

Jaetun muistin muuntaminen viestin välitykseksi

Otto Räsänen

15. lokakuuta 2007

1 Motivaatio

2 Valtuuden välitys

- Peruskäsitteitä

3 Kolme algoritmia

- Valtuuden välitys käyttäen laskuria ilman ylärajaa
- Valtuuden välitys, kun jonojen pituudella on yläraja
- Valtuuden välitys käyttäen satunnaisia tunnistenumeroita

4 Jaetun muistin simulointi

- Jaetun muistin simulointi käyttäen itsestabiloituvaa valtuuden välitystä

Miksi jaettu muisti halutaan pystyä muuntamaan viestin välitykseksi?

- Monet itsestabiloituvat algoritmit, kuten Dijkstran algoritmi, käyttävät jaettua muistia.
- Suunnittelu helpompaa, jos käytetään jaettua muistia.
- Käytännöllistä suunnitella algoritmi yhteen järjestelmään ja muuntaa se toimimaan toisissa.

Valtuuden välitys

- Kaksi suoritinta. Vain toisella voi olla yhdellä ajan hetkellä valtuus.
- Valtuus siirretään suorittimelta toiselle käyttäen viestin välitystä.
- Ratkaisee saman ongelman kuin keskinäinen poissulkeminen.
- Voidaan käyttää myös tiedon siirtoon.

Valtuuden hallussapito

- Lähettäjällä on valtuus, kun se luo uuden tunnistenumeron.
- Lähettää samaa viestiä uudelleen, kunnes saa kuittauksen samalla tunnistenumeroilla.
- Vastaanottajalla on valtuus, kun se vastaanottaa "uuden" tunnistenumeron.
- Kuittaa jokaiseen viestiin viimeisimmän valtuuden tunnistenumeroilla.

Turvallinen konfiguraatio

- Konfiguraatio: Sisältää suorittimien tilat sekä jonoissa matkalla olevat viestit.
- Turvallinen konfiguraatio: suorittimien laskurien arvot sekä viestien tunnistenumerot yhtä suuret.
- Itsestabiloituva algoritmi pääsee turvalliseen konfiguraatioon mistä tahansa alkutilasta.
- Turvallisesta k :sta alkaen järjestelmä suorittaa vain "laillisia" toimintoja.

Sender:

```
01  upon timeout
02      send(counter)
03  upon message arrival
04  begin
05      receive(MsgCounter)
06      if MsgCounter ≥ counter then
07          (* token arrives *)
08          begin (* send new token *)
09              counter := MsgCounter + 1
10              send(counter)
11          end
12      else send(counter)
13  end
```

Receiver:

```
13  upon message arrival
14  begin
15      receive(MsgCounter)
16      if MsgCounter ≠ counter then
17          (* token arrives *)
18          counter := MsgCounter
19      send(counter)
20  end
```

- Kun järjestelmä on turvallisessa konfiguraatiossa, niin seuraavan laskenta-askeleen jälkeinen konfiguraatio on myös turvallinen.
- Kaikista konfiguraatioista päädytään turvalliseen konfiguraatioon, kunhan suoritus on reilu.

- Jonojen pituudelle ei aseteta mitään ylärajaa.
- Siten laskurillakaan ei voi olla suurinta arvoa.
- Tästä seuraa, että algoritmin vaatima muistin määrä kasvaa jatkuvasti.

Edellisen algoritmin variaatio

- Oletetaan, että jonojen pituudella yläraja. Merkitään ylärajaa vakiolla cap .
- Kasvatetaan laskuria modulo $cap + 1$.
- Ennen pitkää valitaan sellainen tunnistenumero, jota ei vielä ole käytössä. (Kuten Dijkstran algoritmossa.)
- Algoritmi on itsestabiloituva.

Sender:

```
01  upon timeout
02      send(label)
03  upon message arrival
04  begin
05  receive(MsgLabel)
06      if MsgLabel = label then
           (* token arrives *)
07          begin (* send new token *)
08              label := ChooseLabel(MsgLabel)
09              send(label)
10          end
11      else send(label)
12  end
```

Receiver:

```
13  upon message arrival
14  begin
15      receive(MsgLabel)
16      if MsgLabel ≠ label then
           (* token arrives *)
17          label := MsgLabel
18      send(label)
19  end
```

Satunnaisalgoritmi

- Arvotaan tunnistenumero, jonka pitää poiketa edellisestä.
- Erilaisia tunnistenumeroita tarvitaan vähintään 3.
- Algoritmi stabiloituu todennäköisyydellä 1, kun suoritus reilu.
- Äärellisen pitkässä suorituksessa turvallista konfiguraatiota ei välttämättä saavuteta.

- Jonossa peräkkäin olevia viestejä, joilla on sama tunnistenumero, kutsutaan segmentiksi.
- Lähetetään samaa viestiä niin kauan, kunnes tulee kuittaus samalla tunnistenumerolla.
- Kelpaamattomat kuittaukset "poistuvat" jonosta, joten segmenttien lukumäärä vähenee.
- Turvallinen konfiguraatio saavutetaan $O(n \log n \cdot 2^{2m})$ kierroksessa, missä n on viestien määrä jonoissa ja m segmenttien määrä suorituksen alussa.

Kommunikointi käyttäen jaettua muistia

- Kaksi suoritinta, P_i ja P_j , jakavat kaksi yhdensuuntaista rekisteriä.
- Luku- ja kirjoitusoperaatiot atomisia: yhden askeleen aikana suoritetaan vain yksi operaatio.
- Suoritin P_i kirjoittaa rekisteriin r_{ij} ja lukee rekisteristä r_{ji} .
- P_j käyttää rekistereitä toiseen suuntaan.

Tiedon siirto käyttäen valtuuden välitystä

- Sovitaan, kumpi suorittimista toimii lähettäjänä ja kumpi vastaanottajana.
- Valtuuden mukana tietoa siirtyy kumpaankin suuntaan, joten kahden suorittimen välillä vain yksi instanssi algoritmista.
- Tarvitaan kuitenkin niin monta instanssia kuin on naapureita.

Paikalliset muuttujat

- Suorittimella P_i on paikallinen muuttuja R_{ij} , johon tallennetaan rekisteriin r_{ij} viimeksi kirjoitettu arvo.
- Vain rekisteriin kirjoittava suoritin ylläpitää viimeisintä arvoa muuttujassa.
- Viimeisin arvo lähetetään valtuuden mukana lukijalle.
- Suoritin P_i lähettää siis R_{ij} :n arvon P_j :lle, ja P_j vastaavasti R_{ji} :n arvon P_i :lle.

Simulointi käytännössä

Rekisteriin r_{ij} kirjoittaminen

- Suoritin P_j kirjoittaa arvon paikalliseen muuttujaansa R_{ij} .

Rekisteristä r_{ij} lukeminen

- Suoritin P_j vastaanottaa valtuuden.
- Suoritin P_j vastaanottaa valtuuden uudelleen. Lukuoperaation tulos on tämän valtuuden yhteydessä vastaanotettu R_{ij} :n arvo.

Kaksivaiheinen lukuoperaatio

- Lukuoperaation aikana suoritin ei jatka simuloitavan ohjelman suorittamista.
- Valtuuden välitystä kuitenkin jatketaan muiden suorittimien kanssa.
- Valtuus vastaanotettava kahdesti, jotta luettu arvo ei voisi olla vanhempi, kuin rekisteriin ennen lukuhetkeä kirjoitettu arvo.

Esimerkki

