujaansa ennen kuin tilanne stabiloituu.

Itsestabiloiva virheitä eristävä algoritmi. Muunnettu algoritmi näkyy kuvassa 4 ”Itsestabiloiva virheitä eristävä algoritmi prosessori P_i:lle”. Ensimmäiseksi edellisen algoritmin monikoihin ympättiin kenttä A_i joka sisältää *Processors_i*:n monikoiden syötekenttien arvot. Seuraavana prosessoreille lisättiin korjauslaskurit *RepairCounter* jotka laskevat etäisyyttä siitä asti kun

vikatilanne huomattiin. Alkaen turvallisesta konfiguraatiosta tilapäisten vikojen sattua yksikään prosessori ei vaihda A_i :n arvoa $d+1$ pulssiin. Korjausprosessi aloitetaan nollaamalla korjauslaskuri (rivi 17).

```

01  upon a pulse
02      ReadSeti := ∅
03      forall Pj ∈ N(i) do
04          ReadSeti := ReadSeti ∪ read(Processorsj)
05          If RepairCounteri ≠ d + 1 then
06              RepairCounteri := min(RepairCounteri, read(RepairCounterj))
07      Od
08      ReadSeti := ReadSeti \\ <i, *, *, *>
09      ReadSeti := ReadSeti ++ <*, 1, *, *>
10      ReadSeti := ReadSeti ∪ {<i, 0, Ii, Ai>}
11      forall Pj ∈ processors(ReadSeti) do
12          ReadSeti := ReadSeti \\ NotMinDist(Pj, ReadSeti)
13      write Processorsi := ConPrefix(ReadSeti)
14      if RepairCounteri = d + 1 then
15          if (Oi ≠ ComputeOutputi(Inputs(Processorsi))) or
16              ((exists) <*, *, *, A> ∈ Processorsi | A ≠ Inputs(Processorsi)) then
17              RepairCounteri := 0
18      Else
19          RepairCounteri := min(RepairCounteri + 1, d + 1)
20          Write Oi := ComputeOutputi(RepairCounteri,
21              MajorityInputs(Processorsi))
22          if RepairCounteri = d + 1 then
23              Ai := Inputs(Processorsi)

```

Kuva 4 Itsestabiloiva virheitä eristävä algoritmi prosessori P_i:lle

Virheen tapahtuminen todetaan vertaamalla $ComputeOutput_i$:lla prosessorien syötteistä laskettua arvoa $Processors_i$:n monikoiden A arvoihin. Korjauslaskuria lisätään yhdellä joka pulssilla. $MajorityInputs(Processors_i)$ käyttää tulostukseen vain sellaisia monikoita, joilla etäisyysarvo on yhtä paljon tai vähemmän kuin korjauslaskurin arvo (riveillä 20 ja 21). A_i arvo asetetaan vain jos korjauslaskuri on arvoltaan $d + 1$ (viimeiset rivit).

5 Virheenhakukoodit ja korjaus

Tässä luvussa kerrotaan kuinka pitkien suoritusten (task) stabiloimattomat tila-algoritmit voidaan muuntaa itsestabiloiviksi käyttäen virheenhakukoodeja, tilatietojen pyramididatarakenteita ja vahtikoiralaskureita (watchdog counter).

Virhemalli. Itsestabiloivien algoritmien virhemallit usein tukeutuvat huonoimman tilavirheen kuvaavaan *pahimman tilapäisen vian malliin* (malicious transient fault model), jolloin keskimääräinen vakiinnuttamisaika (average convergence time) pitenee. Paikallisen stabiloinnin keskimääräinen vakiinnuttamisaika saadaan lyhenemään käyttämällä edellisen sijaan *tavanomaisen tilapäisen vian mallia* (non-malicious transient fault model). Tässä mallissa vikatilanteessa prosessorin uusi tila on saatu tasaisella todennäköisyydellä prosessorin tila-avaruudesta. Uuden konfiguraation todennäköisyys on sitten yhdistetty todennäköisyys siitä, että vikaantuneet prosessorit saavat jokainen itsenäisesti tilansa tästä samasta konfiguraatiosta.

Virheenhakukoodit. Keskimääräistä vakiinnuttamisaikaa voidaan lyhentää myös virheenhakukoodien (error-detection code) käytöllä jolloin virheiden löytäminen tehostuu. Virheen havaitseminen jälkeinen menettely toteutetaan sovelluskohtaisesti. Vakiinnuttamisajan minimoimisen merkityksen kasvaessa virheenhakukoodin kokoa jatkuvasti lisätään. Tällöin tosin virheenhakukoodin sopimisen todennäköisyys tilatietoihin vähenee koska väliaikainen vika voi hyvinkin korruptoida ison osan tai vaikka kaikki prosessorin muistibitit.

Korjauskeema pitkille suorituksille käyttää hyväkseen virheenhakukoodeja ja pyramiditilatietorakennetta [Dol00, sivut 123-133]. Oletuksena on, että kaikki virheet löytyvät virheenhakukoodeilla. Tällöin vian havainneet prosessorit alustavat pyramidinsa tyhjiksi ja alkavat keräämään vikaantumattomilta naapureilta tilatietoja. Pyramidit voidaan koostaa luotettavasti näistä tiedoista ja normaalisuoritusta voidaan jatkaa.

Vikatilanteen sattumisen jälkeen prosessoreiden käyttötilat (status) ovat joko *viallinen*, *rajatapaus*, tai *toiminnassa*. Prosessorit, jotka ovat havainneet naapureidensa vaihtaneen käyttötilansa *viallinen* -tilaan, mutta eivät itse ole havainneet vikaa, vaihtavat tilansa *rajatapaus* -tilaan. Nämä prosessorit odottavat kunnes vikaantuneet prosessorit ovat koonneet pyramidinsa ja vasta sitten kokoavat

omansa uudelleen. Tämän tilanteen tunnistamiseen käytetään topologiankeräysproseduuria. Siinä joka aikayksikössä vikaantuneet ja *rajatapaus* –prosessorit lähettävät keräämänsä topologian naapureilleen ja samaan aikaan vastaanottavat vastaavasti naapureidensa topologiatiedot. Tämä tapahtuu ajassa joka on aikayksiköissä sama kuin korruptoituneen alueen halkaisija. Tämän jälkeen voidaan yhden aikayksikön aikana vertaamalla vikaantuneiden prosessoreiden naapureiden tiloja tunnistaa tilanne jossa kaikki vikaantuneet prosessorit ovat korjannet tilatietopyramidinsa. Nyt puuttuvat ainostaan vikaantuneiden prosessoreiden omat vikatilannetta edeltävät tilatiedot ja ne koostetaan vikaantumattomien naapureiden tiedoista ja prosessoreiden omista tilasiirtymäfunktioista. Vikatilaanteeseen sotkeentuneet prosessorit voivat päätellä topologiatiedoista ja kierroslukulaskureista alkaen vikatilanteesta milloin muu järjestelmä on koonnut pyramidinsa valmiiksi ja muuttavat tilansa *toiminnassa* –käyttötilaan kaikki samanaikaisesti.

Sattumanvaraisen tilan mahdollisuus on olemassa jos virheenhakukoodit eivät havaitsekaan kaikkia virhemallin sallimia vikoja. Näitä vikoja haetaan tilapäisen vian nuuskijoilla (transien fault detectors) ja vahtikoiralaskureilla (watchdog counters). Kun nuuskija löytää vian, prosessori laittaa vahtikoiralaskurin päälle ja antaa korjausskeeman hoitaa järjestelmän kuntoon. Kun laskuri pääsee ylärajalleen, järjestelmä nuuskitaan taas ja laskuri nollataan jos vika ei ole poistunut. Yläraja on laskettu siten, että se täsmää korjausskeeman aikavaativuuteen eli $O(d)$.

6 Yhteenveto

Paikallisen stabiloinnin tarkoituksena on rajata virheen vaikutusalueita siten, että vikatilanteen vahingot jäävät mahdollisimman pienelle alueelle. Tämä saadaan aikaan varhaisella vian havaitsemisella ja sen eristämällä niin että vahingot eivät pääse laajenemaan. Lisäksi viat voidaan korjata niin että turvallinen konfiguraatio tai jopa vikatilannetta edeltävä konfiguraatio palautetaan.

Itsestabiloivat algoritmit eivät takaa muutosprosessiin kuluvaan aikaan eivätkä käyttäytymistään järjestelmän stabiloitumisen aikana. Käsitellyt kolme erilaista paikallisen stabiloinnin tapaa lisäävät hyödyllisiä ominaisuuksia itsestabiloivien algoritmeihin. Superstabiloivuus lisää itsestabiloiviin algoritmeihin aikatakuun sekä hallittavuuden virhemallinsa rajoissa. Tilapäisten vikojen käsittelyssä käytetään virheitä eristäviä itsestabiloivia algoritmeja jotka pyrkivät palauttamaan n korruptoivan vian jälkeen turvallisen konfiguraation $O(n)$ syklissä satunnaisesta lähtötilanteesta. Pitkille suorituksille voidaan käyttää virreehakukoodeja, pyramiditilatietorakenteita ja vahtikoiralaskureita nopeuttamaan virreehakuprosessia ja saavuttamaan niiden avulla itsestabiloivia ominaisuuksia sekä toipumisen paikallisuutta ja hallittavuutta myös ajallisesti.

7 Viitteet

Do100 Shlomi Dolev. Self-Stabilization. MIT Press, 2000.