Vikasietoisuus reaaliaikaisissa hajautetuissa järjestelmissä

Marja Tuominen

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Avoimien hajautettujen järjestelmien seminaari
Helsinki 26.4.2000

Avoimien hajautettujen järjestelmien seminaari

Helsingin yliopisto
Tietojenkäsittelytieteen laitos
Marja Tuominen
26. huhtikuuta 2000 Helsinki

Vikasietoisuus on keskeinen ominaisuus reaaliaikaisissa hajautetuissa järjestelmissä. Luonteensa vuoksi reaaliaikaisen järjestelmän tulee pystyä toimimaan luotettavasti, vaikka jokin siihen kuuluvista komponenteista lakkaisikin toimimasta tai toimisi virheellisesti. Vikasietoisuuspolitiikalla pyritään määrittelemään ne keinot ja menetelmät, joilla järjestelmä pystyy toipumaan sitä kohdanneista vioista. Vikasietoisuuspolitiikkaan kuuluu myös niiden virheiden määrittely, joista järjestelmän tulee selviytyä kaatumatta.

Hajautetut järjestelmät ovat yleensä niin laajoja, että ne toimivat useassa erilaisessa ympäristössä. Tästä johtuen sopivan vikasietoisuuspolitiikan valinta on entistä hankalampaa. Jos yritetään kerralla varautua kaikkiin mahdollisiin vikoihin, mitä järjestelmässä voi esiintyä, vikasietoisuuden toteuksesta tulee liian suuri ja resursseja kuluttava osio. Jos taas varaudutaan vain joihinkin spesifisiin virheisiin, koko järjestelmän luotettavuus laskee. Järjestelmässä pitäisikin olla mahdollisuus sopeutuvaan vikasietoisuuteen. Sopeutuvassa vikasietoisuudessa järjestelmä voi vaihta vikasietoisuuspolitiikkaa ja toipumismenetelmiä vaihtuvan toimintaympäristön mukaan.

Aiheluokat: C.2.4
Avainsanat: vikasietoisuus, reaaliaikaisuus, sopeutuva vikasietoisuus, toipuminen

Sisällys

1 JOHDANTO **1**

2 REAALIAIKAISUUS **2**

3 VIKASIETOISUUS **3**

3.1 Vikakategoriat *3*

4 TOIPUMISMENETELMIÄ **6**

4.1 Laitteistoviat *6*
4.2 Ohjelmistoviat *8*
4.2.1 Perinteinen tapa *8*
4.2.2 Korvaava toiminto *9*
4.2.3 Hajautettu toipumislohko *9*

5 SOPEUTUVA VIKASIETOISUUS **11**

5.1 Motivointi *11*
5.2 Toimintaperiaate *12*
5.2.1 Toimintaympäristön tyypit *12*
5.2.2 Vikamallit *12*
5.2.3 Diagnostiikka *14*
5.3 Proteus ja ROAFTS *14*
5.3.1 Proteus *14*
5.3.2 ROAFTS *16*

6 YHTEENVETO **19**

1 Johdanto

Hajautettu järjestelmä koostuu kokoelmasta prosesseja, jotka toimivat samanaikaisesti eri tietokoneilla ja jotka kommunikoivat usein varsin hitaiden ja epäluotettavien tietoliikenneyhteyksien avulla. Pyydetyn palvelun onnistuminen riippuu mahdollisesti usean eri komponentin suorituksen onnistumisesta. Vian sattuminen palvelun suorituksen aikana tällaisessa järjestelmässä saattaa siis olla todennäköisempää kuin yksittäisen tietokoneen sisällä. Vian takia ohjelmat saattavat tuottaa vääriä vastauksia tai ne saattavat lopettaa toimintansa ennen kuin ovat saanet suorittamansa tehtävän loppuun.

Vikasietoisuus tarkoittaa, että järjestelmä pystyy hallitusti toipumaan sitä kohdanneista vioista. Kaikkiin mahdollisiin vikoihin ei välttämättä voida, eikä ole tarpeenkaan, varautua. *Vikasietoisuuspolitiikalla* kuvataan, minkälaisista vioista järjestelmän tulee pystyä toipumaan ja mitä menetelmiä tähän toipumiseen käytetään.

Reaaliaikajärjestelmissä palveluille asetettu vasteaika vaikeuttaa vikasietoisuuspolitiikan suunnittelua ja toteutusta. Monet toipumismenetelmät ovat liian aikaavieviä järjestelmässä, jossa vasteaikojen noudattaminen on elintärkeätä. Perinteisesti suunniteltaessa reaaliaikaisia järjestelmiä, vikasietoisuuspolitiikka on lyöty lukkoon jo hyvin aikaisessa vaiheessa. Suunnitteluaikana ei kuitenkaan välttämättä tiedetä, millaisia vikoja järjestelmän toiminnassa on odotettavissa. Lisäksi olisi suotavaa, että järjestelmä pystyisi toimimaan tehokkaasti myös useissa erilaisissa toimintaympäristöissä. *Sopeutuva vikasietoisuus* tarjoaakin järjestelmälle mahdollisuuden siirtyä vikapolitiikasta toiseen, sen mukaan kuinka järjestelmän ympäristö muuttuu.

2 Reaaliaikaisuus

Reaaliaikajärjestelmä on kuten mikä tahansa tietokonejärjestelmä, joka saa palvelupyyntöjä, joihin se reagoi tuottaen vastauksen. Vastaukseen kuuluvat kaikkine toimenpiteet, jotka palvelun pyytäjä edellyttää tehtäväksi. Vasteajaksi kutsutaan aikaa, joka kuluu pyynnön lähettämisestä vastauksen saamiseen.

Järjestelmä on yleisesti ymmärrettynä *reaaliaikainen*, jos vasteaika pystytään pitämään alle sellaisen rajan, jonka ylittäminen aiheuttaa järjestelmässä vakavia seuraamuksia. Se mikä luokitellaan vakavaksi seuraamukseksi vaihtelee järjestelmästä ja sen toimintatarkoituksesta riippuen. Jos vasteajan ylittäminen voi johtaa onnettomuuksiin tai ihmisuhreihin, järjestelmää kutsutaan *turvallisuus-kriittiseksi* (safety-critical system). Turvallisuus-kriittisiä järjestelmiä ovat esimerkiksi lentokoneiden ja ydinvoimaloiden valvontajärjestelmät. Tavallisia reaaliaikaisia järjestelmiä kutsutaan *myös pehmeiksi reaaliaikajärjestelmiksi* (soft-real-rime) ja turvallisuus-kriittisiä järjestelmiä sanotaan puolestaan *koviksi reaaliaikajärjestelmiksi* (hard-real-time).

Reaaliaikaisuuden saavuttamiseksi on olemassa erilaisia mekanismeja kuten keskusmuistipohjaisuus, rinnakkaisuus ja nopeat tiedonsiirtomenetelmät. Keskusmuistipohjaisuus tarkoittaa sitä, että käsiteltävät tiedot sijaitsevat tietokoneen keskusmuistissa eivätkä jollain hitaalla ulkoisella tietovälineellä. Järjestelmässä voi olla sekä laitteisto- että ohjelmistotasolla monistettuja komponentteja, jotka suorittavat samoja tehtäviä. Näin järjestelmään saadaan rinnakkaisuutta ja järjestelmän saamia palvelupyyntöjä voidaan toteuttaa nopeammin.

3 Vikasietoisuus

Vikasietoisuus tarkoittaa järjestelmän kykyä jatkaa toimintaansa laitteisto- tai ohjelmistovirheestä huolimatta. Etenkin hajautetuissa järjestelmissä vian ilmeneminen silloin tällöin on hyvinkin mahdollista. Järjestelmä ei kuitenkaan saa lopettaa toimintaansa kokonaan vian sattuessa. Vian ilmetessä vikasietoinen järjestelmä joko epäonnistuu tehtävässään hallitusti tai naamioi vian niin, ettei se näy käyttäjälle.

3.1 Vikakategoriat

Hajautetussa järjestelmässä esiintyvät viat voidaan karkeasti jaotella kahteen ryhmään: laitteistovikoihin ja ohjelmistovikoihin. Cristian tarjoaa tarkemman jaottelun erilaisille hajautetussa järjestelmässä ilmeneville vioille [Cri91]: laiminlyönti (omission failure), vastausvika (response failure), ajoitus (timing failure), kaatuminen (crash failure) ja bysanttilainen vika (byzantine failure).

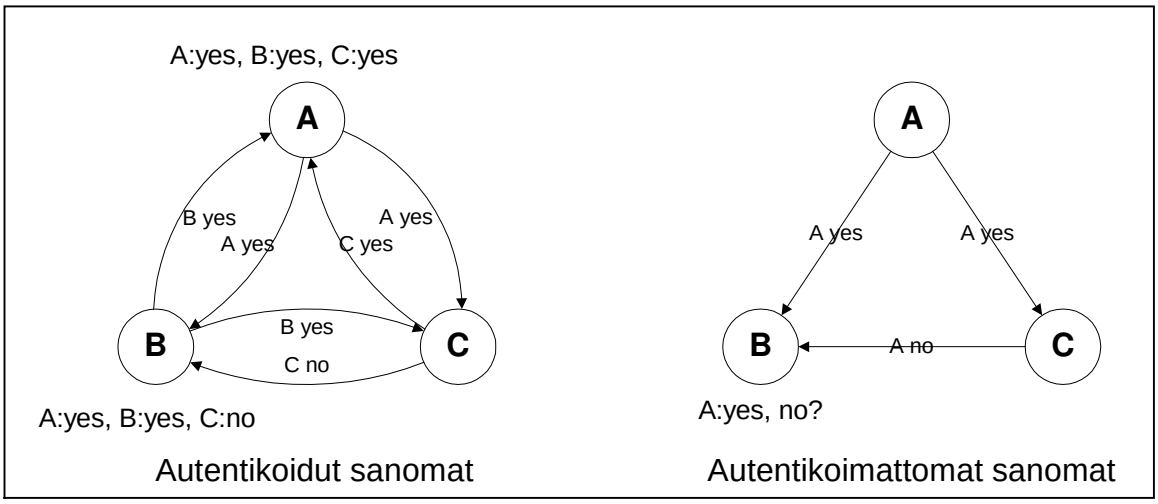
Laiminlyönti ilmenee, kun palvelin ei anna vastausta esitettyyn palvelupyyntöön. Huomattakoon, että tämä vika eroaa vastausviasta. Esimerkiksi UDP:ssä esiintyy toisinaan laiminlyönti –vikoja, sillä se toisinaan hukkaa sanomia,

Vastausvika tarkoittaa, että palvelin vastaa väärin esitettyyn palvelupyyntöön. Vastausvika jakautuu kahteen eri tapaukseen: virheelliseen palautusarvoon ja virheelliseen tilasiirtymään. Virheellinen tilasiirtymä tarkoittaa puolestaan, että palvelimen lähettämällä vastauksella on vääränlainen vaikutus järjestelmän resursseihin, esimerkiksi erilaisten tilamuuttujien arvoon.

Ajoitusviat liittyvät läheisesti reaaliaikaisiin järjestelmiin. Ajoitusvika syntyy, kun haluttua vastausta ei saada määritellyn aikaintervallin sisällä. Tämä voi tarkoittaa sekä liian myöhään, että liian aikaisin saapuvia vastauksia. Esimerkiksi siirrettäessä

videokuvaa ja ääntä yhtäaikaisesti, on tärkeää, että ääni ja kuva siirtyvät riittävän yhtäaikaisesti.

Kaatuminen käsitetään yleensä toistuvaksi laiminlyöntiviaksi. Palvelin siis jättää jatkuvasti vastaamatta sille lähetettyihin palvelupyyntöihin. Huomattakoon, että asiakkaiden on toisinaan hyvin hankala erottaa toisistaan kaatumista, hidasta vastaamista tai katkennutta yhteyttä. Palvelimen kaatuminen voidaan todeta käyttämällä ajastimia sekä palvelupyyntöjen uudelleenlähettämistä. Kaatumiset voidaan luokitella kolmeen luokkaan sen perusteella, mikä kaatuneen palvelimen tila on uudelleenkäynnistyksen jälkeen. Amnesia –kaatumiseksi kutsutaan tilaa, jossa palvelin on unohtanut kokonaan tilansa ennen kaatumista ja aloittaa uudestaan jostain alustustilasta. Paussi –kaatumisessa palvelin jatkaa samasta tilasta, missä se oli ennen kaatumistaan. Pysähdys –kaatumisessa palvelin ei käynnisty koskaan uudestaan, ei ainakaan ilman ihmisen väliintuloa.



Kuva 1: Bysanttilaisten kenraalien ongelma

Bysanttilainen vika kuvaa parhaiten tilannetta, jossa kaikki edellämainitut virheet ovat mahdollisia. Se on tilanne, jossa suurin osa palvelimista toimii oikein, mutta jotkin vialliset palvelimet toimivat niin kehnosti kuin mahdollista. Tilannetta kuvataan yleensä *Bysanttilaisten Kenraalien Ongelmana* (kuva 1). Siinä kaupungin ulkopuolelle on majoittunut N kappaletta armeijan divisioonia, jota kutakin komentaa kenraali. Kenraalit

kommunikoivat keskenään viestinviejien välityksellä ja voivat vastata toimintaehdotuksiin ainoastaan äänestämällä kyllä tai ei. Kuitenkin jotkut kenraaleista ovat ’pahoja’ ja yrittävät estää ’hyviä’ kenraaleita pääsemästä päätökseen. Algoritmi ratkaisee Bysanttilaisten Kenraalien Ongelman, jos se saa kaikki ’hyvät’ kenraalit vastaaman joko kyllä tai ei tietyn aikarajan sisällä.

4 Toipumismenetelmiä

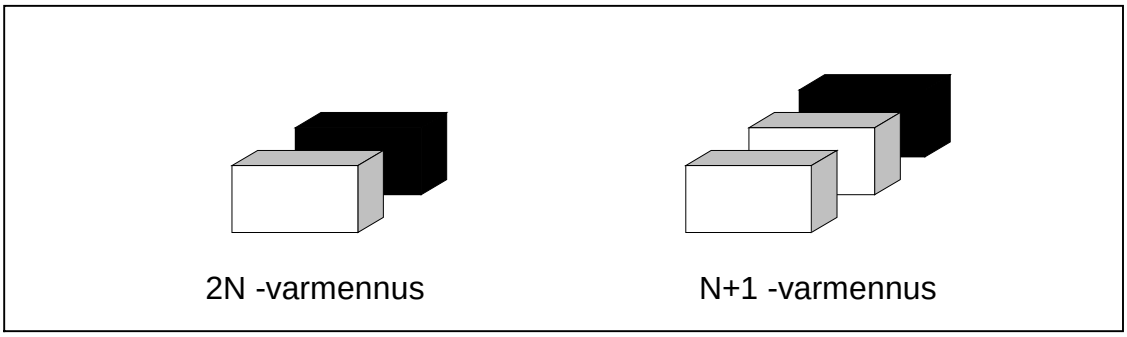
Edellä esitellyistä hajautettua järjestelmää kohtaavista vioista voidaan useimmiten toipua, kunhan vikojen ilmenemiseen on varauduttu etukäteen. Laitteistovikoihin voidaan yleensä varautua vain monistamalla järjestelmän kriittisiä osia. Ohjelmistossa ilmeneviin vikoihin on olemassa useita toipumisalgoritmeja, mutta jos järjestelmä on reaaliaikainen, monet niistä eivät ole käytettävissä hitautensa takia.

4.1 Laitteistoviat

Nykyisin tietokonejärjestelmien laitteisto pyritään tekemään mahdollisimman luotettavaksi. Tästä huolimatta laitteistokomponentit – viimeistään ikääntyessään – saattavat mennä rikki. Laitteistovikoihin voidaan varautua monistamalla laitteistokomponentteja. Esimerkiksi kaksi palvelinta voidaan liittää toisiinsa siten, että toinen toimii toisen varamiehenä. Hajautetussa järjestelmissä voidaan monistaa järjestelmän kannalta kriittisiä palvelimia, jolloin voidaan taata, että tarjotut resurssit ovat käytettävissä, vaikka jokin yksittäinen palvelin menisikin epäkuntoon.

Varmennusmenetelmiä on kahdenlaisia. Ensimmäistä kutsutaan *2N –varmennukseksi*. Tämä tarkoittaa käytännössä sitä, että yksittäisellä resurssilla on tarkoin nimetty vararesurssi. Tällaista varmennuksessa varakomponentti on tarvittaessa valmis korvaamaan rikkimenneen komponentin. Yleensä se tarkoittaa, että kyseinen varamies on ”erikoistunut” nimenomaan varmentamansa komponentin tarjoamiin palveluihin. 2N –varmennusta voi kuvata analogisesti vaikkapa jalkapallojoukkueella, jossa maalivahdilla on vaihtopenkillä varamaalivahti, joka on valmis astumaan kentälle pelaavan maalivahdin loukkaantuessa.

Toinen varmennusmenetelmä on *N+1 –varmennus*. Siitä on olemassa kaksi eri versiota: *korvaava* ja *täydentävä* varmennus. Korvaavassa varmennuksessa yksi varakomponentti voi tarvittaessa korvata usean eri komponentin eli se toimii varayksikkönä useammalle kuin yhdelle komponentille. Jalkapallojoukkueanalogiaa ajatellen tämä tarkoittaisi sitä, että varalla oleva hyökkääjä voi korvata kenet tahansa joukkueen hyökkääjistä. Täydentävässä varmennuksessa varamiestä ei otetakaan peliin, vaan muut pelaajat hoitavat myös loukkaantuneen pelaajan tehtävät. Tietokonejärjestelmässä tämä tarkoittaisi sitä, että toimivat komponentit pystyvät hoitamaan myös rikkoontuneen komponentin tehtävät.



Kuva 2: Laitteiston varmennusmenetelmät

Monissa järjestelmissä on jokin tietty valvontaprosessi tai –palvelin, joka on erikoitunut tunnistamaan järjestelmässä tapahtuvat laitteistovirheet. Tässä tehtävässä saattaa kuitenkin toimia mikä tahansa muukin järjestelmään kuuluva komponentti, joka toimii valvojana omien tehtäviensä lisäksi.

Silloin kun varsinainen komponentti toimii normaalisti, varakomponentti on odottavassa tilassa. Tälle odotustilalle on kaksi mahdollista tapaa: *kuuma* (hot standby) ja *kylmä* (cold standby) odotustila. Kuumassa tilassa varakomponentti toimii kuten varsinainenkin komponentti eli se suorittaa samat tehtävät. Tällöin se on heti valmis jatkamaan varsinaisen yksikön sijaisena, jos se vioittuu. Kylmässä tilassa varakomponentilla on yleensä jonkinlainen lokitiedosto, johon kirjataan varsinaisen komponentit suorittamat tehtävät. Saatuaan tiedon, että varakomponentti on nyt varsinainen komponentti, se suorittaa lokitiedostoon merkityt tehtävät ja jatkaa sitten toimintaansa. Yleensä

varakomponentti suorittaa lokitiedostoon siihen mennessä kertyneet merkinnät aina tietyn väliajoin, jotta joutuessaan korvaamaan varsinaisen komponentin, sen ei tarvitse suorittaa suurta määrää laskentaa ennen varsinaisen toimintansa jatkamista. Kummallakin odotustilamallilla on puolensa, mutta kuuma odotustila sopii paremmin reaaliaikaisiin järjestelmiin, jossa palvelujen saatavuus on jatkuttava heti, vaikka jokin komponentti lakkaisikin toimimasta.

4.2 Ohjelmisto viat

Hajautetussa järjestelmässä eri komponentit tarvitsevat usein toistensa palveluita suoriutuakseen niille annetuista tehtävistä. Tämä johtaa siihen, että palvelun suorituksessa saattaa ilmetä vikoja eri puolella järjestelmää, toisin kuin monoliittisessa järjestelmässä.

4.2.1 Perinteinen tapa

Perinteinen tapa käsitellä ohjelmistovikoja hajautetuissa järjestelmissä on naamioida ne poikkeuksiksi tai piilottaa ne kokonaan käyttäjältä. Esimerkiksi OSI:n kerrosmallissa kukin kerros joko piilottaa havaitsemansa vian ylemmältä kerrokselta tai naamioi sen muuttamalla vian poikkeukseksi. Tällaista menetelmää kutsutaan hierarkkiseksi naamioinniksi.

Hajautetussa järjestelmässä monet sovellukset käyttävät samoja resursseja. Tästä huolimatta sovelluksien tekemät muutokset halutaan pitää yhtenäisinä koko järjestelmän kannalta katsottuna. Transaktioksi sanotaan tapahtumaa, jossa kaikki sen aiheuttamat muutokset ovat pysyviä tai mikään niistä ei ole. Tätä ominaisuutta kutsutaan atomisuudeksi. Atomisuuden saavuttamiseksi on olemassa erilaisia sitoutumisprotokollia. Ne perustuvat yleensä ajatukseen, että transaktion aiheuttamat muutokset ovat alustavia, kunnes kaikki transaktioon liittyvät prosessit ovat ilmoittaneet sitoutuvansa kyseiseen transaktioon. Jos jokin prosessi ei sitoudu, transaktio perutaan ja koko touhu aloitetaan taas alusta.

Sekä hierarkkinen naamiointi että sitoutumisprotokollat ovat pahimmassa tapauksessa aikaavieviä. Ne saattavat soveltua käytettäväksi sellaisissa tilanteissa, kun järjestelmän reaaliaikavaatimukset eivät ole tiukkoja. Mutta ne ovat ehdottomasti soveltumattomia turvallisuus-kriittisiin järjestelmiin, joissa tapahtumien onnistuminen tai epäonnistuminen vaaditun aikarajan sisällä on elintärkeää. Seuraavissa kappaleissa esitellään kaksi mahdollista toipumismenetelmää turvallisuus-kriittisiin ympäristöihin: *korvaava toiminto* ja *hajautettu toipumislohko*.

4.2.2 Korvaava toiminto

Korvaava toiminto (compensating transaction) [KiL90] liittyy hajautettujen transaktioiden sitoutumisprotokollaan. Optimistisissa sitoutumisprotokollissa on aina olemassa mahdollisuus, että jokin myöhemmin sitoutumisesta ilmoittava prosessi päättääkin olla sitoutumatta. Tällöin myös aikaisemmin sitoutuneiden prosessien tekemät muutokset pitää perua. Jotta tällainen dominoefekti voitaisiin välttää, transaktioon voi kuulua sitoutumisen ja abortoinnin lisäksi myös korvaavan toiminnon suorittaminen. Jos jokin prosessi päättä abortoida kesken transaktion, se voi tehdä varsinaisen abortoinnin sijaan jonkin korvaavan toiminnon. Korvaava toiminto voi olla jotain sellaista, mikä lieventää varsinaista epäonnistumista. Tällaisen toiminnon avulla on tarkoitus välttää toisten prosessien pakottamista abortoimaan oma osuutensa tästä transaktiosta.

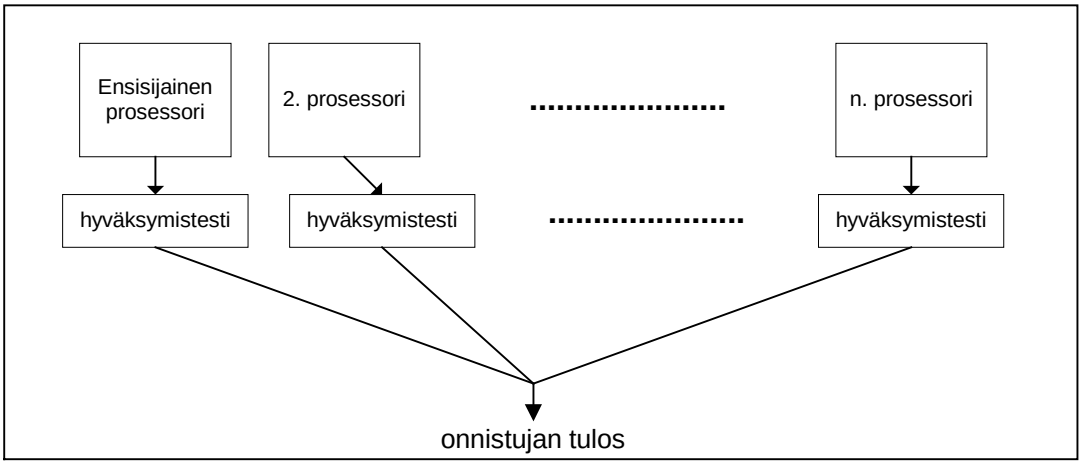
Riippuen sovellusalasta korvaavan toiminnon käyttö voi olla hyvin rajoitettua. Aina ei ole mahdollista käyttää mitään tehokasta korvaavaa toimintaa. Kuitenkin tällaisen menetelmän käyttäminen, aina kun se on mahdollista, soveltuu hyvin reaaliaikajärjestelmiin, sillä se vähentää transaktioiden peruuntumista ja uudelleenyritystä.

4.2.3 Hajautettu toipumislohko

Hajautettu toipumislohko (distributed recovery block, DRB) [KiL90] on Kane Kim:n esittämä vikasietoisuusmenetelmä, joka perustuu Randell:n alunperin kuvaamaan toipumislohko –tekniikkaan (RB). Toipumislohkon syntaktinen esitys on seuraavanlainen: *varmista* T:llä B_1 *else* B_2 … *else* B_n *else virhe*. T kuvaa hyväksymistestiä ja B_n:t

ovat vaihtoehtoisia ohjelman versioita eli toipumislohkoja. Kaikki lohkot tuottavat saman tai samanlaisen laskennallisen tuloksen ja tuloksen oikeellisuus tarkastetaan hyväksymistestillä T. Toipumislohkoja suoritetaan peräkkäin, kunnes saadaan hyväksyttävä lopputulos. Näin ollen ensimmäisen lohkon epäonnistuessa siirrytään seuraavaan jne.

Hajautetussa toipumislohko –menetelmässä edellä kuvatut toipumislohkot voidaan suorittaa rinnakkain eri prosessoreilla, joista yksi toimii ensisijaisena prosessorina (kuva 3). Prosessorien lisäksi apuna käytetään ajastimia, jotta pysytään annettujen aikarajoitusten puitteissa. Kukin prosessori tekee hyväksymistestin suoritettuaan varsinaisen lohkon. Prosessorit vaihtavat keskenään tietoja laskennan onnistumisesta tai epäonnistumisesta. Jos ensisijaisen prosessorin hyväksymistesti on onnistunut, vain sen tuottamat tulokset palautetaan. Seuraavan prosessorin tuloksia käytetään vain, jos ensisijainen prosessori ei pysty tuottamaan hyväksyttävää tulosta annetussa ajassa. Koska prosessorit suorittavat laskentansa samanaikaisesti, hajautetun toipumislohko –tekniikan vaatima aika pahimmassa mahdollisessa tapauksessa on huomattavasti pienempi kuin perinteisessä toipumislohko –tekniikassa.



Kuva 3: Hajautetun toipumislohkon periaate

5 Sopeutuva vikasietoisuus

Sopeutuvan vikasietoisuuden tarkoitus on ottaa dynaamisesti huomioon järjestelmän toimintaympäristö ja sopeuttaa kulloinenkin vikasietoisuuspolitiikka tähän ympäristöön.

5.1 Motivointi

Korkea vikasietoisuus on sinällään erittäin tavoiteltava ominaisuus. Kuitenkin luotettavuuden kasvattaminen tarkoittaa yleensä myös järjestelmän monimutkaisuuden ja vasteaikojen kasvamista, mikä ei ole tavoiteltavaa, etenkään reaaliaikaisissa järjestelmissä. Perinteisesti tätä ongelmaa on lähestytty niin, että järjestelmälle asetetaan jo suunnitteluvaiheessa vaatimukset eri palvelujen vasteajoista sekä rajataan tarkasti, minkälaisista virheistä järjestelmän tulee pystyä toipumaan. Eli järjestelmälle suunnitellaan hyvin tarkkaan vikasietoisuuspolitiikka ja tämän perusteella valitaan esimerkiksi millaista toipumisalgoritmia järjestelmässä käytetään. Tämä johtaa kuitenkin tarpeettomaan jäykkyyteen ja ehkä jopa liiankin tiukkaan vikasietoisuuspolitiikkaan joidenkin järjestelmän tarjoamien palvelujen osalta.

Koska hajautetut reaaliaikaiset järjestelmät toimivat usein nopeasti vaihtuvassa ympäristössä, myös vikasietoisuusvaatimukset muuttuvat. Niinpä järjestelmän pitäisikin pystyä mukautumaan näihin uusiin vaatimuksiin ja tarjota erilaisia ratkaisuja eri tilanteisiin. Esimerkiksi lentokoneessa toimiva tutka saattaa toimia aivan moitteettomasti, kunnes kone joutuu yllättäen keskelle valtavaa lintuparvea. Tällöin järjestelmää ohjaavat sensorit saavat nopeassa tahdissa paljon syötteitä ja järjestelmä saattaa tukkeutua, jos se ei pysty käsittelemään tällaista äkillistä tietotulvaa ja reagoimaan siihen kyllin nopeasti. Olosuhteet, joissa järjestelmä toimii, saattavat siis muuttua hyvin nopeasti. Jos järjestelmää suunnitellessa ei ole varauduttu tällaisiin odottamattomiin tilanteisiin, seuraukset saattavat olla vakavia. Jos alunperin on varauduttu pahimpaan mahdolliseen vaihtoehtoon, suuri osa järjestelmän resursseista saattaa olla suurimman osan ajasta pois hyödyllisestä käytöstä. Reaaliaikaisissa järjestelmissä pitäisikin olla mahdollisuus vaihtaa vikasietoisuuspolitiikkaa tarpeen mukaan. Sopeutuva vikasietoisuus tarjoaa juuri tällaisen mahdollisuuden.

5.2 Toimintaperiaate

Sopeutuvan vikasietoisuuden perusidea on, että se pystyy monitoroimaan järjestelmän toimintaympäristöä ja dynaamisesti valitsemaan sopivan vikasietoisuuspolitiikan [GLL93]. Tässä se eroaa perinteisestä menetelmästä, jossa vikasietoisuuspolitiikka on päätetty jo järjestelmän suunnitteluvaiheessa ja jonka pitää toimi kaikissa mahdollisissa tilanteissa. Käytännössä järjestelmässä toimii yleensä yksi tai useampi vikasietoisuusmanageri, joka tutkii järjestelmän tilaa ja huolehtii vikasietoisuuspolitiikan vaihdosta. Sopivan vikasietoisuuspolitiikan valintaan vaikuttavat järjestelmän toimintaympäristön eri tilat ja järjestelmässä esiintyvien vikojen tyypit.

5.2.1 Toimintaympäristön tyypit

Tärkein toimintaympäristön piirre vikasietoisuuspolitiikan kannalta liittyy vasteaikojen kriittisyyteen. Tämän perusteella voidaan määritellä kaksi toimintaympäristön tilaa: kova reaaliaikaisuus ja pehmeä reaaliaikaisuus. Kovassa reaaliaikaisuudessa annetut vasteajat ovat ehdottomia ja niiden huomioonottamisen laiminlyönnistä voi tulla vakavia seurauksia. Tällaiseen tilaan ei sovi aikaavievät diagnostiikka- ja toipumisalgoritmit. Pehmeässä reaaliaikaisuudessa ei ehdottomia vasteaikoja ole. Hajautetut järjestelmät ovat nykyisin hyvin heterogeenisiä ja niinpä jokin järjestelmän saattaa toimia pehmeässä reaaliaikaisuuden tilassa ja jokin toinen taas kovassa reaaliaikaisuuden tilassa. Tai jokin järjestelmä saattaa vuorotella näiden kahden tilan välillä.

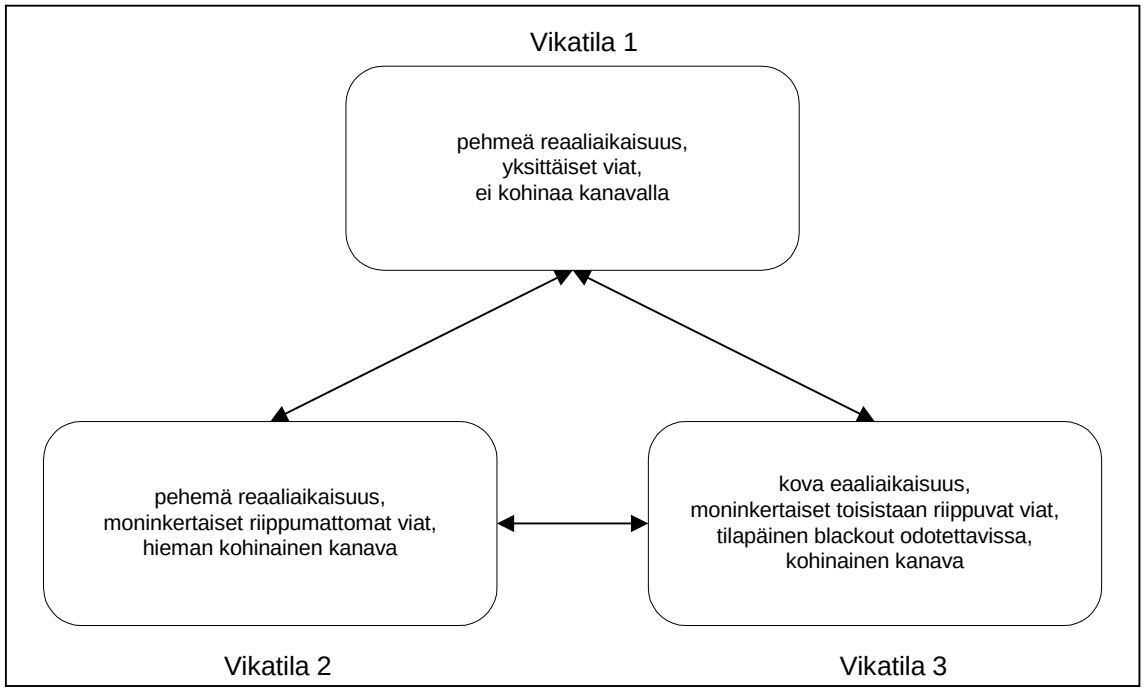
5.2.2 Vikamallit

Jotkin järjestelmässä ilmenevät viat ovat pelkästään sisäisen toiminnan aiheuttamia, toiset taas saattaa aiheuttaa jokin ulkopuolinen toiminta. Vikamalleja voidaan luokitella seuraavasti [KiL90]:

- määrä
  - yksittäinen vika
  - moninkertaiset toisistaan riippumattomat viat
  - moninkertaiset toisistaan riippuvat viat
  - tilapäinen blackout

- jatkuvuus
  - pysyvä
  - ohimenevä
  - ajoittainen
- esiintymistiheys
  - satunnainen
  - puuskittainen
- lähde
  - laitteisto (prosessori, levy, kommunikointikanava)
  - ohjelmisto

Kukin näistä ryhmittelyistä vaatii omat toipumistekniikkansa. Esimerkiksi jos ollaan tilassa, jossa vain yksittäisiä vikoja esiintyy, on turha käyttää menetelmiä, jotka osaavat käsitellä monikertaisia toisistaan riippuvia vikoja. Kuva 4 esittelee esimerkin erilaisten tilojen joukosta. Kutakin tilaa varten on olemassa oma vikasietoisuuspolitiikkansa, joka otetaan käyttöön kun järjestelmä siirtyy tilasta toiseen.



Kuva 4: Esimerkki järjestelmän vikatiloista

5.2.3 Diagnostiikka

Sopeutuva vikasietoisuus on pohjimmiltaan kontrollisysteemi, joka pyrkii toimintaympäristöstä keräämiensä tietojen avulla päättämään, mikä on kulloinkin optimaalinen vikasietoisuuspolitiikka. Diagnostiikka on oleellinen osa tätä päätöksentekoa.

Perinteisessä vikasietoisuusmallissa vikadiagnostiikan tarkoitus on tunnistaa esiintyvän vian luonne ja sijainti, sekä ilmoittaa tällaisen vian havaitsemisesta. Sopeutuvassa vikasietoisuudessa vikadiagnostiikalla on laajempi rooli. Sen täytyy tunnistaa mahdollinen konflikti nykyisen vikasietoisuuspolitiikan ja järjestelmässä vallitsevan toimintaympäristön välillä. Tämä on paljon vaikeampi tehtävä kuin miltä se kuulostaa. Yleensäkin kontrollisysteemien tekemät päätökset perustuvat tietoon nykyisestä tilasta sekä arvaukseen tulevasta. Eli mukana on aina jonkin verran epävarmuutta johtuen havaintojen ja ennustusmallin epätarkkuudesta. Reaaliaikaisissa järjestelmissä tämä epävarmuus korostuu, sillä tarjolla on yleensä hyvin rajoitettu määrä tietoa siinä aikarajassa, jossa päätös pitää tehdä.

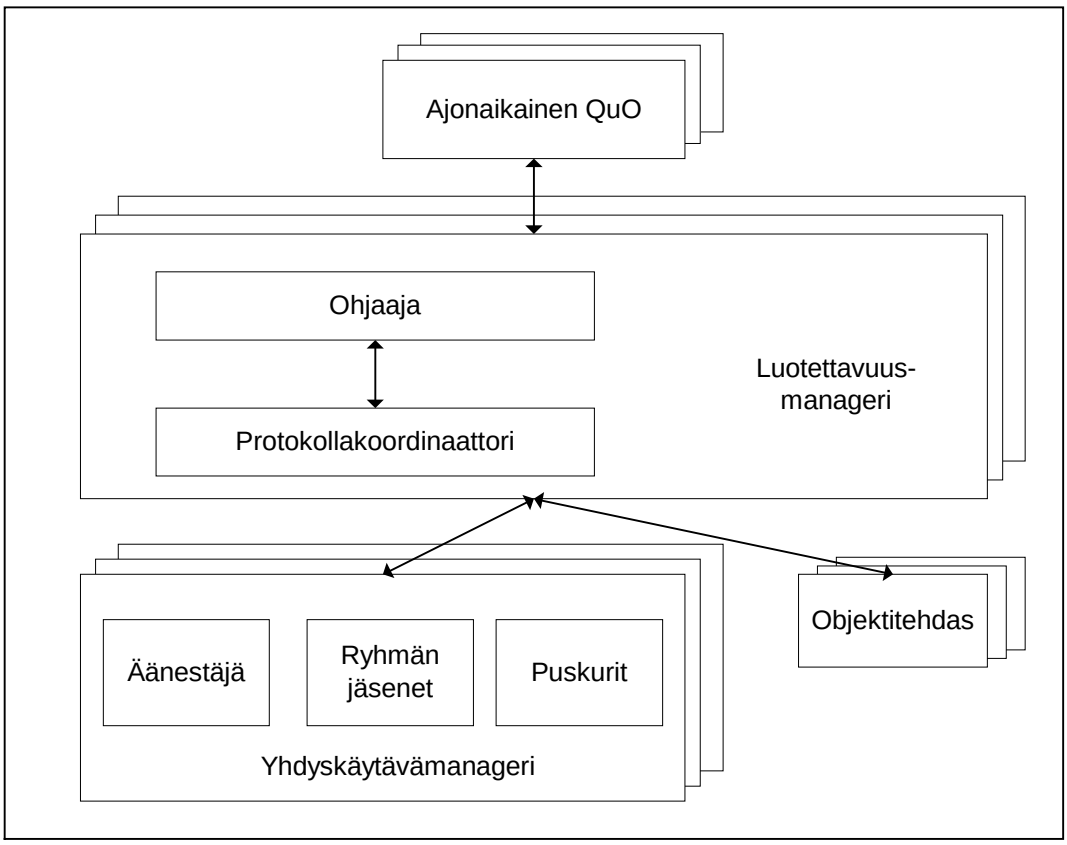
5.3 Proteus ja ROAFTS

Tässä luvussa esitellään kaksi hajautetuissa järjestelmissä toimivaa sopeutuvan vikasietoisuuden tarjoavaa arkkitehtuuria: Proteus ja ROAFTS. Kummankin rakenne ja toimintaidea käydään läpi siltä osin, kuin ne vaikuttavat vikasietoisuuden toteuttamiseen.

5.3.1 Proteus

Proteus [Sab99] on AQuA –arkkitehtuuriin (Adaptive Quality of service Availability) rakennettu menetelmä, joka tarjoaa hajautetuille sovelluksille mahdollisuuden saada haluamansa tasoista palvelun laatua (quality of service, QoS). Tämä mahdollistetaan tarjoamalla sovellukselle laatuobjekteja (Quality Object, QuO), jotka ovat eräänlaisia sopimuksia sovelluksen ja sen alla toimivan järjestelmän välillä. Näillä sopimuksilla määritellään halutun palvelun taso. Sopimuksien avulla sovellus voi esimerkiksi vaatia järjestelmältä tietyntyyppistä vikasietoisuutta ja muuttaa näitä vaatimuksiaan aina tarpeen

tullen. Proteus ei sinällään ole suunniteltu erityisesti reaaliaikaisia järjestelmiä varten, mutta sopimusten avulla sovellukset voivat määritellä myös vasteaikoja haluamilleen palveluilleen. Kuva 5 esittelee Proteuksen rakenteen. Proteus koostuu kolmesta keskeisestä osasta: luotettavuusmanagerista (dependability manager), yhdyskäytävästä (gateway) ja objektitehtaasta (object factory).



Kuva 5: Proteuksen rakenne

Luotettavuusmanageri jakautuu kahteen osaan: ohjaajaan ja protokollakoordinaattoriin. Ohjaaja tekee päätökset, jotka koskevat järjestelmän uudelleenkonfigurointia. Nämä päätökset perustuvat järjestelmän ilmoittamiin vikoihin ja sovelluksilta tulleisiin laatuvaatimuksiin. Protokollakoordinaattori huolehtii yhdessä yhdyskäytävän kanssa valitun vikasietoisuuden toteuttamisesta. Proteus pystyy toipumaan järjestelmään kuuluvien komponenttien kaatumisista, ajoitusvirheistä ja sovelluksissa esiintyvistä virheellisistä arvoista.

Yhdyskäytävällä on kaksi tehtävää. Ensinnäkin, se tarjoaa sovelluksille CORBA –rajapinnan. Yhdyskäytävä tulkkaa prosessitason ja objektitason kommunikaatiota kääntämällä ORB:n (Object Request Broker) ymmärtämiä IIOP sanomia AquA:n ymmärtämään muotoon ja päinvastoin. Yhdyskäytävän toinen tehtävä on raportoida järjestelmässä ilmenevät viat luotettavuusmanagerille.

Kussakin järjestelmän solmussa toimii yksi objektitehdas. Objektitehtaan tehtävänä on käynnistää ja lopettaa prosesseja sekä tarjota tietoa isännästään luotettavuusmanagerille.

5.3.2 ROAFTS

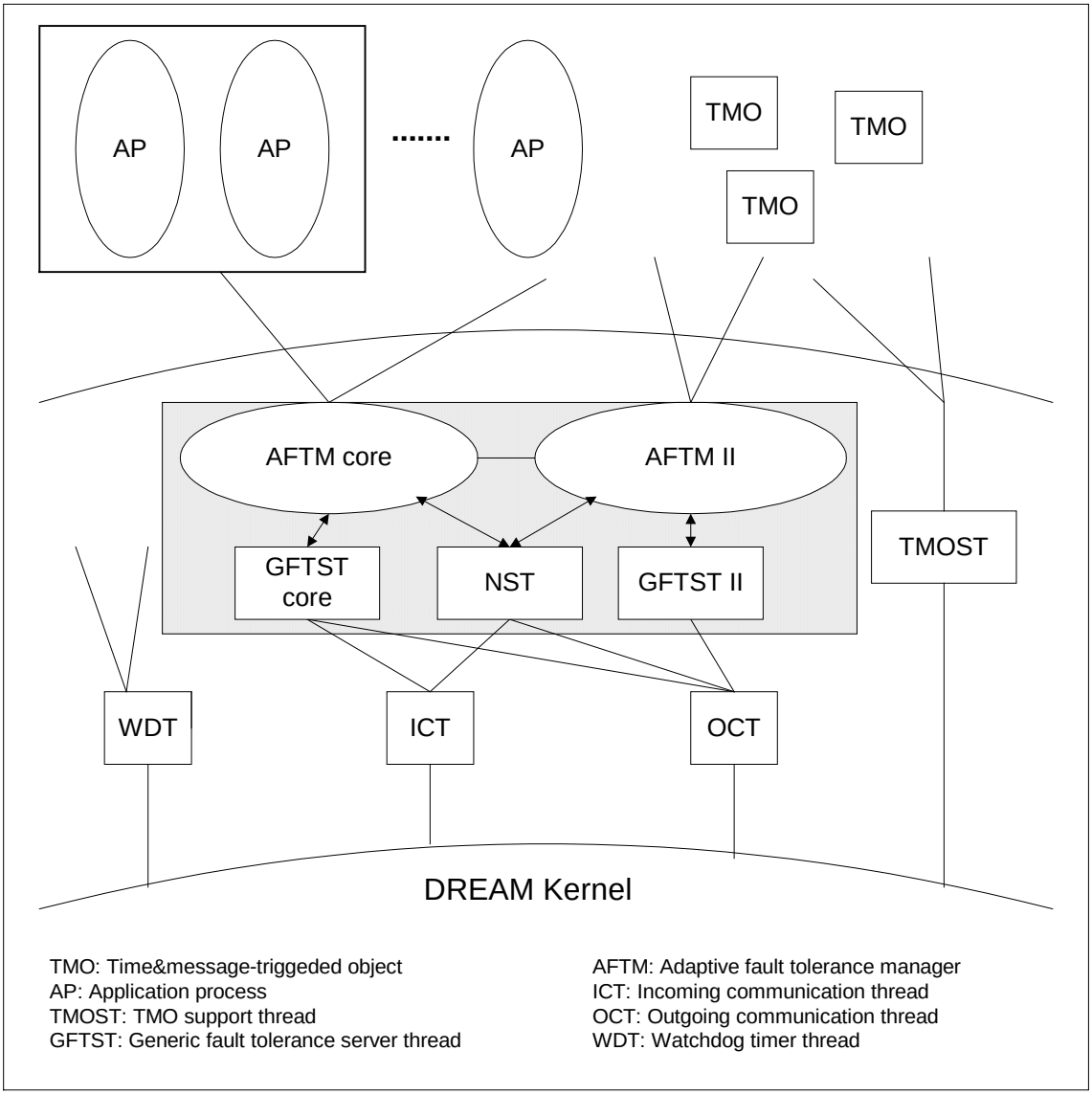
ROAFTS [Kim98] (Real-time Object-oriented Adaptive Faul Tolerance Support) on välitason arkkitehtuuri, joka tarjoaa sopeutuvan vikasietoisuuden sekä perinteisille prosessikeskeisille, että uudemmille objektipohjaisille hajautetuille reaaliaikaisille järjestelmille. ROAFTS tarjoaa erilaisia vikasietoisuus –vaihtoehtoja. Näihin kuuluvat mm. vikasietoiset laskentayksiköt sekä tietokoneverkon seuranta ja uudelleenkonfigurointi.

Vikasietoisia laskentayksiköitä on kahdenlaisia: yksinkertaisia (simplex station) ja kaksinkertaisia (duplex station). Yksinkertaisissa laskentayksiköissä käytetään toipumisalgoritmeina esimerkiksi taaksepäin sitoutuvia transaktioiden sitoutumisprotokollia. Nämä laskentayksiköt toimivat yleensä pehmeän reaaliaikaisuuden tilassa. Kaksinkertaisissa laskentayksiköissä käytetään yleensä hajautettu toipumislohko –menetelmää. Kaksinkertaiset laskentayksiköt suorittavat kovan reaaliaikaisuuden vaativia tehtäviä.

Tietokoneverkon seurannan tarkoituksena on löytää nopeasti hajautetun järjestelmän vikaantuneet osat ja suorittaa tarvittavat toipumistoimenpiteet verkon toimivuutta ajatelleen. ROAFTS:ssa käytetään valvontaan perustuvaa verkon seurantaa (supervisor-based network surveillance, SNS). SNS –mallissa verkossa on kahdenlaisia solmuja: työläinen ja valvoja. Työläiset ovat oman toiminnallisuutensa oikeellisuuden lisäksi vastuullisia seuraamaan naapureidensa tilannetta, sekä itseensä liittyviä

kommunikointilinkkejä. Valvojalla on samanlaiset tehtävät kuin työläiselläkin, mutta sen lisäksi sen tehtävänä kerätä tietoa verkkoon kuuluvilta työläisiltä mahdollisten vikojen varalta. Näin saatujen *vikaepäilyraporttien* perusteella valvoja pystyy päättelemään, mikä osa verkosta on vian aiheuttaja. Vian paikannettuaan, valvoja lähettää vikailmoituksen kaikille verkon toimiville työläisille. Jos valvoja vikaantuu, sen naapurit huomaavat tämän ja käynnistävät vaalin uuden valvojan valitsemiseksi.

ROAFTS:sta on rakennettu kolme prototyyppiä. Ensimmäinen toimii DREAM kernelin päällä, toinen Sunin Solaris –alustalla ja kolmas edelleen kehitteillä oleva toimii Microsoftin NT –alustalla. Kuva 6 esittelee ROAFTS:n perusarkkitehtuurin. Tummalla varjostettuun alueeseen kuuluvat komponentit vastaavat sopeutuvan vikasietoisuuden toteuttamisesta. NST (Network surveillance thread) vastaa tietokoneverkon seurannasta. GFTST (Generic fault tolerance support thread) tarjoaa sovelluksille tuntumattoman vikasietoisuuden. Sen tehtävänä on hoitaa kaikkiin vikasietoisuuspolitiikkoihin liittyvät yksityiskohdat. AFTM ytimen (Adaptive fault tolerance manager -core) on järjestelmäprosessi, jonka tehtävänä on sopeuttaa järjestelmän vikasietoisuustoiminta sen tilanteen mukaan, mikä kulloinkin on vallalla.



Kuva 6: ROAFTS arkkitehtuuri

6 Yhteenveto

Vikasietoisuus on erittäin tärkeä reaaliaikaisten hajautettujen järjestelmien ominaisuus. Kaikkia mahdollisia vikoja ei voida ottaa huomioon järjestelmää suunniteltaessa, mutta on tärkeää, että järjestelmä ei kaadu totaalisesti, kun siinä ilmenee jokin vika. Varsinkin turvallisuus-kriittisissä järjestelmissä on tärkeää, että järjestelmä suoriutuu mallikkaasti sitä kohtaavista virheistä.

Hajautetuissa järjestelmissä ilmeneviin vikoihin voidaan varautua erilaisilla vikasietoisuuspolitiikoilla. Vikasietoisuuspolitiikat määritellään sen mukaan, millaisia virhekategorioita järjestelmässä odotetaan ilmenevän ja mihin näistä virheistä järjestelmän pitää pystyä varautumaan. Sopivan vikasietoisuuspolitiikan etsintä sisältää aina kompromisseja. Mitä korkeammaksi järjestelmän vikasietoisuutta nostetaan, sitä enemmän se vaatii resursseja. Reaaliaikajärjestelmissä palveluille asetetut vasteajat rajoittavat toipumismenetelmiin kuluvaa aikaa, joten yleensä tarvitaan selkeä rajanveto, miten pitkälle vikasietoisuus tällaisissa järjestelmissä viedään.

Nykyään hajautetut järjestelmät toimivat mitä moninaisimmissa ympäristöissä. Tästä seuraa, että vikasietoisuudenkin pitää sopeutua toimimaan erilaisissa ympäristöissä. Sopeutuvan vikasietoisuuden avulla järjestelmään voidaan sisällyttää useita vikasietoisuuspolitiikoita, joita voidaan vaihdella aina tarpeen vaatiessa. Näin vältytään tuhlaamasta järjestelmän resursseja ylimitoitetulla vikasietoisuudella ja taataan joustavaa palvelua sovellusasiakkaille.

Lähteet

[CoD98] Coulouris G., Dollimore J., Kindberg T., *Distributed systems concepts and design*. Addison Wesley, England 1998.

[Cri91] Cristian F., Understanding Fault-Tolerant Distributed Systems. *Communications of ACM*, vol. 34, no 2, 1991.

[KiL90] Kim K.H., Lawrence T.F., Adaptive fault tolerance: issues an approaches. *Proceedings of the 2nd IEEE Workshop on Future Trends of Distributed Computing Systems*, Cairo, 1990, 38 – 46.

[Kim98] Kim K.H., ROAFTS: a middleware architecture for real-time object-oriented adaptive fault tolerance support. *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*, Washington DC, 1998, pp 50 – 57.

[GGL93] Goldberg J., Greenberg I., Lawrence T.F., Adaptive Fault Tolerance. *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems,* Princeton, 1993, pp 127 – 132.

[Sab99] Sabnis C., Cukier M., Ren J., Rubel P., Sanders W.H., Bakken D.E., Karr D., Proteus: a flexible infrastructure to implement adaptive fault tolerance in AQuA. *Dependable Computing for Critical Applications*, vol 7, 1999, pp 149 – 168.