

Itsestabiloiva bysanttilainen yhteisymmärrys

Timo Virkkala

Helsinki 29.10.2007

Hajautetut algoritmit -seminaari

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

1 Johdanto

Idealisessa tilanteessa jotakin algoritmia suoritetaan ympäristössä, jossa mikään ei voi mennä vikaan. Algoritmi käynnistetään, ja jonkin ajan kuluttua suoritus päättyy ja tuottaa tuloksenaan halutun vastauksen.

Reaalimaailmassa tämän ideaalitalanteen saavuttaminen on valitettavan harvinaista. Varsinkin hajautettujen järjestelmien tapauksessa vikoja voi ilmetä monilla tahoilla. Järjestelmän toimijoiden (**solmujen**, engl. *nodes*) suoritus voi häiriintyä ja niiden välisten yhteyksien (**verkon** ja sen linkkien eli **kaarien**, engl. *graph*, *arcs*) luotettavuus voi kärsiä. Tavoitteena onkin saada algoritmin toimimaan halutulla tavalla huolimatta näistä epäluotettavuustekijöistä.

Tässä artikkelissa esitellään algoritmi, jonka tavoitteena on saada järjestelmän solmut pääsemään sopimukseen (**yhteisymmärrykseen**, engl. *agreement*, ei vakiintunutta suomennosta) jostakin arvosta huolimatta järjestelmässä tapahtuvista virheistä [DaD06]. Algoritmi pyrkii ottamaan huomioon sekä ohimenevät (*transient*) että pysyvät (*permanent*) virheet, tietyn oletuksen.

Algoritmia, joka päättyy mistä tahansa alkutilasta hallittuun, konsistenttiin tilaan kutsutaan **itsestabiloivaksi** (*self-stabilizing*) [Dij74]. Tällainen algoritmi pystyy toipumaan mistä tahansa määrästä virheitä, kunhan järjestelmän toiminta on tämän jälkeen virheetöntä riittävän pitkän ajan. Itsestabiloivuuteen paneudutaan tarkemmin kappaleessa 2.

Virheitä, jotka saavat järjestelmän osat toimimaan aktiivisesti algoritmin suoritusta haitaten kutsutaan **bysanttilaisiksi virheiksi** (*byzantine faults*) [LSP82]. Algoritmia, joka kykenee toimimaan bysanttilaisten virheiden läsnäollessa kutsutaan usein bysanttilaiseksi algoritmiksi. Bysanttilaisten virheiden alaisuudessa toimimiseen perehdytään kappaleessa 3.

2 Itsestabiloivuus

Lähes kaikissa järjestelmissä, ja erityisesti hajautetuissa järjestelmissä voi esiintyä valtava määrä erilaisia vikoja ja häiriöitä. Perinteinen lähestymistapa virheiden hallintaan on ollut ohjelmoida algoritmien toteutuksiin tarkistuksia tyypillisimpien virheiden havainnointiin ja pyrkiä sitten toipumaan näistä virheistä, tai ainakin tuoda järjestelmä alas hallitusti. Kaikkiin mahdollisiin virhetilanteisiin varautuminen tällä

tavoin on kuitenkin raskas tehtävä jo suhteellisen pienissäkin järjestelmissä.

Vikasietoisempi vaihtoehto on pyrkiä suunnittelemaan järjestelmä **itsestabiloivaksi**, eli sellaiseksi, että se mistä tahansa alkutilasta käynnistettynä päätyy hallittuun, konsistenttiin tilaan rajallisessa määrässä askeleita [Dij74]. Voidaan siis ajatella, että tietyn ajan kestäneen virheellisen toiminnan jakson jälkeen virheet loppuvat jollakin tietyllä ajanhetkellä, joka valitaan algoritmin käynnistyshetkeksi. Itsestabiloiva algoritmi pystyy siis toipumaan johonkin konsistenttiin tilaan riippumatta yksittäisten solmujen tilasta käynnistyshetkeksi valitulla hetkellä. Tällainen itsestabiloiva algoritmi pystyy suoriutumaan myös muunlaisista virheistä kuin niistä, jotka ohjelmoija on osannut ottaa huomioon ohjelman kirjoitushetkellä.

3 Bysanttilainen yhteisymmärrys

Oman ongelmansa muodostavat ns. *bysanttilaiset virheet*, jossa osan hajautetun järjestelmän solmuista ajatellaan toimivan vihamielisesti, eli pyrkivän aktiivisesti haittaamaan muun järjestelmän toimintaa [LSP82]. Todellisuudessa kyseessä on useimmiten solmu, jonka jokin vikatilanne aiheuttaa sen lähettämien arvojen olevan epärealistisia tai saa solmun lähettämään odottamattomia viestejä. Ajatteleamalla solmujen toimivan aktiivisesti muita vastaan voidaan kuitenkin helpommin käsitellä pahimpia mahdollisia vikaskenaarioita.

Bysanttilaisten virheiden nimitys tulee esimerkkitapauksesta, jossa joukko Bysantin imperiumin armeijaosastoja komentajineen on kokoontunut vihollisen ylivoimaisen armeijan ympärille. Komentajien tavoitteena on saada aikaiseksi synkronoitu hyökkäys vihollista vastaan. Armeijaa johtaa kenraali, joka lähettää joukon viestinviejiä välittämään muiden osastojen komentajille viestin hyökkäyksen ajankohdasta. Mitä suurempi osa osastoista saadaan hyökkäämään oikeaan aikaan, sitä pienemmiksi jäävät tappiot vihollisen ylivoimaista armeijaa vastaan.

Vihollinen on kuitenkin viekas. Se voi lahjoa osan komentajista tai jopa kenraalin puolelleen. Luotettavien komentajien täytyykin päästä yhteisymmärrykseen huolimatta siitä, että osa komentajista saattaakin toimia muita vastaan ja lähettää eri vastaanottajille erilaisia viestejä tavoitteenaan saada osastot manipuloitua hyökkäämään yksi kerrallaan ylivoimaista vihollista vastaan.

Jotta osastojen komentajat pääsisivät yksimielisyyteen hyökkäyksen ajankohdasta, on niiden kommunikoitava keskenään. Kun kenraali lähettää viestinsä osastojen

komentajille, komentajat välittävät kenraalilta saamansa viestin eteenpäin kaikille muille komentajille. Tämän jälkeen komentajat välittävät edelleen muilta komentajilta vastaanottamiaan viestejä, kunnes voidaan olla varmoja siitä, että kaikki luotettavat komentajat ovat yksimielisiä hyökkäyksen ajankohdasta. Jos armeijan kenraali on luotettava, kaikkien luotettavien komentajien on päädyttävä siihen ajankohtaan, jonka kenraali lähetti alkuperäisissä viesteissään.

On osoitettu [LSP82], että ongelman ratkaiseminen vaatii yhteensä vähintään $3f + 1$ komentajaa (joista yksi on kenraali), jos joukossa on f epäluotettavaa komentajaa. Tätä pienemmällä määrällä yksimielisyyteen päätyminen on mahdotonta, koska viestejä sopivasti manipuloimalla epäluotettavat komentajat voivat estää luotettavia komentajia pääsemästä koskaan yksimielisyyteen hyökkäyksen ajankohdasta.

Otetaan esimerkiksi tapaus, jossa on yksi epäluotettava komentaja. Kun kenraali K_0 lähettää viestin "hyökkäys klo. 13:00", luotettava komentaja K_1 välittää sen eteenpäin komentajalle K_2 , joka on petturi. K_2 lähettääkin K_1 :lle viestin, joka väittää kenraalin käskeneen hyökkäyksen klo. 15:00. Toisessa tapauksessa petturi onkin kenraali, joka lähettää K_1 :lle viestin, jossa sanotaan hyökkäyksen tapahtuvan klo. 13:00 ja K_2 :lle viestin, jonka mukaan hyökätään klo. 15:00. Kumpikin komentajista välittää kuuliaisesti saamansa viestin eteenpäin. Komentajalla K_1 ei ole mitään mahdollisuutta erottaa näitä kahta tilannetta toisistaan.

Ongelman ratkaiseminen vaatii neljännen komentajan (K_3) mukaanottamisen ($3f + 1 = 4$, kun $f = 1$) [PSL80]. Nyt ensimmäisessä tapauksessa (petturi on K_2) komentaja K_1 saa "hyökkäys klo. 13:00" -viestin sekä K_0 :lta että K_3 :lta. Tällöin K_2 :n välittämän viestin sisällöllä ei ole väliä, koska enemmistö viesteistä on oikean ajankohdan kannalla. Toisessa tapauksessa (petturi on kenraali K_0) komentajat voivat etukäteen sopia, että ristiriitatilanteessa (millään kellonajalla ei ole enemmistöä) otetaan aikaisin ehdotettu kellonaika. Tällöin jos K_0 käskää K_1 :tä hyökkäämään klo. 13:00, K_2 :ta klo. 15:00 ja K_3 :a klo 17:00, päätyvät komentajat yhden viestinvaihtokierroksen jälkeen hyökkäämään klo. 13:00.

4 Itsestabiloiva bysanttilainen yhteisymmärrys

Itsestabiloivuuden ja bysanttilaisten virheiden sietämisen yhdistäminen on jo suurempi ongelma. Lähes kaikki tunnetut ratkaisut yhteisymmärryksen saavuttamiseen huolimatta bysanttilaisista virheistä edellyttävät solmujen välistä synkronointia ennen protokollan aloittamista. Jos järjestelmän solmut ja niitä yhdistävä tietoliikenne-

verkko voivat olla missä tahansa alkutilassa, on yhteisymmärryksen saavuttaminen huomattavasti hankalampaa.

Ariel Daliotin ja Danny Dolevin kehittämä SS-BYZ-AGREE -algoritmi yhdistää onnistuneesti nämä kaksi vikamallia [DaD06]. Algoritmi toipuu mistä tahansa järjestelmän tilasta – jopa sellaisesta, jossa enemmän kuin kolmasosa järjestelmän solmuista toimii bysanttilaisesti tai muuten väärin, kunhan järjestelmän tietoliikenneverkko sekä riittävä määrä järjestelmän solmuja toimivat oikein riittävän pitkän ajan.

Algoritmin tavoitteena on saada kaikki järjestelmän normaalisti toimivat solmut pääsemään yhteisymmärrykseen kenraalin lähettämästä arvosta. Mikä tahansa järjestelmän solmuista voi toimia kenraalina. Kun kenraali aloittaa protokollan suorittamisen, se lähettää kaikille muille solmuille yksityisen arvonsa. Kaikki oikein toimivat solmut vastaanottavat tämän viestin ja aloittavat näin myös protokollan suorituksen. Jos kaikki korrekrit solmut aloittavat protokollan suorituksen riittävän pienellä aikavälillä, järjestelmän kaikki korrekrit solmut päätyvät samaan arvoon. Lisäksi, jos kenraali toimii oikein, kaikki korrekrit solmut päätyvät kenraalin lähettämään arvoon ja hyväksyvät sen. Jos sen sijaan kaikki korrekrit solmut eivät aloita protokollaa riittävän pienellä aikavälillä, niin jos jokin korrekki solmu hyväksyy tyhjästä poikkeavan arvon, niin kaikki korrekrit solmut päätyvät yhteisymmärrykseen tästä arvosta ja hyväksyvät sen.

4.1 Määritelmiä

Solmu on **ei-viallinen** (*non-faulty*), jos se noudattaa protokollaa täsmällisesti (tottelevaisuus, *obedience*), käsittelee kaikki vastaanottamansa protokollaan kuuluvat viestit π :n aikayksikön sisällä (rajallinen käsittelyaika, *bounded processing time*), sekä sen paikallinen kello on riittävän lähellä reaaliaikaa (rajallinen poikkeama, *bounded drift*). Solmu on **viallinen** (*faulty*), jos se rikkoo mitään näistä ehdoista. Solmun kellon katsotaan olevan riittävän lähellä reaaliaikaa, jos se noudattaa globaalia vakiota $0 < \rho < 1$ (tyypillisesti $\rho \approx 10^{-6}$) siten, että kaikilla reaaliaikaintervalleilla $[u, v]$ pätee $(1 - \rho)(v - u) \leq \text{kello}(v) - \text{kello}(u) \leq (1 + \rho)(v - u)$, missä $\text{kello}(x)$ on solmun paikallisaika reaaliajanhetkellä x .

Jos solmu on toiminut ei-viallisesti riittävän pitkään (Δ_{solmu} aikayksikköä), sen katsotaan olevan **korrekki** (*correct*).

Viestintäverkko on **ei-viallinen**, jos se toimittaa kaikki lähetetyt viestit perille δ :n aikayksikön sisällä, sekä säilyttää viestin sisällön ja tiedon viestin lähettäjistä muut-

tumattomana.

Jos viestintäverkko on toiminut ei-viallisesti riittävän pitkään (Δ_{verkko} aikayksikköä), sen katsotaan olevan **korrekti** (*correct*).

Kun viestintäverkko toimii ei-viallisesti, jokainen jonkin ei-viallisen solmun lähettämä viesti tulee vastaanotetuksi ja käsitellyksi jokaisessa ei-viallisessa solmussa $d \equiv \delta + \pi:n$ aikayksikön sisällä.

Järjestelmä on **johdonmukainen** (*coherent*), jos vähintään $n - f$ solmua toimii korrektisti (päätosvaltainen joukko, *quorum*), missä n on järjestelmän solmujen kokonaisuus ja f on potentiaalisesti ei-korrektien solmujen määrän yläraja järjestelmän tasapainotilassa, sekä viestintäverkko toimii korrektisti (verkon korrektius, *network correctness*).

Solmu **päättää** ajanhetkellä τ jos se lopettaa protokollan suorituksen sillä paikallisaianhetkellä ja palauttaa epätyhjän ($\neq \perp$) arvon. Solmu **keskeyttää**, jos se lopettaa ja palauttaa tyhjän arvon (\perp). Solmu **palauttaa arvon**, jos se joko keskeyttää tai päättää.

4.2 SS-BYZ-AGREE -protokollan toiminta

Yhteisymmärryksen muodostaminen alkaa, kun jokin solmuista alkaa toimia kenraalina ja lähettää viestin (*Initiator*, G , m) kaikille solmuille. Jokainen tämän viestin vastaanottava solmu kutsuu SS-BYZ-AGREE -protokollaa, joka puolestaan kutsuu INITIATOR-ACCEPT -primitiiviä. Jos jokin solmu (joka ei ole vastaanottanut kenraalin alkuperäistä viestiä) toteaa vastaanottamiensa viestien perusteella riittävän suuren osan muista solmuista kutsuneen protokollaa, se suorittaa osia INITIATOR-ACCEPT primitiivistä (kuitenkaan varsinaisesti kutsumatta primitiiviä) ja osallistuu näin yhteisymmärryksen muodostamiseen. SS-BYZ-AGREE -protokollan pseudokoodi on esitetty liitteessä 1, samoin kuin primitiivien INITIATOR-ACCEPT ja MSGD-BROADCAST.

Protokollan suoritus jakautuu kahteen osaan. Ensimmäisessä osassa suoritetaan *esihyväksyminen* ja toisessa *hyväksyminen*. Esihyväksyminen alkaa, kun solmu joko kutsuu INITIATOR-ACCEPT -primitiiviä (vastaanotettuaan kenraalin aloitusviestin) tai suorittaa sen osia (vastaanotettuaan muiden solmujen esihyväksyntäviestejä) ja päättyy, kun solmu q suorittaa $\mathbf{I-accept}\langle G, m, \tau_q^G \rangle$ -operaation, missä G on protokollan aloittanut kenraali, m on esihyväksytty arvo (sekä myös kenraalin lähettämä alkuperäinen arvo, jos kenraali on ei-viallinen) ja τ_q^G on aika, jolloin solmu q katsoo

Suoritetaan solmussa q . τ_q on solmun q paikallisaika. Jokainen lohko suoritetaan vain kerran ja vain, jos sen esiehto pätee. Lohko Q suoritetaan vain, jos protokollaa kutsutaan, eli kun solmu q vastaanottaa viestin ($Initiator, G, m$) solmulta G . Protokollan muut osat suoritetaan alkaen riviltä R1, kun solmu q tekee I-accept -operaation, tai kun τ_q^G on määritelty.

Q. INITIATOR-ACCEPT(G, m)

R1. **if** I-accept(G, m', τ_q^G) **and** $\tau_q - \tau_q^G \leq 4d$ **then**

R2. $value := \langle G, m' \rangle$;

R3. MSGD-BROADCAST($q, value, 1$);

R4. **stop and return** $\langle value, \tau_q^G \rangle$.

S1. **if** τ_q mennessä vastaanotettu r erillistä viestiä ($p_i, \langle G, m'' \rangle, \tau_i, i$)
(missä $\tau_q \leq \tau_q^G + (2r + 1) \cdot \Phi$ ja $\forall i, j : (1 \leq i \leq r) \wedge (p_i \neq p_j \neq G)$) **then**

S2. $value := \langle G, m'' \rangle$;

R3. MSGD-BROADCAST($q, value, r + 1$);

R4. **stop and return** $\langle value, \tau_q^G \rangle$.

T1. **if** τ_q mennessä $|broadcasters| < r - 1$ (missä $\tau_q > \tau_q^G + (2r + 1) \cdot \Phi$) **then**

T2. **stop and return** $\langle \perp, \tau_q^G \rangle$.

U1. **if** $\tau_q > \tau_q^G + (2f + 3) \cdot \Phi$ **then**

U2. **stop and return** $\langle \perp, \tau_q^G \rangle$.

siivous:

$3d$ arvon palauttamisen jälkeen nollassa suorituskertaan

 liittyvät INITIATOR-ACCEPT ja MSGD-BROADCAST;

 Poista kaikki $(2f + 3) \cdot \Phi + 3d$ aikayksikköä vanhemmat arvot ja viestit.

Kuva 1: SS-BYZ-AGREE -protokolla

kenraalin G lähettäneen viestinsä.

Esihyväksymisen tarkoituksena on muodostaa arvio ajasta, jolloin kenraali on aloittanut protokollan suorituksen sekä valita arvo, jota käytetään kandidaattina hyväksymisvaiheessa muodostettavalle yhteisymmärrykselle. Esihyväksymisessä solmut osoittavat tukensa kenraalin G lähettämälle arvolle m lähettämällä kaikille solmuille (myös itselleen) viestin ($support, G, m$). Kun solmu on vastaanottanut tällaisen viestin riittävältä määrältä solmuja, se ilmoittaa kaikille solmuille olevansa valmis tämän arvon hyväksymiseen lähettämällä viestin ($ready, G, m$). Kun riittävä määrä $ready$ -viestejä on vastaanotettu, solmu esihyväksyy arvon suorittamalla operaation I-accept(G, m, τ_q^G).

Suoritetaan solmussa q . τ_q on solmun q paikallisaika. Rivit L1 ja L2 suoritetaan toistuvasti, kunnes I-accept on suoritettu. Loput rivit suoritetaan korkeintaan kerran, jos esiehdot pätevät. K-lohko suoritetaan vain, jos ja kun primitiiviä kutsutaan.

K1. **if** $\tau_q - last_ \tau_q > 7d$ **and if** hetkellä $\tau_q - d$ $initiator[G, _] = \perp$ **then**

K2. lähetä $(support, G, m)$ kaikille;

K3. $initiator[G, m] := \tau_q - d$;

L1. **if** vastaanotettu $(support, G, m)$ vähintään $\geq n - 2f$ erilliseltä solmulta $\alpha \leq 4d$ aikayksikön sisällä toisistaan **then**

L2. $initiator[G, m] := \max[initiator[G, m], (\tau_q - \alpha - 2d)]$;

L3. **if** vastaanotettu $(support, G, m)$ vähintään $\geq n - f$ erilliseltä solmulta $2d$ aikayksikön sisällä toisistaan **then**

L4. lähetä $(ready, G, m)$ kaikille;

M1. **if** vastaanotettu $(ready, G, m)$ vähintään $\geq n - 2f$ erilliseltä solmulta **then**

M2. lähetä $(ready, G, m)$ kaikille;

M3. **if** vastaanotettu $(ready, G, m)$ vähintään $\geq n - f$ erilliseltä solmulta **then**

M4. $\tau_q^G := initiator[G, m]$; I-accept $\langle G, m, \tau_q^G \rangle$; $last_ \tau_q := \tau_q$;

siivous:

Poista kaikki $\Delta + 7d$ aikayksikköä vanhemmat arvot ja viestit.

if $last_ \tau_q > \tau_q$ **then** $last_ \tau_q := \perp$.

Kuva 2: INITIATOR-ACCEPT(G, m) -primitiivi

Esihyväksymisen jälkeen alkaa hyväksymisvaihe. Siinä solmut pyrkivät päättämään onko protokollan suorituksessa mukana riittävä määrä korrekkeja solmuja ja voidaan esihyväksymisessä valittu kandidaattiarvo hyväksyä. Esihyväksymisvaiheessa ankkuroitua protokollan aloitusaikaa ja ennalta tiedettyä viestien välittämiseen ja käsittelyyn kuluva aikaa käytetään rajoittamaan tietyn tyyppisten viestien odotusaikaa. Jos riittävää määrää odotetun tyyppisiä viestejä ei saada määräaikaan mennessä, voidaan yhteisymmärrykseen pyrkiminen keskeyttää.

Hyväksymisvaiheessa solmut suorittavat viestinvaihtokierroksia, joissa ne yrittävät päästä yhteisymmärrykseen lopullisesta arvosta. Kierrokset perustuvat lähetettyihin ja vastaanotettuihin viesteihin ja voivat mennä päällekkäin. Näin solmut voivat kiirehtiä protokollan läpi tilanteessa, jossa järjestelmä toimii luotettavasti. Solmut lakkaavat osallistumasta kierroksiin liittyvään viestinvaihtoon $3d$ aikayksikköä sen jälkeen, kun ne ovat hyväksyneet jonkin arvon.

Suoritetaan (p, m, k) -kolmikolle solmussa q . Solmut lähettävät jokaisen yksittäisen viestin vain kerran. Solmut suorittavat lohkot vain kun τ^G on määritelty. Vastaa-
notetut viestit tallennetaan kunnes ne voidaan käsitellä. Yksittäisen solmun lähet-
tämät toisteiset viestit jätetään huomiotta. Operaatio `hyväksy` (p, m, k) suoritetaan
vain kerran.

Solmussa $q = p$:

V. lähetä $(init, p, m, k)$ kaikille;

W1. Hetkellä $\tau_q : \tau_q \leq \tau_q^G + 2k \cdot \Phi$

W2. **if** vastaanotettu $(init, p, m, k)$ solmulta p **then**

W3. lähetä $(echo, p, m, k)$ kaikille;

X1. Hetkellä $\tau_q : \tau_q \leq \tau_q^G + (2k - 1) \cdot \Phi$

X2. **if** vastaanotettu $(echo, p, m, k)$ vähintään $\geq n - 2f$ erilliseltä solmulta **then**

X3. lähetä $(init', p, m, k)$ kaikille;

X4. **if** vastaanotettu $(echo, p, m, k)$ vähintään $\geq n - f$ erilliseltä solmulta **then**

X5. hyväksy (p, m, k) ;

Y1. Hetkellä $\tau_q : \tau_q \leq \tau_q^G + (2k + 2) \cdot \Phi$

Y2. **if** vastaanotettu $(init', p, m, k)$ vähintään $\geq n - 2f$ erilliseltä solmulta **then**

Y3. **broadcasters** := **broadcasters** \cup $\{p\}$;

Y4. **if** vastaanotettu $(init', p, m, k)$ vähintään $\geq n - f$ erilliseltä solmulta **then**

Y5. lähetä $(echo', p, m, k)$ kaikille;

Z1. Millä tahansa hetkellä:

Z2. **if** vastaanotettu $(echo', p, m, k)$ vähintään $\geq n - 2f$ erilliseltä solmulta **then**

Z3. lähetä $(echo', p, m, k)$ kaikille;

Z4. **if** vastaanotettu $(echo', p, m, k)$ vähintään $\geq n - f$ erilliseltä solmulta **then**

Z5. hyväksy (p, m, k) ;

siivous:

 Poista kaikki $(2f + 3) \cdot \Phi$ aikayksikköä vanhemmat arvot ja viestit.

Kuva 3: MSGD-BROADCAST (p, m, k) -primitiivi

4.3 Esimerkkejä

Tarkastellaan eräitä esimerkkitalanteita verkossa, jossa on neljä solmua q_0, q_1, q_2 ja q_3 (eli $n = 4$). Solmut varautuvat yhden bysanttilaisen virheen torjuntaan ($f = 1$). Oletetaan ensin, että kaikki solmut toimivat oikein, mitään virheitä ei ole esiintynyt ja kaikki solmut aloittavat siten puhtaasta alkutilasta. Tässä protokollaa tarkas-
tellaan lähinnä lähetettyjen viestien kannalta; protokollan yksityiskohdat löytyvät

liitteestä 1.

Solmu q_0 toimii kenraalina ja lähettää muille solmuille viestin ($Initiator, q_0, m$), missä m on jokin arvo. Kukin solmu q_1 , q_2 ja q_3 vastaanottaa tämän viestin ja kutsuu primitiiviä $INITIATOR-ACCEPT(q_0, m)$. Kukin solmu toimii identtisesti, joten tarkkaillaan protokollan suoritusta solmun q_1 näkökulmasta. Sen tietorakenteet ovat tyhjiä, joten ehdon K1 katsotaan pätevän, K-lohko suoritetaan ja siis q_1 lähettää kaikille viestin ($support, q_0, m$).

Pian tämän jälkeen q_1 vastaanottaa saman viestin kaikilta neljältä solmulta (myös kenraalilta ja itseltään; kaikki solmut suorittavat samaa protokollaa ja viestit lähetetään kaikille, myös itselle). $4 \geq n - f$ pätee, joten q_1 lähettää kaikille viestin ($ready, q_0, m$). Jälleen sama viesti vastaanotetaan kaikilta solmuilta, jolloin solmu q_1 määrittelee arvot $\tau_{q_1}^{q_0}$ ja $last_ \tau_{q_1}$ ja suorittaa $I-accept\langle q_0, m, \tau_{q_1}^{q_0} \rangle$.

Tämän jälkeen suoritus jatkuu riviltä R1. Ehdot pätevät, joten q_1 kutsuu primitiiviä $MSGD-BROADCAST(q_1, \langle q_0, m \rangle, 1)$. q_1 lähettää aluksi kaikille viestin ($init, q_1, m, 1$). Tämän jälkeen se vastaanottaa viestit ($init, q_0, m, 1$), ($init, q_1, m, 1$), ($init, q_2, m, 1$) sekä ($init, q_3, m, 1$) ja vastaavasti lähettää eteenpäin viestit ($echo, q_0, m, 1$), ($echo, q_1, m, 1$), ($echo, q_2, m, 1$), ($echo, q_3, m, 1$). Solmu q_1 vastaanottaa vastaavanlaisia $echo$ -viestejä ($echo, q_i, m, 1$) yhteensä 16 kappaletta (4 erilaista viestiä, kukin 4:ltä erilliseltä solmulta). Vastaanotettuaan jonkin näistä viesteistä 3:lta erilliseltä solmulta solmu q_0 hyväksyy arvot ($p, m, 1$), missä p on sen solmun numero, johon liittyvää $echo$ -viestiä vastaanotettiin ensimmäisenä 3:lta solmulta. Tämän jälkeen se päättää protokollan suorituksen palauttaen arvon $\langle q_0, m \rangle$ alkuajanhetkelle $\tau_{q_1}^{q_0}$. Samaan lopputulokseen päätyvät myös kaikki muut solmut.

Tarkastellaan sitten tilannetta, jossa solmu q_2 toimii bysanttilaisesti. Se voi pyrkiä muuttamaan laskennan lopputulosta usein eri tavoin. q_2 voi esimerkiksi toimia kenraalimaisesti ja lähettää muille solmuille $Initiator$ -viestin (tai useita viestejä). Tällöin lopputulos riippuu siitä, milloin q_2 viestinsä lähettää ja lähettääkö se saman arvon usealle solmulle. Jos se lähettää $Initiator$ -viestinsä ennen "oikeaa" kenraalia q_0 , ja lähettää saman arvon vähintään kolmelle solmulle (joista se voi itse olla yksi, jos se tämän jälkeen toimii konsistentisti tukien omaa arvoaan; tällöin q_2 :n voitaisiin itse asiassa todeta toimivan korrektisti), tulee sen lähettämä arvo hyväksytyksi kaikissa solmuissa. Jos se puolestaan lähettää eri solmuille eri arvoja, ei se saa millekään lähettämälleen arvolle riittävää tukea ja näin kaikki solmut päätyvät tyhjään arvoon. Jos q_2 puolestaan lähettää oman $Initiator$ -viestinsä q_0 :n jälkeen, jättävät muut solmut q_2 :n $Initiator$ -viestit huomiotta ja päätyvät normaalisti q_0 :n

lähettämään arvoon.

Solmu q_2 voi myös pyrkiä haittaamaan muiden solmujen yhtenäisyyspyrkimyksiä protokollan muissa vaiheissa. Tällöin se voi saada muut solmut ohjattua johonkin arvoon vain, jos muut solmut eivät ole jo suorittamassa protokollaa jonkin muun kenraalin aloittamana, ja tällöinkin johonkin epätyhjään arvoon vain, jos se itse asiassa toimii korrektisti. Jos muut solmut ovat jo asettuneet tukemaan jonkin muun kenraalin (esim. $q_0:n$) arvoa, ne käytännössä jättävät $q_2:n$ viestit huomiotta.

Kaikki tähän asti mainitut lopputulokset ovat täysin algoritmin tavoitteiden mukaisia. Kaikissa vaihtoehdoissa oikein toimivat solmut päätyvät samaan arvoon, ja jos yhteisymmärryspyrkimyksen aloittajana toimii korrekti kenraali q_0 , kaikki oikein toimivat solmut päätyvät sen lähettämään arvoon.

Koska algoritmi on itsestabiloiva, pystyy se myös pääsemään hallittuun lopputulokseen mistä tahansa alkutilasta. Oletetaan esimerkiksi, että ensimmäisen esimerkitapauksen (kaikki solmut toimivat oikein) tilanteessa, jossa solmu q_1 on juuri lähettänyt viestinsä ($init, q_1, m, 1$) kaikille solmuille, tapahtuu tietoliikennekatko, eikä q_1 saa tähän viestiin liittyviä *echo*-vastauksia miltään solmulta. Lisäksi tässä tilanteessa q_2 alkaa toimia bysanttilaisesti. Nyt (tietoliikenneyhteyksien palattua) q_1 voi saada odottamansa ($echo, q_1, m, 1$)-viestin ainoastaan $q_2:lta$, jolloin rivien X2 ja X4 ehdot eivät täyty. Tällöin algoritmin jatko riippuu siitä, mitä solmut q_0 ja q_3 olivat ehtineet lähettää ja vastaanottaa ennen tietoliikennekatkosta. Jos ne olivat ehtineet vastaanottaa riittävän määrän ($echo, q_1, m, 1$)-viestejä, voi suoritus edetä niissä Y- ja Z-lohkoihin, jolloin myös solmu q_1 pääsee niiden lähettämien *init'*- ja *echo'*-viestien avulla takaisin mukaan yhteisymmärryksen muodostamiseen. Voi myös olla, että riittävä määrä viestejä ei pääse perille, jolloin solmujen ajastimet ylittyvät ja solmut siirtyvät seuraavaan viestinvaihtokierrokseen. Ennen pitkää kaikki oikein toimivat solmut pääsevät yhteisymmärrykseen kenraalin lähettämästä arvosta ja hyväksyvät sen.

5 Yhteenveto

Tässä artikkelissa on esitelty SS-BYZ-AGREE -algoritmi itsestabiloivaan yhteisymmärryksen muodostamiseen bysanttilaisten virheiden alaisuudessa. Sen avulla saadaan aikaiseksi yhteisymmärrys hajautetussa järjestelmässä riippumatta siitä, min-kälaisia virheitä järjestelmässä on tapahtunut, kunhan virheet taukoavat riittävän pitkäksi ajaksi ja kunhan virheettömän toiminnan aikana riittävä määrä järjestel-

män solmuista toimii luotettavasti. Tämän algoritmin avulla saadaan aikaan luotettavasti toimiva yhteisymmärrys hajautetussa järjestelmässä, joka joutuu toimimaan epäluotettavassa ja osin vihamielisessä ympäristössä.

Lähteet

- DaD06 Daliot, A. ja Dolev, D., Self-stabilizing byzantine agreement. *PODC '06: Proc. 25th annual ACM symposium on Principles of distributed computing*, New York, NY, USA, 2006, ACM Press, sivut 143–152.
- Dij74 Dijkstra, E. W., Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17,11(1974), sivut 643–644.
- LSP82 Lamport, L., Shostak, R. ja Pease, M., The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4,3(1982), sivut 382–401.
- PSL80 Pease, M., Shostak, R. ja Lamport, L., Reaching agreement in the presence of faults. *J. ACM*, 27,2(1980), sivut 228–234.