Helsingin yliopisto
Tilastotieteen johdantokurssin essee
2.1.2007
Jarmo Vestola

Poikkeusten sieppaamisen suoritusnopeus Java-ohjelmassa

1 Johdanto

Java on Sun Microsystemsin 90-luvulla kehittämä ohjelmointikieli, joka on saavuttanut suuren suosion sen laitteistoriippumattomuuden ja oliopohjaisuuden takia. Vahvan tyypityksen ja automaattisen roskienkeruun ansiosta Java-ohjelmat ovat myös turvallisia, mutta samalla ehkä hivenen hitaampia kuin muilla ohjelmointikielillä kirjoitetut ohjelmat.

Java-ohjelmoinnissa poikkeuksilla tarkoitetaan ”poikkeuksellisia tilanteita” kesken ohjelman normaalin suorituksen. Tällaisia ovat muun muassa tiedoston loppuminen tai taulukon indeksin ohi viittaaminen. Ohjelmointikielestä riippuen tällaisiin poikkeuksiin reagoidaan eri tavalla. Toisissa ohjelmointikielissä poikkeuksen aiheuttama virhe lopettaa ohjelman heti, eli toisin sanoen ”kaataa” ohjelman. Toisissa ohjelmointikielissä taas ohjelma jatkaa virheestä huolimatta toimintaansa kohti mahdollista katastrofia, jolloin seuraukset voivat olla pahoja. Java-ohjelmoinnissa on mahdollisuus käyttää kolmatta tapaa, jolla ohjelmoija voi itse ohjelmoida käsittelyn näiden virhetilanteiden varalle ja saada ohjelmat toimimaan parhaalla tavalla poikkeuksen sattuessa.

Yksi tällainen tapa Javassa on niin sanottu try-catch-lause, jossa try-lohkossa suoritetaan koodia, joka voi aiheuttaa poikkeuksen. Try-lohko lopetetaan heti, jos poikkeus syntyy ja ohjelmakoodin suoritus siirtyy catch-lohkoon, jossa poikkeus siepataan ja käsitellään. Koska tällaisia try-catch-lauseita on yksinkertaista tehdä ja ne yleensä vaikuttavat virheiden ratkaisussa näppäriltä oikoteiltä, tulee niitä helposti tehtyä liikaa myös kohtiin, jossa ne voisi helposti välttää käyttämällä esimerkiksi ehto-lauseita. Eräs tällainen ehto-lause on if-else-lause, jossa ehdon totuusarvo tarkastetaan ennen varsinaisen lohkon suoritusta. Jos ehto on tosi, lohko suoritetaan. Epätosi ehto siirtää suorituksen else-osaan

Try-catch-lauseiden heikko kohta on niiden hidas suorittaminen. Poikkeusten käyttäminen tavallisten tilanteiden tarkastamiseen (ei siis "poikkeuksellisen vakaviin" tilanteisiin) ei välttämättä ole järkevää, koska näihin varautuminen try-catch-lauseilla saattaa hidastaa ohjelman suorittamista merkittävästi. Tavallisten tarkistusten hoitamiseen kannattaa siis käyttää esimerkiksi ehtolauseita, kuten if-else-, tai while-lauseita.

2 Testien tarkoitus ja kuvaus

Tässä tilastotieteen esseessä on tarkoitus testata tilastollisesti, miten tällainen try-catch-lause vaikuttaa tavallisen Java-ohjelman suoritusnopeuteen verrattuna if-lauseen nopeuteen suorittaa sama tilanne. Testi käsittää kaksi erillistä koetta, jossa ensimmäisessä ei aiheuteta yhtäkään poikkeusta ja toisessa 1% indeksoinneista aiheuttaa poikkeuksen. Testi tehdään Applen iBook G4 1,33 GHz tietokoneella, joka on nykyisiin PC-tietokoneisiin verrattuna hieman hidas, joten se sopii hyvin tähän testiin.

Testissä ajetaan liitteenä olevaa Java-koodia Eclipse-ohjelman kautta. Koodin periaate on yksinkertaisesti sijoittaa halutun kokoisen taulukon indekseille arvoja suuruusjärjestyksessä ja laskea tähän kulunut aika. Saaduista ajoista lasketaan keskiarvo sekä hajonta ja suoritetaan kahden riippumattoman otoksen t-testi, josta saadaan p-arvo. Lopullisista tuloksista tehdään johtopäätös, joka kertoo milloin try-catch-lauseita on järkevä käyttää if-lauseiden sijaan.

Testin hypoteesit ovat:
H0: Try-catch-lauseella ja if-lauseella ei ole suoritusnopeudessa eroa.
H1: Try-catch-lauseella ja if-lauseella on suoritusnopeudessa eroa.
Testin lähtökohdaksi otetaan merkitsevyystaso 0,01. Tätä pienemmillä p-arvoilla H0 voidaan hylätä.

2.1 Ensimmäinen koe ilman poikkeuksia

Ensimmäisessä kokeessa verrataan try-catch-lauseen ja if-lauseen nopeuseroja ohjelmakoodilla, jossa yhtään poikkeusta ei tapahdu. Tässä t-nimisen taulukon koko on miljoona ja taulukon indekseille annetaan myös arvoja yhdestä miljoonaan. Toistolause käy siis jokaisen taulukon indeksin läpi, sijoittaen sille arvon, joka on yhtä suuri kuin indeksin järjestysluku. Taulukon ”nollas” indeksi saa siis arvokseen 0, ensimmäinen saa arvokseen 1, toinen saa arvokseen 2 jne.

Tämä koodi suoritetaan 10 000 kertaa, kuten myös toinen koe, jotta tietokoneen muiden sovellusten satunnainen laskentateho ei häiritsisi testiä. Tällä tavoin saadaan myös suoritusaikaa kasvatettua millisekunneista sekunteihin, jolloin pienetkin suorituserot saadaan näkyviin. 10 000 suorituskerran jälkeen lasketaan ohjelman suoritukseen kulunut loppuaika muunnettuna lähimpään sekuntiin. Tästä vaiheesta muodostuu ensimmäinen ajokerta, joita tehdään yhteensä kymmenen. Tässä ensimmäisessä kokeessa poikkeuksia ei siis tule, joten catch-lausetta ei jouduta suorittamaan kertaakaan, eikä suoritusnopeudessa pitäisi välttämättä tällöin olla merkittäviä eroja if-lauseeseen verrattuna.

Ensimmäisestä kokeesta tehtiin siis erikseen kymmenen testiajoa, joiden tulokset taulukoitiin. Kokeen tuloksia käytettiin kahden riippumattoman otoksen t-testiin (oletus: samat hajonnat), joka on mahdollista suorittaa osoitteessa: http://www.graphpad.com/quickcalcs/ttest1.cfm.
Ensimmäisestä kokeesta laskettiin siis GraphPad Softwaren nettilaskimella otoskeskiarvo ja -hajonta, t-arvo ja p-arvo. (Oletuksena laskimessa oli siis samat hajonnat, mutta testi tehtiin myös käyttäen toisessa laskimessa eri hajontoja. Tällöin tulokset olivat lähes samat).

2.2 Toinen koe poikkeusten kanssa

Toinen koe on muuten samanlainen kuin ensimmäinen, mutta pienemmillä numeroilla, koska oletettu laskenta-aika kasvaisi muuten liian suureksi. Nyt indekseille yritetään sijoittaa arvoja nollasta 100 000:een, taulukon koon ollessa 99 000. Tämä tarkoittaa siis sitä, että 1% sijoituslauseista aiheuttaa poikkeuksen.

Toistolause toimii taas siten, että taulukon ”nollas” indeksi saa arvokseen 0, ensimmäinen = 1, toinen = 2 jne. Taulukko loppuu kuitenkin indeksiin 99 000, ja kaikki tätä suuremmat sijoituslauseet aiheuttavat poikkeuksen viitatessaan ohi indeksin. Tällöin try-lause loppuu ja suoritetaan catch-lause. Tässä tapauksessa ei tehdä mitään, vaan palataan takaisin indeksointiin ja yritetään taas suorittaa sijoitusta, joka aiheuttaa jälleen poikkeuksen. Toisessa kokeessa siis alkuosa koodia toimii ilman poikkeuksia, mutta aivan loppupäässä jokainen sijoituslause aiheuttaa poikkeuksen, jonka pitäisi olla suoritusnopeudeltaan hitaampi verrattuna if-lauseeseen.

Toisessa kokeessa tyydyttiin vain viiteen testiajoon, koska tuloksista kävi hyvin selväksi try-catch-lauseen hitaus. Toisesta kokeesta laskettiin vain try-catch-lauseen otoskeskiarvo ja -hajonta, tuloksia ei ollut mielekästä testata enää tilastollisesti, koska testitulokset eroavat toisistaan erittäin selkeästi.

3 Testin tulokset

1. Koe: Poikkeuksia ei aiheuteta, indeksoidaan miljoona lukua 10 000 kertaa. 10 testiajoa.

try-catch-lause: 304 sekuntia	if-lause: 336 sekuntia
try-catch-lause: 293 sekuntia	if-lause: 332 sekuntia
try-catch-lause: 302 sekuntia	if-lause: 337 sekuntia
try-catch-lause: 299 sekuntia	if-lause: 337 sekuntia
try-catch-lause: 313 sekuntia	if-lause: 347 sekuntia
try-catch-lause: 287 sekuntia	if-lause: 321 sekuntia
try-catch-lause: 308 sekuntia	if-lause: 347 sekuntia
try-catch-lause: 299 sekuntia	if-lause: 330 sekuntia
try-catch-lause: 284 sekuntia	if-lause: 315 sekuntia
try-catch-lause: 282 sekuntia	if-lause: 314 sekuntia

Unpaired t test results P value and statistical significance:
try-catch-lauseen otoskeskiarvo: 297,1 if-lauseen otoskeskiarvo: 331,6
try-catch-lauseen otoshajonta: 10,4 if-lauseen otoshajonta: 11,8
The two-tailed P value is less than 0.0001
t = 6.9444
df = 18

2. Koe: Indeksoidaan 100 000 lukua 10 000 kertaa. 1% indeksoinneista aiheuttaa poikkeuksen. Suoritetaan 5 testiajoa.

try-catch-lause: 675 sekuntia	if-lause: 31 sekuntia
try-catch-lause: 667 sekuntia	if-lause: 30 sekuntia
try-catch-lause: 626 sekuntia	if-lause: 30 sekuntia
try-catch-lause: 695 sekuntia	if-lause: 32 sekuntia
try-catch-lause: 631 sekuntia	if-lause: 30 sekuntia

try-catch-lauseen otoskeskiarvo: 658,8
try-catch-lauseen otoshajonta: 29,5

Ensimmäisen kokeen testeistä saadut lukemat viittaavat siihen, että try-catch-lause on if-lausetta nopeampi suorittaa, jos poikkeuksia ei aiheuteta. Itse asiassa yksikään if-lauseen suorituskerta ei ollut tätä nopeampi. Kaksisuuntaisen riippumattoman t-testin antama tulos myös vahvistaa, että tällaista tai vieläkin suurempaa eroa on erittäin epätodennäköistä saada sattumalta (1/10 000). Tulos on siis tilastollisesti erittäin merkittävä. Testin lähtökohdaksi otettiin merkitsevyystaso 0,01, jota pienemmillä p-arvoilla nollahypoteesi uskalletaan hylätä ja vaihtoehtohypoteesi asettaa voimaan. Koska saatu p-arvo on erittäin paljon pienempi kuin 0,01, voidaan H0 hylätä. Try-catch-lauseella ja if-lauseella on suoritusnopeudessa eroa, jos poikkeuksia ei aiheuteta. Testituloksia tarkastelemalla voidaan sanoa, että try-catch-lause on siis hivenen nopeampi.

Toisen kokeen testeistä saadut lukemat taas viittaavat ilman tilastollista testaamistakin erittäin selvästi siihen, että poikkeuksien sattuessa try-catch-lause on merkittävästi if-lausetta hitaampi. Vaikka vain 1% indeksoinneista aiheutti poikkeuksen, suorituskyvyssä tämä vastasi jo 20-kertaista eroa.

4 Yhteenveto ja johtopäätös

Tässä esseessä haluttiin siis tarkastella try-catch-lauseen ja if-lauseen suoritusnopeuden eroja Java-ohjelmointikielellä tehdyllä ohjelmalla. Tarkoituksena oli lähinnä tutkia, kuinka pieniä/suuria erot todellisuudessa olisivat. Tutkimus suoritettiin tietokoneessa suoritettavalla ohjelmakoodilla, joka indeksoi taulukkoa kahdessa eri kokeessa. Ensimmäisessä kokeessa hitaasti suoritettavia poikkeuksia ei tule ja toisessa kokeessa poikkeuksia tulee 1% tapauksista.

Nollahypoteesina oli kummassakin tapauksessa, että try-catch-lauseella ja if-lauseella ei ole suoritusnopeudessa eroa. Oikeasti oletin, että try-catch-lause olisi ensimmäisessä tapauksessa noin puolet hitaampi kuin if-lause ja toisessa tapauksessa alle 10 kertaa hitaampi.

Ensimmäisen kokeen tulokset osoittavat kiistattomasti, että nollahypoteesi on väärä. Try-catch-lauseella ja if-lauseella on suoritusnopeudessa eroa, jos poikkeuksia ei aiheuteta. Tarkemmin sanottuna try-catch-lause on hieman if-lausetta nopeampi. Tämä oli itselleni pienoinen yllätys, sillä odotin if-lauseen olevan nopeampi. Toisessa kokeessa poikkeuksia sattui 1% tapauksista ja if-lause osoittautui ilman tilastollista testaustakin noin 20 kertaa nopeammaksi, kuin saman ohjelmakoodin suorittanut try-catch-lause. Myös tämä tulos yllätti itseni.

Ohjelmoijan kannalta selitys saaduille tuloksille on todennäköisesti se, että ensimmäisessä kokeessa if-lause tarkastaa ehtonsa aina ennen sijoituslauseen suoritusta, kun taas try-catch-lauseessa mitään ei koskaan tarkasteta, vaan tehdään pelkästään sijoituslause. Toisin sanoen if-lause joutuukin suorittamaan yhden rivin koodia enemmän. Toisessa kokeessa if-lause toimii samalla tavalla kuin ensimmäisessä. Kun koodia on 1/10 alkuperäisestä, myös aika on 1/10 ensimmäisestä tapauksesta. Try-catch-lauseessa puolestaan joudutaan nyt sieppaamaan poikkeus catch-osassa, joka jostain kummasta syystä hidastaa suoritusta aivan älyttömästi, vaikka koodia on paljon vähemmän. Johtopäätös on, että try-catch-lausetta kannattaa käyttää, jos poikkeuksia ei todennäköisesti tule yhtään tai vain muutamia. Tavallisiin tarkistuksiin sitä ei kannata missään nimessä käyttää sen hitauden takia.

5 Liite: Käytetty ohjelmakoodi

Testissä käytetty ohjelmakoodi ja kipinä testin tekoon on haettu ja muokattu tähän tarkoitukseen seuraavasta osoitteesta:
http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/5/MittaaPoiAikaa.java

```
////////////////////////////////////////////////////////////////////////
//  Esimerkki Java-kurssilla syksyllä 2005 / Arto Wikla 29.11.2005
//
//  Kokeillaan try-catch- ja if-lauseiden suoritusnopeutta.
//
//  Ohjelman on tarkoitus havainnollistaa sitä, ettei tavallisia
//  tarkistuksia (esim. virheellistä indeksointia) kannata hoitaa
//  poikkeuksia sieppaamalla.
////////////////////////////////////////////////////////////////////////

import java.util.Scanner;
public class MittaaPoiAikaa {

private static Scanner lukija = new Scanner(System.in);
public static void main(String[] args) {

    long tryAlkuAika, tryLoppuAika, tryKulutettuAika, tryAika=0,
    ifAlkuAika, ifLoppuAika, ifKulutettuAika, ifAika=0;
    int kierros=1000;
```

```
    System.out.print("Moneenko kertaan taulukkoa yritetään indeksoida?  ");
    int lkm = lukija.nextInt();

    System.out.print("Moniko indeksoinneista on on kelvollinen?         ");
    int koko = lukija.nextInt();

    int[] t = new int[koko];  // myös 0 on mahdollinen!!

    /* Estetään virheelliset indeksoinnit
     * sieppaamalla poikkeus: (try-catch-lause)
     */

    do {
    tryAlkuAika = System.currentTimeMillis();

    for (int i=0; i<lkm; ++i)
      try {
        t[i] = i;
      }
      catch (ArrayIndexOutOfBoundsException e) {
        // ei tehdä mitään
      }
    tryLoppuAika = System.currentTimeMillis();
    tryKulutettuAika = tryLoppuAika - tryAlkuAika;
    tryAika=tryAika+tryKulutettuAika;


      /* Estetään virheelliset indeksoinnit
       * tarkistamalla indeksi: (if-lause) */

    ifAlkuAika = System.currentTimeMillis();

    for (int i=0; i<lkm; ++i)
        if (i < t.length)
          t[i] = i;
        else ;
         // ei tehdä mitään

    ifLoppuAika = System.currentTimeMillis();
    ifKulutettuAika = ifLoppuAika - ifAlkuAika;
    ifAika=ifAika+ifKulutettuAika;

    kierros--;
    }
    while (kierros>0);

    System.out.print("try-catch-lause: "+tryAika/1000+" sekuntia");
    System.out.println("\t \t if-lause: "+ifAika/1000+" sekuntia \n");
    System.out.println("valmis");
  }
}
```

6 Lähteet

http://fi.wikipedia.org/wiki/Java
http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/5/Poikkeuksista.html
http://www.graphpad.com/articles/interpret/Analyzing_two_groups/unpaired_t.htm
http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/5/MittaaPoiAikaa.java