SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3674

Razvoj okruženja za ispitivanje alata Groningen

Marko Ratkaj

Zagreb, Lipanj 2014

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3674

Razvoj okruženja za ispitivanje alata Groningen

Marko Ratkaj

Zagreb, Lipanj 2014

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 12. ožujka 2014.

ZAVRŠNI ZADATAK br. 3674

Pristupnik: **Marko Ratkaj (0036447289)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Razvoj okruženja za ispitivanje alata Groningen**

Opis zadatka:

Današnji sustavi na kojima se izvršavaju aplikacije vrlo su kompleksni, a prvo što se očekuje od njih je da budu efikasni. U tom smislu potrebno je pokretati aplikacije s odgovarajućom kombinacijom parametara sustava koji će jamčiti dobre performanse tijekom njenog rada. Međutim, određivanje optimalnih parametara vrlo je teško zbog toga što je njihov broj izuzetno velik, raspon vrijednosti je također velik i međudjeluju na vrlo složene načine. Kako bi se olakšalo traženje optimalnih parametara za Java virtualni stroj Google je razvio alat Groningen. U sklopu ovog završnog rada potrebno je proučiti taj alat, razviti okruženje unutar kojega se on može pokretati i ispitivati te potom, koristeći tu okolinu, obaviti ispitivanje nad nekom aplikacijom raspoloživom na Internetu. Radu priložiti izvorni kod razvijenih i korištenih programa. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 13. lipnja 2014.

Mentor:

Doc.dr.sc. Stjepan Groš

Predsjednik odbora za završni rad modula:

Prof.dr.sc. Siniša Srbljić

Djelovođa:

Doc.dr.sc. Tomislav Hrkać

Sadržaj

1. Uvod

Java je objektno orijentirani programski jezik opće namjene. Prvu verziju Jave izdaje Sun Microsystems 1995. godine, a razvijena je s ciljem jednostavnosti, objektne orijentiranosti, robusnosti, sigurnosti, prenosivosti i neovisnosti o arhitekturi računala. Tu prenosivost ostvaruje zahvaljujući konceptu prevođenja izvršnog koda u univerzalni međukod tzv. bytecode koji se izvodi korištenjem virtualnog stroja JVM (engl. *Java Virtual Machine*). Razvoj jezika pratio je i razvoj razvojnog okruženja JDK (engl. *Java Development Kit*) i izvršnog okruženja JRE (engl. *Java Runtime Environment*). Tijekom tog razvoja Java je evoluirala od klijentskog programskog jezika do idealne platforme za razvoj poslužiteljskih aplikacija. Koliko dobro Java virtualni stoj upravlja dretvama, te zauzećem i oslobađanjem memorije postupkom automatskog upravljanja memorijom (engl. *garbage collcetion*), određuje performanse Java aplikacija.

Neke Java aplikacije poslužuju veliki broj klijenata i pritom pristupaju golemim bazama podataka kao što su npr. financijske baze podataka. Cilj je poslužiti te klijente uz najmanje moguće kašnjenje. Termin kašnjenja se odnosi na vrijeme koje prođe od zahtjeva korisnika do trenutka kada on primi odgovor. Optimizacijom Java virtualnog stoja postižu se bolje performanse aplikacija ili Web servisa, a time i manja kašnjenja. Međutim, određivanje optimalnih parametara je teško zbog njihovog velikog broja i raspona vrijednosti.

Postojeća optimizacijska rješenja su usmjerena samo na virtualni stoj ili sakupljač smeća (engl. *Garbage Collector - GC*) i uglavnom su teška za korištenje. Neki primjeri sličnih programa su Groningen[1] i Velocitop Catapult[2]. Groningen je program otvorenog koda razvijan od strane Googlea, dok je Velocitop Catapult zatvorenog koda. Oba programa izvode automatizirane testove i traže optimalne postavke virtualnog stroja. Groningen je još uvijek u razvoju te se njegov izvorni kod može preuzeti s GitHub repozitorija[3]. Razvoj Velocitop Catapulta je prekinut i probnu verziju aplikacije nije moguće skinuti zbog nevažećeg linka[4].

Cilj ovog rada je proučiti alat Groningen, razviti okruženje unutar kojega se on može pokrenuti i ispitivati te potom obaviti ispitivanje nad nekom aplikacijom raspoloživom na Internetu. Daljnji rad će se odnositi i na istraživanje mogućih optimizacija Linux operacijskog sustava u svrhu izvlačenja dodatnih performansi iz Java virtualnog stoja.

S obzirom na činjenicu da postoji mnogo različitih implementacija Java virtualnog stoja, fokus ovog rada će biti na OpenJDK, besplatnoj i otvorenoj implementaciji Java platforme. Verzija Java platforme koja će biti korištena tijekom cijelog ovog rada:

- java version "1.7.0_45"
- OpenJDK Runtime Environment (IcedTea 2.4.3) (build 1.7.0_45-b31)
- OpenJDK 64-Bit Server VM (build 24.45-b08, mixed mode)

Također, svi prikazani postupci instalacija i testiranja provedeni su na Gentoo GNU/Linux operacijskom sustavu verzije kernela 3.12.20-gentoo. Svi postupci su dovoljno generički da bi trebali raditi i na drugim distribucijama.

2. Groningen

Fino podešavanje sakupljača smeća, izazovno je i iziskuje mnogo vremena. Potrebno je mnogo ciklusa stvaranja novih postavki virtualnog stoja, praćenja rada aplikacije pod stvarnim opterećenjem i pregledavanja zapisnika sakupljača smeća dok sustav ne bude radio dovoljno dobro. Najvažniji aspekt podešavanja postavki sakupljača smeća je vrijeme koje aplikacija provodi čekajući. Kada aplikacija čeka ne može izvršavati koristan posao. Iz tog razloga želimo optimizirati rad sakupljača smeća i time smanjiti vrijeme provedeno čekajući, minimizirati kašnjenje i maksimizirati propusnost.

Groningen je aplikacija bazirana na Javi, koja ovaj proces automatizira. Koristi se generacijskim genetskim algoritmom za stvaranje novih skupova modificiranih postavki virtualnog stroja. S tim postavkama, kroz zadano vrijeme, izvodi eksperimentalne zadatke. Najdetaljniji prikaz rada sakupljača smeća dolazi iz njegovih zapisnika. Upravo se oni na kraju svakog eksperimenta sakupljaju i obrađuju na temelju čega se eksperimenti ocjenjuju. Na ovaj način Groningen, na temelju ocjena prethodnih eksperimenata, evoluira nove generacije postavki i eksperimenata pod stvarnim opterećenjem. Nastavlja prolaziti kroz sljedeći skup postavki sve dok ne pronađe skoro optimalne postavke.

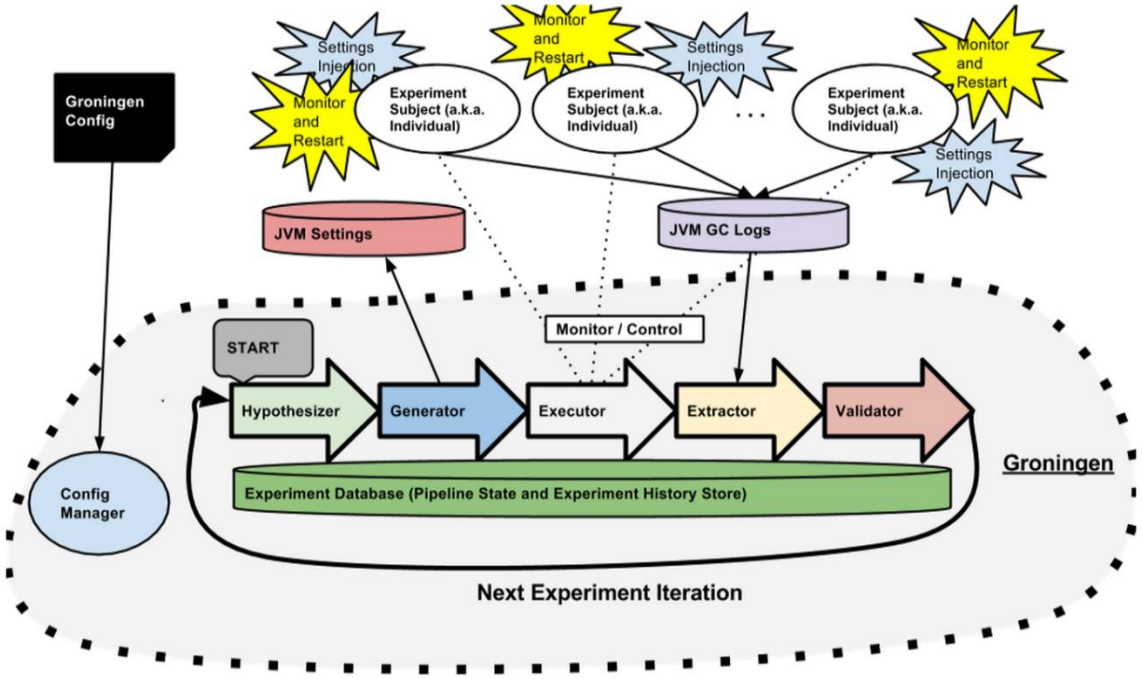
Groningen je osmišljen da podešava aplikaciju na koju je usmjeren na trajnoj osnovi. Korisnik definira područje pretraživanja svih mogućih postavki virtualnog stroja i Groningen će automatski pretraživati kroz to područje dok ne pronađe najbolji skup postavki. Osmišljen je da radi bez ljudske intervencije i da obavijesti korisnika o postavkama koje bi mogle popraviti performanse. Proces je invazivan na način da je instance aplikacije potrebno ponovno pokretati mnogo puta da bi promijenili postavke virtualnog stroja. Iz tog razloga instance koje se testiraju trebaju bi biti podskup stvarne aplikacije ili dio okruženja za provjeru kvalitete.

2.1. Struktura alata Groningen

Groningen se sastoji od cjevovoda procesa koji sadrži šest glavnih komponenti. Te komponente su:

- Hypothesizor,
- Generator,
- Executor,
- Extractor,
- Validator,
- Experiment Database.

Slika 1 prikazuje strukturu Groningena a pojedinačne komponente su opisane u nastavku.



Sl. 1 Struktura Groningena

2.1.1. Hypothesizer

Hypothesizer proces je prvi stadij cjevovoda jednog Groningenovog ciklusa. Ovaj process koristi izlaze *Validatora* i *Extractora* kako bi stvorio novi skup postavki virtualnog stroja. Pri prvom pokretanju *Hypothesizer* stvara slučajan skup postavki virtualnog stroja dok pri svakom idućem koristi generacijski genetski algoritam za stvaranje novih skupova postavki na temelju ocjena prethodnih generacija. Ocjene se generiraju koristeći „fitnes“ funkciju nad podacima u eksperimentalnoj bazi podataka. Fitnes funkciju je moguće podesiti prema aplikaciji nad kojom se koristi. Općenito imamo tri optimizacijska cilja:

- Kašnjenje,
- Propusnost,
- Zauzeće memorije.

Fitnes funkcija je linearna jednadžba Ax+By+Cz gdje su A,B i C konstante definirane od strane korisnika. Za stvaranje generacijskog genetskog algoritma korišteno je *Watchmaker*[5] okruženje. Kromosomi su jednostavno polje cijelih brojeva koje predstavlja postavke virtualnog stroja. Mutiraju se s malom vjerojatnošću kako bi se izbjeglo da program zapne u lokalnom maksimumu. Nove generacije postavki se spremaju u eksperimentalnu bazu podataka.

U tablici 1 dane su sve postavke virtualnog stroja na koje Hypothesizer utječe.

GC serijski način rada	-XX:+UseSerialGC
GC paralelni način rada	-XX:+UseParallelGC -XX:+UseParallelOldGC -XX:ParallelGCThreads=<N>
GC istovremeni način rada	-XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=<N> -XX:+CMSIncrementalMode -XX:+CMSIncrementalPacing -XX:CMSIncrementalDutyCycle=<N> -XX:CMSIncrementalDutyCycleMin=<N> -XX:CMSIncrementalSafetyFactor=<N> -XX:CMSIncrementalOffset=<N> -XX:CMSExpAvgFactor=<N>
Postavke koje utječu na stog	-Xms<min> -Xmx<max> -XX:MinHeapFreeRatio=<minimum> -XX:MaxHeapFreeRatio=<maximum>
Ostale postavke	-XX:MaxGCPauseMillis=<N> -XX:GCTimeRatio=<N> -XX:YoungGenerationSizeIncrement=<Y> -XX:TenuredGenerationSizeIncrement=<T> -XX:AdaptiveSizeDecrementScaleFactor=<D>

Tablica 1 Postavke virtualnog stroja na koje Groningen utječe

2.1.2. Generator

Generator čita postavke virtualnog stroja, koje je stvorio *Hypothesizer* proces, iz eksperimentalne baze podataka i stvara konfiguracijske datoteke potrebne za pokretanje instanci aplikacije s novim postavkama.

2.1.3. Executor

Executor je zadužen za ponovno pokretanje eksperimenata, tj. instanci aplikacija te ih tijekom cijelog eksperimenta nadzire. Ako se eksperimentalni zadatak prečesto ponovno pokreće tada se on označava kako bi ga kasnije *Validator* proglasio nevažećim. *Executor* koristi *Experiment* proces da bi dobio listu procesa koje je potrebno ponovno pokrenuti. To je potrebno da bi bilo moguće promijeniti postavke virtualnog stroja. Podaci o ponovnim pokretanjima zapisuju se u eksperimentalnu bazu za kasniju provjeru valjanosti od strane *Validator*a.

2.1.4. Extractor

Extractor pokreće *Executor* proces na kraju svakog eksperimenta te on izvlači podatke o sakupljaču smeća iz njegovih zapisnika. Te datoteke se otvaraju, podaci se raščlanjuju i

razvrstavaju i zatim pohranjuju u eksperimentalnu bazu podataka. Podaci se bilježe u što je moguće više detalja kako bi Groningen mogao koristeći fitnes funkciju odrediti koliko su uspješne bile određene mutacije u danoj generaciji.

2.1.5. Validator

Validator je posljednja faza cjevovoda i provodi provjeru uspješnosti eksperimenata. Njegova zadaća je spriječiti da abnormalne generacije utječu na stvaranje novih generacija postavki. *Validator* postavlja prag valjanosti kao pozitivni cijeli broj. Ako eksperiment nije valjan *Validator* postavlja zastavicu ne valjanosti u eksperimentalnoj bazi podataka kako bi *Hypothesizer* znao da ga ne smije koristiti u stvaranju nove generacije postavki.

2.1.6. Experiment Database

Eksperimentalna baza podataka sadrži sve podatke koje stvaraju i koriste sve glavne komponente Groningena. Također, omogućuje pohranjivanje dijela podataka u radnu memoriju radi brzog čitanja i pisanja u raznim stadijima cjevovoda. Članovi cjevovoda ne komuniciraju međusobno već isključivo preko eksperimentalne baze podataka.

2.2. Instalacija alata Groningen

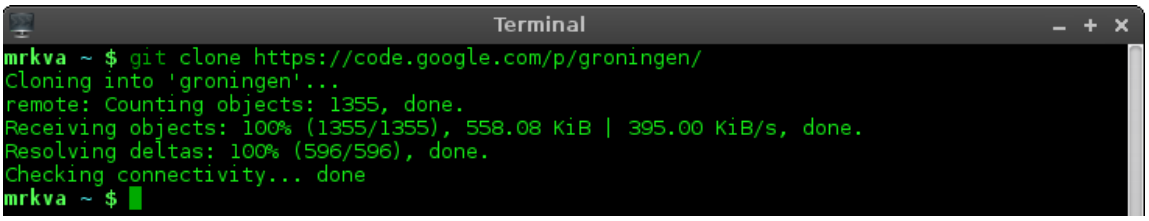
Za ispravan rad Groningena potreban je *protobuf*[6] verzije 2.4.1 ili noviji. *Protobuf* omogućuje kodiranje strukturiranih podataka u format koji je učinkovit a ujedno i proširiv. U slučaju lokalno instaliranog i podešenog *protobufa*, koji nije instaliran na neku sistemsku lokaciju (kao npr. */usr/bin*), potrebno je postaviti sistemsku varijablu *PROTOC EXECUTABLE* tako da pokazuje pravilnu putanju. To je moguće učiniti slijedećom naredbom, pritom je potrebno zamijeniti „negdje“ sa pravilnom putanjom.

```
$ export PROTOC_EXECUTABLE=/negdje/protobuf-2.4.1/build/bin/protoc
```

Ako je protobuf nasnimljen na sistemsku lokaciju (najvjerojatnije */usr/bin/protoc* ili */usr/local/bin/protoc*) nije potrebno izvršavati prethodnu naredbu.

Lokalnu kopiju Groningen repozitorija moguće je dohvatiti slijedećom naredbom, te se njen izlaz može vidjeti na slici 2.

```
$ git clone https://code.google.com/p/groningen/
```



Sl. 2 Dohvaćanje Groningen repozitorija

Potrebno je postaviti se u groningen direktorij:

```
$ cd groningen
```

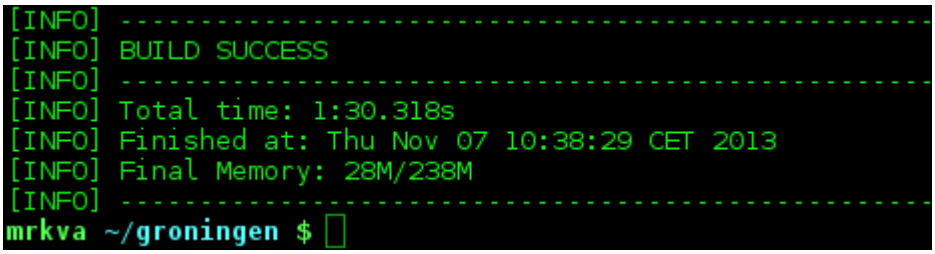
Da bi izgradili Groningen potrebno je imati instaliran Apache Maven[7], alat za automatizaciju procesa izgradnje aplikacije. Groningen možemo izgraditi sljedećim naredbama:

```
$ mvn package
$ mvn assembly:assembly
```

Također je moguće generirati definicije za razvojno okruženje Eclipse.

```
$ mvn eclipse:eclipse
```

Sve tri *mvn* naredbe bi trebale završiti izvođenje porukom „*build success*“.

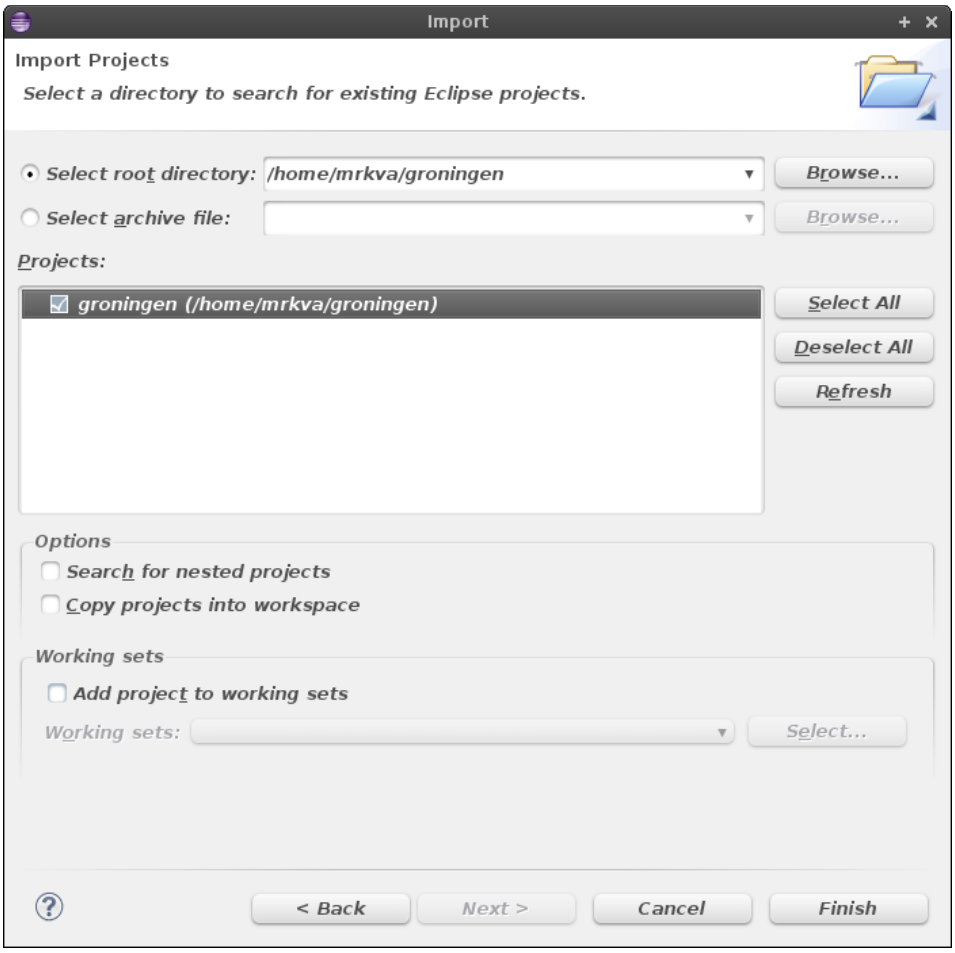


Sl. 3 Uspješno izvođenje *mvn* naredbi

Zbog načina na koji Maven prevodi projektni model objekta na Eclipseov izvorni format, potrebno je pokrenuti Eclipse sa sljedećom naredbom:

```
$ M2_REPO="${HOME}/.m2/repository" eclipse
```

U Eclipse se može unijeti Groningen odabirom opcije „*File*“, pa „*Import...*“ i naznačite „*Existing Project into Workspace*“. Kako se može vidjeti na slici 4, u polje „*Select root directory*“, potrebno je unijeti lokaciju Groningen direktorija i kliknuti „*Finish*“.



Sl. 4 Uvođenje Groningena u Eclipse

Pokazalo se da u ovoj fazi nedostaje nekoliko datoteka i direktorija. To uzrokuje mnoga upozorenja i pogreške u izvođenju programa, pa čak i njegov prestanak rada. Primjerice, pokretanjem programa primijećeno je da nedostaje zapisnik „*tmp-groningen events*“, te također direktorij „*alloc/logs*“ unutar kojega bi se trebao nalaziti, što uzrokuje prestanak rada aplikacije uz poruku o pogrešci. Direktorije je moguće stvoriti slijedećom naredbom:

```
$ mkdir -p alloc/logs
```

Dok se zapisnik stvara koristeći naredbu *touch*:

```
$ touch alloc/logs/tmp-groningen_events
```

Također, potrebno je stvoriti direktorij „*my exp settings*“ te u njemu datoteke 0 i 1, bez kojih će program raditi neispravno i ispisivati poruke upozorenja. Te datoteke će služiti za privremenu pohranu generiranih postavki. Direktorij stvorite *mkdir* naredbom:

```
$ mkdir my_exp_settings
```

Touch naredbom stvorite potrebne datoteke.

```
$  touch my_exp_settings/0
$  touch my_exp_settings/1
```

Za izvođenje programa potrebno je stvoriti i zapisnik „*groningen.log*“ kako je učinjeno u nastavku:

```
$  touch groningen.log
```

U ovom fazi Groningen je potpuno nasnimljen i spreman za uporabu.

2.3. Friesian aplikacija

Friesian je jednostavna aplikacija, osmišljena da simulira memorijsko ponašanje serverskog Java virtualnog stroja. To je višedretvena, 100% Java aplikacija koja se može koristiti za provjeru učinkovitosti optimizacija Java virtualnog stroja. Brzo se pokreće i simulira dnevno opterećenje koje korisnici stvaraju na stvarnim Java Web aplikacijama.

Friesian je stvoren kao pomoćni alat pri razvoju Groningena. Zbog svoje svrhe i jednostavnosti Friesian će biti korišten za testiranje Groningena tijekom ovog rada.

2.3.1. Instalacija

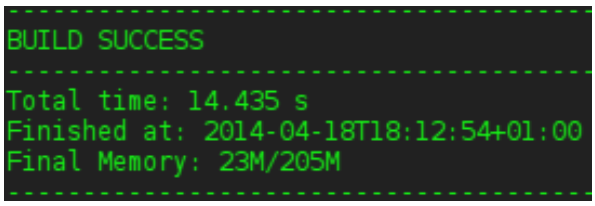
Potrebno je klonirati git repozitorij koristeći sljedeću naredbu:

```
$  git clone https://code.google.com/p/friesian/
```

Ponovno koristimo Apache Maven da bi izgradili Friesian aplikaciju:

```
$  mvn package
$  mvn assembly:assembly
```

Obje naredbe trebaju završiti porukom o uspjehu kao na slici 5.



Sl. 5 Uspješno stvaranje Friesian aplikacije

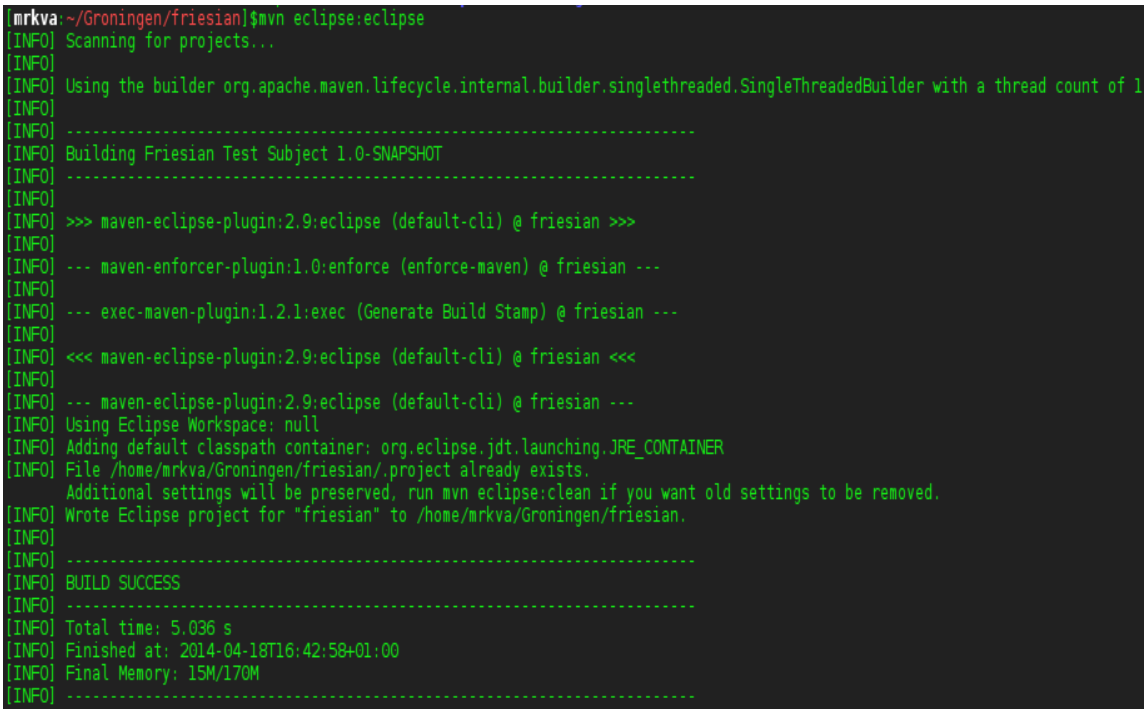
Najjednostavniji način pokretanja Friesian aplikacije je kroz komandnu liniju koristeći naredbu:

```
$  java -jar target/friesian-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Alternativni način pokretanja je uvesti Friesian u Eclipse. Prvo je potrebno stvoriti potrebne Eclipse definicije.

```
$  mvn eclipse:eclipse
```

Naredba će generirati ispis kao na slici 6.



Sl. 6 Stvaranje Eclipse definicija

Kao i kod uvođenja Groningena, potrebno je pokrenuti Eclipse sljedećom naredbom:

```
$  M2_REPO="${HOME}/.m2/repository" eclipse
```

Unesite Friesian u Eclipse odabirom opcije „*File*“,pa „*Import...*“ i naznačite „*Existing Project into Workspace*“. U polje „*Select root directory*“ unesite lokaciju Friesian direktorija i kliknite „*Finish*“.

Da bi pokrenuli Friesian, kliknite na ikonu „*Run*“ ili pritisnite „*Ctrl+F11*“ i kliknite na „*Run as a Java Application*“. Friesian proces će se sada izvoditi u pozadini.

3. Podešavanje alata Groningen

Podešavanje se vrši pomoću jedne konfiguracijske datoteke na koju se Groningen usmjerava preko parametara komandne linije. Primjer konfiguracijske datoteke sa Groningen wiki[8] dan je u nastavku.

```
// Sample user groningen config
user: "john"
cluster {
  cluster: "localhost"
  subject_group {
    subject_group_name: "1234" // process group ID (PGID)
    exp_settings_files_dir: "my_exp_settings" // experiment settings dir
  }
}
param_block {
  input_log_name: "STDOUT" // parse as input
  number_of_executor_threads: 3 // number of threads to monitor subjects
  duration: 30 // time in minutes that experiment is run in prod
  restart: 4 // restart threshold
  latency_weight: 100
  throughput_weight: 200
  memory_weight: 50000
  num_crossovers: 1
  mutation_prob: 0.2
  stagnant_gens: 24
  elite_count: 2
}
jvm_search_restriction: {
  gc_mode: USE_CONC_MARK_SWEEP
}
```

Važno je pritom obrisati komentare iz konfiguracijske datoteke inače će program javljati grešku kod čitanja konfiguracije. Također je potrebno uvesti izmjene kako je objašnjeno u nastavku. Da bi usmjerili Groningen na Friesian aplikaciju, potreban nam je PGID (Process Group ID) pokrenute Friesian aplikacije. Prvo je potrebno pokrenuti Friesian. To je najlakše koristeći sljedeću naredbu ili koristeći Eclipse kako je ranije navedeno:

```
$ java -jar target/friesian-1.0-SNAPSHOT-jar-with-dependencies.jar
```

PGID se može saznati korištenjem *ps*[9] naredbe. Naredbu je potrebno koristiti sa opcijama „axjf“ te je nužno pronaći liniju friesian procesa kao na slici 7. Traženi PGID broj se nalazi u trećem stupcu i u ovom slučaju iznosi 9758.

```
$ ps axjf
```

Sl. 7 Ispis *ps* naredbe sa naznačenim friesian procesom i PGID brojem

Ovaj broj je potrebno dodati u polje *subject group name* u konfiguracijskoj datoteci. Pritom polje *user* kao i ime konfiguracijske datoteke mogu biti proizvoljni. Polje *exp settings files dir* treba sadržavati punu putanju do direktorija *my exp settings* stvorenog tokom nasnimavanja Groningen alata. Taj će se direktorij koristiti za pohranjivanje postavki virtualnog stroja po završetku *Generator* procesa. Također, da bi se program uspješno izvodio

nad Friesian aplikacijom, polje *elite count* treba imati vrijednost jedan, umjesto dva kako je navedeno u primjeru konfiguracijske datoteke na Groningen wiki.

Nakon tih izmjena konfiguracijska datoteka bi trebala izgledati kao u nastavku:

```
user: "user1"
cluster {
   cluster: "localhost"
   subject_group {
      subject_group_name: "9758"
      exp_settings_files_dir: "/home/gentoo/groningen/my_exp_settings"
   }
}
param_block {
   input_log_name: "STDOUT"
   number_of_executor_threads: 3
   duration: 30
   restart: 4
   latency_weight: 100
   throughput_weight: 200
   memory_weight: 50000
   num_crossovers: 1
   mutation_prob: 0.2
   stagnant_gens: 24
   elite_count: 1
}
jvm_search_restriction: {
   gc_mode: USE_CONC_MARK_SWEEP
}
```

Objašnjenja ovih konfiguracijskih postavki:

- ***input_log_name*** – Datoteka u koju će Groningen ispisivati trenutni status;
- ***number_of_executor_threads*** – Broj dretvi koje ce *Executor* koristiti za praćenje stanja eksperimenta;
- ***duration*** – Vrijeme u minutama koliko će se eksperiment izvoditi;
- ***restart*** – Maksimalan dozvoljeni broj ponovnog pokretanja eksperimenata koji koriste *Executor* i *Validator* pri ocjenjivanju valjanosti;
- ***latency_weight*** – Važnost kašnjenja pri stvaranju novih generacija u *Hypothesizer* procesu;
- ***throughput_weight*** - Važnost propusnosti pri stvaranju novih generacija;
- ***memory_weight*** - Važnost memorijskog otiska aplikacije pri stvaranju novih generacija postavki;
- ***num_crossovers*** – Broj prijelaznih poena za stvaranje novih generacija
- ***mutation_prob*** – Vjerojatnost mutacija;
- ***stagnant_gens*** – Broj generacija koje stagniraju prije nego što se generacijski genetski algoritam zaustavi;
- ***elite_count*** – Broj elitnih populacija;
- ***gc_mode*** – Način rada sakupljača smeća.

Groningen se pokreće koristeći sljedeću naredbu:

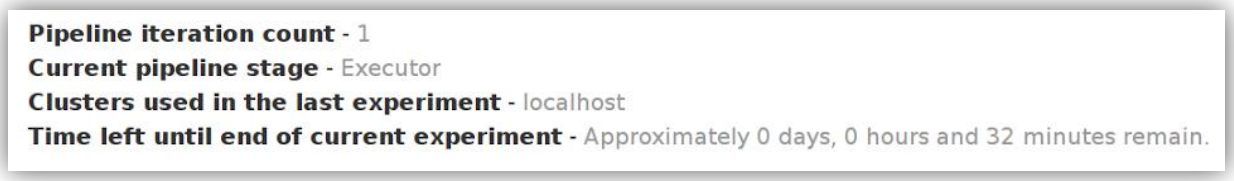
```
$  java -jar target/groningen-1.0-SNAPSHOT-jar-with-dependencies.jar
   --eventLogPrefix=groningen.log
   --configFileNames=proto:txtfile:my_config.txt
```

Ovime se pokreće Groningen Web sučelje na pristupnim vratima 8080. Da bi pristupili tom sučelju potrebno je pokrenuti Web pregledanik i otvoriti adresu *http://localhost:8080/* . Pojaviti će se sučelje prikazano na slici 8.

Sl. 8 Groningen Web sučelje

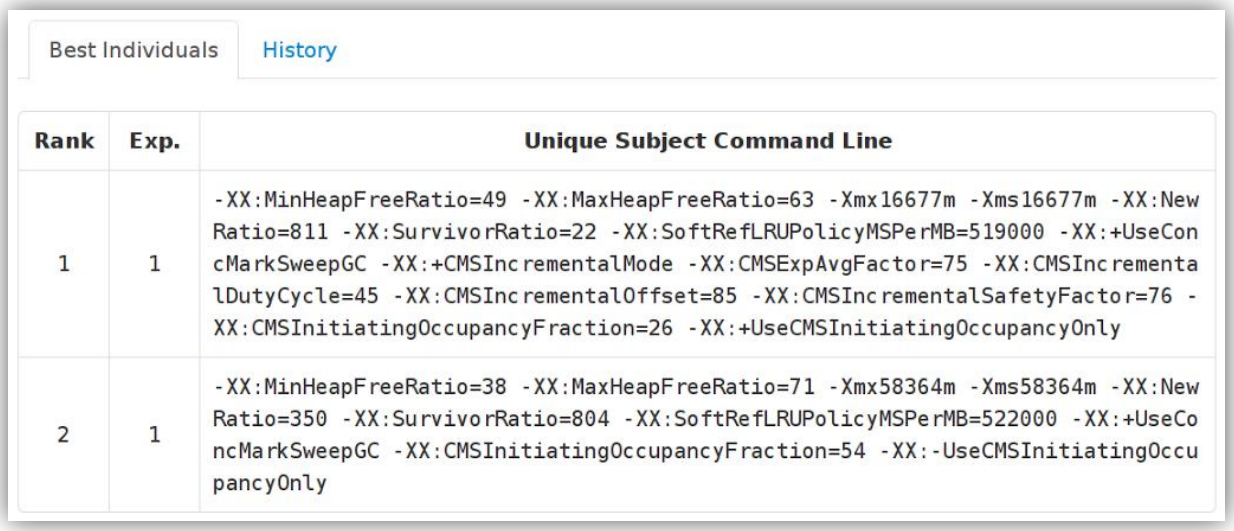
4. Groningen web sučelje

Primarni način interakcije alata Groningen je preko Web sučelja. To je HTTP *serverlet* koji sluša na pristupnim vratima 8080. Vrata na kojima se prikazuje Web sučelje se može proizvoljno mijenjati koristeći argumente komandne linije. U razvoju je i RPC (engl. Remote Procedure Call) sučelje koje će omogućiti upravljanje Groningenom iz neke druge aplikacije sa ciljem ugradnje Groningena u oblak. Web sučelje je vrlo jednostavno. Pritiskom na korisničko ime otvara se prozor koji se sastoji od dva dijela. Prvi dio prikazan je na slici 9 i prikazuje broj iteracija kroz cjevovod u polju „*Pipeline iteration count - 1*“. Trenutni stadij cjevovoda moguće je vidjeti u polju „*Current pipeline stage - Executor*“, dok polje „*Time left until end of current experiment*“ prikazuje prestalo vrijeme do kraja trenutnog eksperimeta, u ovom slučaju to su 32 minute.



Sl. 9 Web sučelje, trenutni status Groningena

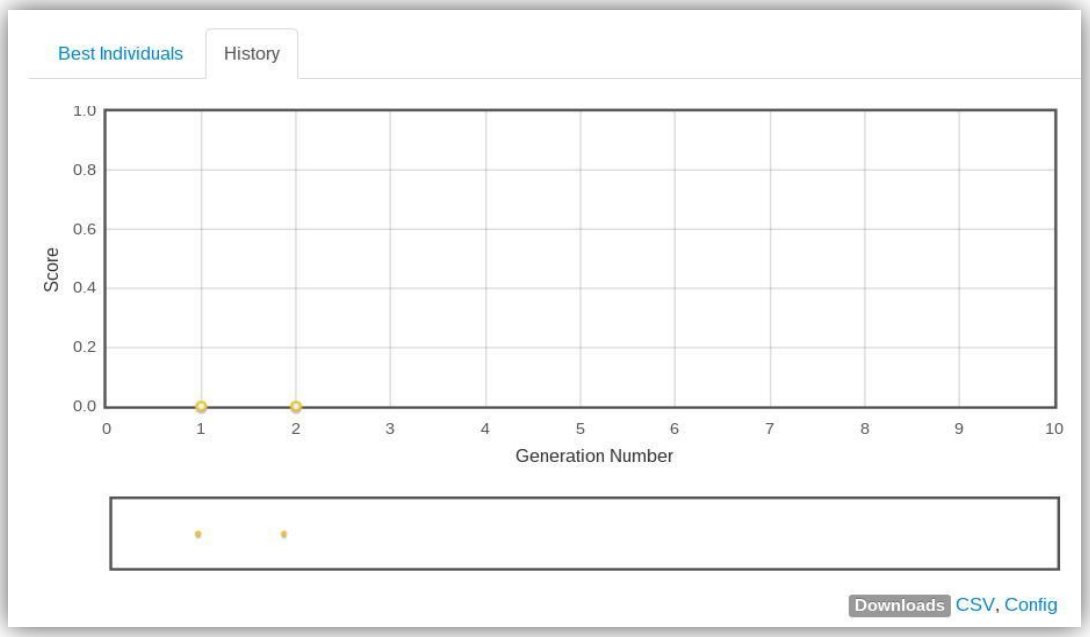
Drugi dio Web sučelja prikazan je na slici 10 i sadrži tablicu sa rangom i brojem eksperimenta te također postavke virtualnog stroja koristene tokom tih eksperimenata.



Sl. 10 Web sučelje, rezultati eksperimenata

Važno je napomenuti da se u trenutnom stadiju razvoja Groningena prikaz broja iteracija ne mijenja te da je preostalo vrijeme do kraja eksperimenta vrlo neprecizno.

Pritiskom na polje „*History*“ moguće je vidjeti graf provedenih eksperimenata i njihove ocjene.



Sl. 11 Povijest eksperimenata

U doljnjem desnom kutu pritiskom na „*Config*“ moguće je preuzeti konfiguracijsku datoteku sa kojom je Groningen pokrenut. Pritiskom na CSV preuzima se csv (engl. Comma Separated Values) datoteka koja sadrži vrijednosti iz tablice na slici 9. Iz nepoznatih razloga ocjene eksperimenata na grafu su uvijek nula. Također, u određenim stadijima cjevovoda može doći do privremenog prestanka rada Web sučelja te je u tom slučaju potrebno kratko sačekati i ponovno učitati stranicu.

5. Zaključak

Groningen obećava riješiti vrlo složen problem automatizacije procesa optimizacije Java virtualnog stroja. Međutim, Groningen je trenutno daleko od ostvarenja tog cilja. Tokom proučavanja ovog alata javljali su se mnogi problemi od kojih su neki:

- Nedostatak određenih direktorija i datoteka nakon nasnimavanja;
- Problem traženja odgovarajućih postavki da bi program radio;
- Razni problemi sa Web sučeljem navedeni ranije;
- Manjkave i netočne informacije na Groningen wiki stranicama.

Također, u izvornim kodovima je primijećeno puno zakomentiranih blokova koda i puno „*TODO*“ komentara koji opisuju što je još potrebno napraviti. Primjerice iduća naredba, unutar Groningen direktorija, otkriva gotovo stotinu takvih komentara, što je moguće vidjeti na slici 12.

```
$  grep -iR todo | grep .java | wc -l
```

```
[gentoo:~/FER/zavrsni_rad/groningen]$ grep -iR todo | grep .java | wc -l
96
```

Sl. 12 Veliki broj TODO komentara unutar izvornog koda

To upućuje na činjenicu da je program daleko od produkcije. Prema GitHub statistici, razvoj teče vrlo sporo sa periodima i od nekoliko mjeseci bez ikakvih promjena što dovodi do određenog skepticizma glede dovršetka Groningen projekta. Međutim, čak i da projekt ne bude dovršen, Groningen daje vrlo zanimljiv pristup problemu optimizacije. Njegov dizajn i ideja se mogu iskoristiti i proširiti u budućim projektima.

6. Literatura

[1] Groningen, https://github.com/sladeware/groningen/wiki/,
[2] Java perfomance tunning, Velocitop Catapult, http://www.javaperformancetuning.com/tools/catapult/,
[3] Groningen GitHub repozitorij, https://github.com/sladeware/groningen
[4] Velocitop Catapult, probna verzija, http://www.velocitop.com/try.html
[5] Watchmaker okruženje, http://watchmaker.uncommons.org/
[6] Protobuf, https://code.google.com/p/protobuf/
[7] Apache Maven, http://maven.apache.org/
[8] Primjer konfiguracijske datoteke, https://github.com/sladeware/groningen/wiki/Deployment#config
[9] Priručne stranice ps naredbe, http://unixhelp.ed.ac.uk/CGI/man-cgi?ps

Sažetak

Optimizacijom Java virtulnog stroja postižu se bolje performanse Java aplikacija ili Web servisa, ali je traženje optimalnih postavki izazovan i dugotrajan posao. Groningen je alat, baziran na Javi, koji automatizira taj proces. Koristi se generacijskim genetskim algoritmom za stvaranje novih skupova postavki. Groningen je još u ranom razvoju, ali sadržava zanimljive ideje i zanimljiv pristup problemu optimizacije Java virtualnog stroja. U radu je opisan postupak instalacije i podešavanja Groningena za testiranje.

By optimizing Java virtual machine it is possible to achive better perfomance of Java applications and Web services, but searching for the right JVM settings is a challanging job. Groningen is a tool writen in Java, wich promises to automate that process. It uses generational genetic algorithm to generate new sets ov JVM settings. Groningen is still in early development but it has an interesting approach to the problem of optimization. In this thesis, it's shown how to install and set Groningen for testing.