SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Anton Grbin

v8boinc - Razvoj raspodijeljenog superračunala

Zagreb, travanj 2014.

Ovaj rad izrađen je na Sveučilištu u Zagrebu, Fakultetu Elektrotehnike i Računarstva, na Zavodu za elektroničke sustave i obradbu informacija pod vodstvom doc. dr. sc. Ante Đereka i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2013./2014.

POPIS SLIKA

SADRŽAJ

1. Uvod

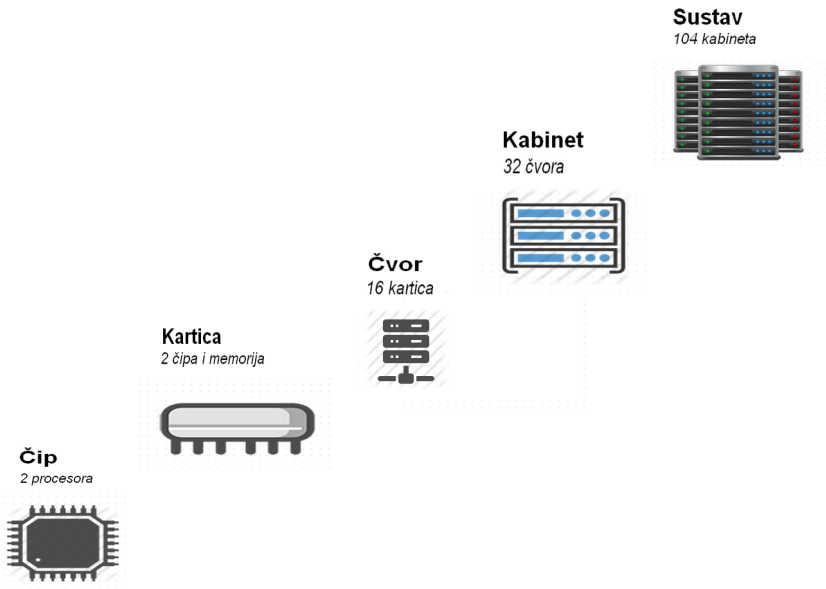
Kako bi postigli daljnji napredak u brzini izvršavanja računalnih procesa, stručnjaci danas sve više i više pribjegavaju paralelnom izvršavanju, odnosno istodobnom izvršavanju različitih dijelovi zadataka na više procesorskih jedinica. U laboratorijskim okruženjima tako susrećemo superračunala i grozdove računala koji znanstvenici koriste u razne svrhe, od modeliranja vremenskih prognoza do ispitivanja djelovanja novih lijekova. Ovaj rad opisuje razvoj financijski isplativog raspodijeljenog sustava za paralelno računanje s visokim performancama koji će se moći koristiti na našem sveučilištu, a temelji se na volonterskom računarstvu.

U uvodnom poglavlju navodimo oblike raspodijeljenih sustava za paralelno računanje i njihove značajnije predstavnike. Na kraju poglavlja ukratko opisujemo model koji smo implementirati tijekom akademske godine 2013./14. koristeći postojeću računalnu infrastrukturu i pokazujemo kako razvijena podrška može biti od koristi sveučilištu uz ostale doprinose našeg rada.

1.1. Oblici raspodijeljenog računanja

Superračunalo je usko povezani skup višeprocesorskih računala s brzom mrežom, zajedničkim spremničkim prostorom i sustavima za hlađenje[4]. U trenutku pisanja ovog rada najbrže superračunalo na svijetu je Tianhe-2, razvijeno na Kineskom nacionalnom fakultetu obrambenih tehnologija[18] sastavljeno od 3 000 000 procesorskih jezgri s ukupnom procesorskom moći od 33 petaFLOPS ili $3.3 \cdot 10^{16}$ decimalnih operacija po sekundi (engl. *FLOPS - floating point operations per second*).

Iako nije lokacijski raspodijeljen, superračunalo izvodi masivno paralelne zadatke raspoređene na tisuće procesora što je prikazano na slici 1.1. Ovakva postrojenja koriste se u znanstvene svrhe kao što su modeliranje vremenskih prognoza, provjera sigurnosti kripto-sustava te razne dinamičke simulacije. Spomenimo kako je simulacija fizičkog razmještaja proteina jedan od zahtjevnijih računalnih problema današnjice, a izrazito je bitan za razvoj lijekova i bolje shvaćanje nekih bolesti[23].



Slika 1.1: IBM BlueGene/L, diagram razmještaja procesorskih jedinica
Ovo superračunalo ima ukupno $2 \cdot 2 \cdot 16 \cdot 32 \cdot 104 = 212992$ procesorskih jedinica[4].

Budući da je izgradnja postrojenja kao što je superračunalo izrazito financijski zahtjevan projekt, stručnjaci su razmotrili i ostale izvedbe masivno paralelnih rješenja. Jedno takvo rješenje je računarenje u grozdu (engl. *grid computing*) u kojem su računala opće namjene iste arhitekture i platforme spojena brzom mrežom u funkcionalnu računalnu cjelinu. Programska podrška namijenjena ovom modelu paralelnog računarstva omogućava da se cijeli skup računala može koristiti kao jedna homogena cjelina. Ako se za takvu programsku podršku koristi neko od postojećih rješenja otvorenog koda, cijena ovakvih sustava drastično je smanjena u odnosu na izgradnju superračunala.

Računarenje u grozdu podržava i Sveučilišni računalni centar - SRCE pomoću računalnog grozda nazvanog Isabella[2]. Grozd u trenutku pisanja ovog rada broji 64 procesora s ukupno 480 procesorskih jezgri. Desetak domaćih znanstvenih projekata ga koriste kao izvor procesorske moći. Kada bi se njegova procesorska moć okvirno pretvorila u standardnu mjernu jedinicu FLOPS, dobili bi vrijednost 4.8 teraFLOPS. Primjer problema koji zahtjeva računalnu snagu koju grozd Isabella može obraditi u jednoj sekundi je rješavanje sustava 10 000 jednadžbi s 10 000 nepoznanica koristeći savršeno paralelni algoritam. Ovakav problem mogao bi se susresti u projektiranju kompleksnog elektroničkog sklopa ili kod optimizacije rezultata u Internet tražilicama[22].

1.2. Volontersko računarstvo

Osim superračunala i grozdova računala postoji i financijski još efikasnije rješenje za masivno paralelno računarstvo - volontersko računarstvo. Radi se o oportunističkom raspodijeljenom modelu u kojem pri proračunu sudjeluju računala koja se u danom trenutku ne koriste za originalnu namjenu. To uključuje računala u laboratorijima, osobna računala, prenosive uređaje koji po noći imaju ciklus punjenja baterije, pa čak i mrežnu opremu kao što su usmjerivači. U literaturi se za ovaj model susrećemo s nazivima volontersko računarstvo (engl. *volunteer computing*), računarenje na javnim resursima (engl. *public resource computing*), mreže radnih stanica (engl. *network of workstations*)[7, 29]. Glavna motivacija za volontersko računarstvo je činjenica da svjetska računalna moć danas nije koncentrirana u ustanovama sa superračunalima ili grozdovima već je dominantan značajan broj računala i uređaja koji se nalaze u vlasništvu pojedinaca[7].

Volontersko računarstvo uzima zamah tijekom zadnjeg desetljeća prošlog stoljeća, kada se općenito Internet ubrzano razvija. Prvi projekt volonterskog računarstva pojavljuje se 1996. godine s projektom GIMPS[1] (engl. *Great Internet Mersenne Prime Search*) kojemu je cilj pronaći velike proste brojeve. Ovaj projekt u trenutku pisanja ovog rada ima više od 100 000 aktivnih korisnika volontera koji doniraju preko 400 teraFLOPS računalne snage. Deset trenutno najvećih prostih brojeva pronađeno je zahvaljujući upravo ovom projektu.

Vjerojatno najpoznatiji projekt volonterskog računarstva je SETI@home[8], lansiran 1999. godine sa svrhom obrađivanja signala prikupljenih radio-teleskopima u potrazi za znakovima vanzemaljske inteligencije. Iako projekt do trenutka pisanja nije uspio pronaći tražene znakove, infrastruktura na kojoj je pokrenut otvorenog je koda i služi kao podloga za mnoge druge projekte volonterskog računarstva. Nastala na Berkeley sveučilištu BOINC (engl. *Berkeley Open Infrastructure for Network Computing*) danas poslužuje više desetaka projekata koji imaju prepoznatljiv nastavak u imenu "@home". U trenutku pisanja cjelokupna BOINC mreža sastoji se od skoro tri milijuna korisnika volontera koji zajedno u prosjeku doniraju 7.9 petaFLOPS procesorske snage.

Izgradnja raspodijeljenog superračunala koristeći BOINC infrastrukturu gotovo da nema troškova osim ljudskih resursa potrebnih za instalaciju programske podrške i održavanje središnjeg poslužitelja. Neka od svjetskih sveučilišta koja također koriste raspodijeljeno superračunalo temeljeno na BOINC infrastrukturi su *University of California* u Berkeleyu, *University of Westminster*, *Stanford University*, *Hungarian*

Academy of Sciences i *Michigan State University*[5, 9].

Ovaj rad opisuje postavljanje BOINC sustava na našem sveučilištu, izazove koje smo putem susreli i rješenja koja smo osmislili kako bi se sustav iskoristio u najvećoj mjeri.

1.3. Izazovi volonterskog računarstva

BOINC se sastoji od središnjeg poslužitelja i raspodijeljenih klijenata koji pokreću BOINC klijentski program. Klijentski program povlači s poslužitelja zadatke sastavljene od izvršne datoteke i ulaznih podataka, pokreće ih, i po završetku obrade izvještava poslužitelja o rezultatima. U ovakvom modelu zadatak se isplati poslati na izvršavanje udaljenom klijentu samo ako će vrijeme potrebno da se preuzmu ulazni podaci biti manje od vremena potrebnog da se izvrši obrada podataka i ako između radilica ne mora postojati komunikacijski kanal. Problemi koji zahtijevaju ovakvu obradu podataka su trivijalno paralelni problemi koji se mogu rastaviti na nezavisne potprobleme [6, odjeljak 5].

Budući da su klijenti raznovrsni po pitanju arhitekture i platformi, izvršna datoteka mora biti pažljivo konstruirana kako bi se mogla pokrenuti na ciljnom računalu. Dodatno, kako se izvršna datoteka pokreće s ostalim aplikacijama koje klijent u tom trenutku koristi u istom operacijskom sustavu, mora postojati mehanizam koji osigurava korisnika da sudjelovanjem u proračunu neće naštetiti svom računalu. Iako klijent možda ima potpuno povjerenje u voditelje projekta u kojem sudjeluje, mrežna komunikacija između njega i poslužitelja ne mora biti sigurna što otvara vrata napadima čovjeka u sredini (engl. *man-in-the-middle attack*). U klasičnom BOINC modelu, izvršne datoteke digitalno su potpisane kako ih se ne bi moglo izmijeniti, dok se za ulazne podatke pretpostavlja da njihovom izmjenom nije moguće naštetiti klijentu. Zbog toga načini korištenja infrastrukture koji prenose aplikaciju kao dio ulaznih podataka, kako je i mi koristimo, mogu biti osjetljivi na ovakav napad.

Prenosivost izvršnih datoteka znači da znanstvenik koji razvija svoju aplikaciju mora uložiti dodatno vrijeme kako bi bio siguran da njegova aplikacija radi na svakoj platformi i arhitekturi. Ovisno o složenosti znanstvene aplikacije prilagođavanje za sve platforme može biti **vremenski vrlo zahtjevno** ili čak neizvedivo. Uključivo sa sigurnosnim pitanjima, heterogenost platformi predstavlja najveće izazove u volonterskom računarstvu.

Kako bi apstrahirali razinu sklopovlja i platforme, mnogi autori predlažu neki vid virtualizacije kao međusloj između klijentskog računala i znanstvene aplikacije[12,

15, 17, 24, 25]. U takvim rješenjima prema klijentu se kao ulazni podatak šalje slika cjelokupnog virtualnog računala (engl. *virtual image*) koje sadrži znanstvenu aplikaciju, a pretpostavlja se da klijent ima ugrađenu neku od virtualizacijskih tehnologija koja bi je mogla pokrenuti. Glavni nedostatci korištenja virtualizacije u ovom načinu su prevelika količina podataka koja se mora prenijeti do klijenta i pretpostavka da će klijent imati željenu programsku podršku što znatno smanjuje obujam prigodnih volontera.

Naš pristup rješavanja problema heterogenih platformi također koristi virtualizacijsku tehnologiju, ali u drugačijem i inovativnom izdanju. Naime, kao međusloj između znanstvene aplikacije i klijentskog operacijskog sustava koristimo JavaScript izvršnu okolinu V8 preuzetu iz Internet preglednika Google Chrome. Budući da je ova programska podrška otvorenog koda i potpuno prenosiva, uspjeli smo je prevesti na strojni jezik mnogih arhitektura. Oslanjajući se na sigurnosni model za izvršavanje JavaScript programa preuzetog iz preglednika Chrome riješili smo problem sigurnog izvođenja znanstvene aplikacije čak i u slučaju izmjene njenog sadržaja.

Kako zahtijevamo da je jezik izgradnje znanstvene aplikacije JavaScript, jednostavno smo proširili rješenje na obradu podataka u Internet preglednicima. To znači da osim što je moguće sudjelovati u zajedničkom proračunu koristeći klijentsku BOINC aplikaciju na stolnim i prijenosnim računalima, svi uređaji koji imaju Internet preglednik također mogu donirati svoju procesorsku snagu jednostavnim posjećivanjem Internet stranice. Time je broj prikladnih platformi dodatno povećan što ujedno povećava i broj volontera koji se mogu priključiti proračunu.

Iako je iskorištavanje Internet preglednika za raspodijeljene proračune u znanstvene svrhe već viđeno u nekoliko instanci[3, 27], korištenje same okoline za izvršavanje JavaScript programa van preglednika u raspodijeljenom sustavu kao virtualizacijski sloj nije spominjano.

Rezultat je mogućnost da znanstveni program napisan u jeziku JavaScript možemo pokrenuti u raspodijeljenom sustavu na uređajima poput stolnih i prijenosnih računala s operacijskim sustavima Windows, OS X i Linux, na mrežnim usmjeriteljima, i svim uređajima koji imaju preglednik što uključuje pametne telefone, igrače konzole i novije televizore.

Činjenica da znanstvenik ne mora prilagođavati izvorni kôd da bi njegova aplikacija radila očekivano na svim ovim platformama čiji će popis s budućim radom samo rasti, predstavlja veliki napredak u volonterskom računarstvu.

1.4. Doprinosi našeg rada

Implementirani sustav omogućuje da korisnik znanstvenik definira računalno zahtjevan zadatak i prijavi ga za rješavanje. Jedna od raspoloživih radilica preuzeti će zadatak i pokrenuti njegovu obradu. Po završetku proračuna radilica će prijaviti rezultate u sustav koji će kasnije biti vidljivi korisniku koji očekuje njegove rezultate.

No, maksimalno iskorištenje infrastrukture znanstvenik će dobiti tek ako prijavi veliki broj zadataka (npr. desetke tisuća) koji će se potom rasporediti po svim slobodnim radilicama.

Tipičan primjer kada je ovakva infrastruktura potrebna je pokretanje algoritma u razvoju s različitim parametrima kako bi se empirijski pokazalo za koje se parametre algoritam ponaša najbolje. Tako su Čupić, Golub i Jakobović u radu [30] pokretali eksperiment u trajanju od dvije minute u skoro dvije tisuće instanci s različitim kombinacijama parametara. Koristeći razvijeni raspodijeljeni sustav ovakvi bi se eksperimenti mogli pokretati efikasno koristeći pomoć studenata, ostalih djelatnika na sveučilištu ili slobodnih radnih stanica u laboratorijima.

Slijedi lista doprinosa koji bi mogli biti značajni svjetskoj, ali i domaćoj akademskoj zajednici:

- Izgrađen je sustav za paralelno volontersko izvođenje JavaScript zadataka spreman za uporabu
- Razvijena je aplikacija koja koristi sustav, a služi za izvršavanje evolucijskih algoritama
- Razvijena aplikacija je zbog svoje jednostavnosti pogodna kao pomoćni alat u edukaciji

Slijede doprinosi infrastrukturi BOINC koju smo proširili kako bi mogla izvršavati JavaScript zadatke:

- Riješen je problem heterogenih platformi, odnosno isti izvorni kod znanstvene aplikacije može se pokrenuti na svim ciljnim računalima
- Riješen je problem sigurnog izvršavanja programa kojemu se ne vjeruje
- Izvršna verzija okoline za izvršavanje JavaScript-a veličine je ispod 10Mb što je više od deset puta manje u odnosu na veličinu virtualnih slika koje koriste druga rješenja s virtualizacijom
- Volonteri koji ne žele ili ne mogu instalirati BOINC klijentski program mogu sudjelovati u proračunu koristeći samo Internet preglednik

– Pokretanjem našeg rješenja na mrežnom usmjeritelju pokazujemo kako je prijedlog uistinu prenosiv, odnosno moguće ga je pokrenuti čak i na platformama koje nemaju mogućnost instalacije standardnih virtualizacijskih podrški

Glavno ograničenje koje predstavlja naš pristup nužnost je da jezik znanstvenih aplikacija bude JavaScript čije su mane i prednosti opisane u odjeljku 5.1.

1.5. Struktura pisanog rada

Poglavlje 2 opisuje korištene komponente pri izgradnji ovog sustava, odnosno daje detaljan uvid u korištenje BOINC infrastrukture kao i JavaScript okoline V8 te opisuje nekoliko alata korisnih za rukovanje jezikom JavaScript kao i prevođenje drugih skriptnih jezika u ovaj jezik.

Poglavlje 3 predstavlja **jsApp**, implementaciju našeg međusloja za izvršavanje JavaScript zadataka u BOINC aplikacijama te njegove prednosti i nedostatke.

Poglavlje 4 pokazuje proširenje BOINC poslužitelja i klijentsku programsku podršku koja podržava raspodijeljeno računanje u pregledniku.

Poglavlje 5 prezentira razvijenu **optJS** aplikaciju koja koristi superračunalo u svrhe izvođenja evolucijskih algoritama, te prikazuje razna mjerenja pri eksperimentalnim pokretanjima.

Poglavlje 6 predstavlja prijašnji rad na ovom području, dok poglavlje 7 opisuje planove za buduća proširenja i zaključuje ukupan pisani rad.

2. Korištene komponente

Ovo poglavlje opisuje korištene komponente pri izgradnji raspodijeljenog superračunala i čitatelju daje podlogu za razumijevanje poglavlja 3 koje predstavlja našu implementaciju znanstvene aplikacije **jsApp**. Prvi dio opisuje samu infrastrukturu za volontersko računarstvo, dok drugi i treći dio uvode pojmove vezane za korišteni jezik izgradnje - JavaScript.

2.1. Infrastruktura za volontersko računarstvo

BOINC[7] je programska podrška za volontersko računarstvo i računarenje na mreži radnih stanica. Projekti koji koriste BOINC su međusobno nezavisni. To znači da svaka grupa znanstvenika koja želi implementirati ovaj sustav za raspodijeljeno računarstvo pokreće svoje poslužitelje, sustave za rukovanje bazama podataka i traži svoje volontere koji će donirati procesorsku moć. Programska podrška otvorenog je kôda, sastoji se od poslužiteljske i klijentske strane i licencirana je pod LGPL (engl. *GNU Lesser General Public License*). Poslužiteljska programska podrška implementirana je koristeći *C++*, *MySQL*, *PHP* i *Python*, a pokreće se na Linux operacijskom sustavu. Klijentski BOINC program implementiran je koristeći *C++* i pokreće se na svim popularnijim platformama.

Slijedi pojednostavljen opis BOINC sustava. Sustav radi na principu arhitekture klijent-poslužitelj u kojoj klijent može biti bilo koji uređaj spojen na globalnu mrežu. S obzirom na postavke o dozvoljenom opterećenju volonterskog računala, klijentski program nalazi se u zauzetom ili slobodnom načinu rada. U slobodnom načinu rada, klijent ostvaruje HTTP (engl. *HyperText Transfer Protocol*) vezu prema središnjem poslužitelju i traži zadatke za obradu. Ako ima zadataka u redu čekanja, središnji poslužitelj šalje klijentu opisnike zadataka koje treba obraditi. Klijent tada preuzima izvršne datoteke i ulazne podatke te pokreće obradu. Po završetku izvršavanja, klijent šalje izlazne podatke središnjem poslužitelju i zahtjeva nove zadatke.

Na poslužiteljskoj strani pokrenuta je podrška koja poslužuje stranice mrežnog

sjedišta, sustav za rukovanje bazama podataka, te procesi koji rukuju sa zadacima i rezultatima koji se šalju prema klijentima. Mrežno sjedište[1] sadrži osnovne informacije o znanstvenom projektu i kontakte odgovornih voditelja. Ovo sjedište ujedno služi za praćenje statistika i za rukovanje postavkama za korisnike volontere koji imaju više spojenih računala na sustav. Baza podataka predstavlja izvor informacija za sve komponente sustava, a čuva zapise o znanstvenim aplikacijama, zadacima i rezultatima koji su trenutno na obradi.

Glavni entiteti u BOINC bazi podataka su **jedinica posla** (engl. *work unit*) i **rezultat** (engl. *result*) koji predstavljaju zadatak i mjesto za njegov pripadajući rezultat. Svaki zadatak može imati više rezultantnih entiteta koji će onda biti raspodijeljeni na različite radne stanice i nakon obrade uspoređeni. Iako se ovom redundancijom isti rad obavlja više puta, rješava se problem zlonamjernih korisnika koji bi mijenjali rezultate ili korisnika koji nesvjesno pokreću procesorske jedinice s greškom. Ostali mehanizmi koje podržava sustav BOINC su:

- Homogena redundancija (engl. *homogeneous redundancy*) - redundantni zahtjevi za obradu šalju se samo računalima istih platformi i arhitektura. Ovaj mehanizam se koristi zbog činjenice da operacije s pomičnom točkom neće uvijek biti istovjetne na različitim arhitekturama, a sustav pokušava usporediti redundantne zahtjeve oktet po oktet.
- Lokalno raspoređivanje (engl. *locality scheduling*) - optimizacija da se zahtjevi za obradu šalju na one klijente koji kod sebe već imaju neke od datoteka koje predstavljaju ulazne podatke.
- Anonimna platforma (engl. *anonymous platform*) - mogućnost da klijent sam preuzme izvorne kodove znanstvene aplikacije, prevede ih za svoju arhitekturu i tako sudjeluje u znanstvenom proračunu. Korisnici, mahom sigurnosni stručnjaci, koji ne vjeruju voditeljima projekta da je znanstvena aplikacija dobroćudna koriste ovu opciju kako bi bili sigurni da aplikacija ne vrši zloćudne operacije.

2.1.1. Razvoj BOINC znanstvene aplikacije

Znanstvena aplikacija mora biti u nekom od izvršnih oblika kako bi se ona mogla pokrenuti na klijentu prilikom obrade. Isto tako, komunikacija koja će se odvijati između BOINC klijenta i znanstvene aplikacije zahtjeva da aplikacija koristi sučelje (engl. *API - application programming interface*) koje je definirano za programske

[1] U našem slučaju - http://v8boinc.fer.hr

jezike *C++* i *Fortran*. Ovdje navodimo protokol za dio komunikacije između BOINC klijentskog programa i znanstvene aplikacije, odnosno one funkcionalnosti koje koristimo pri izgradnji znanstvene aplikacije koja pokreće JavaScript u poglavlju 3.

Kada znanstvena aplikacija učitava ulazne datoteke, mora komunicirati s BOINC knjižnicom da sazna pravu putanju tražene ulazne datoteke.

Poziv *boinc_resolve_filename* iz BOINC knjižnice prima logičko ime ulazne ili izlazne datoteke i pozivatelju vraća punu putanju prema datoteci.

Druga funkcija, *boinc_fraction_done* poziva se kada znanstvena aplikacija može odrediti koji dio posla je već uspješno obradila, a klijentski program tu informaciju tada pokazuje krajnjem korisniku volonteru.

Nakon što se implementira znanstvena aplikacija, dodatan trud je potreban da se prevede na strojni jezik ciljanih arhitektura i da se osigura njeno uspješno pokretanje u svim operacijskim okolinama. Uložen trud i korišteni alati u našem slučaju opisani su u odjeljku 3.2.

2.1.2. Primjer pokretanja nativne raspodijeljene aplikacije

Ovaj pododjeljak opisuje korake koje znanstvenik poduzima kada želi pokrenuti raspodijeljenu aplikaciju koristeći BOINC. Ovdje pretpostavljamo da je znanstvena aplikacija implementirana u jeziku *C++*, dok je primjer pokretanja JavaScript aplikacije opisan u pododjeljku 3.3.

BOINC poslužitelj je skup programskih alata koje je potrebno instalirati na nekoj od Linux inačica. U našem slučaju odabrali smo inačicu *Debian 7.1* i proveli cjelokupan proces postavljanja na poslužitelju *v8boinc.fer.hr*.

Prvi korak je priprema znanstvene aplikacije opisana u pododjeljku 2.1.1.

Izvršni oblik aplikacije se prijavljuje u BOINC sustav i tom prilikom se njen binarni kôd digitalno potpisuje privatnim ključem projekta. Za ovu prijavu znanstvenik mora imati pristup ljuski operacijskog sustava središnjeg poslužitelja i mora imati razumijevanje BOINC alata. Imajući digitalni potpis, klijentski program može sa sigurnošću znati da je izvršna aplikacija preuzeta s poslužitelja nepromijenjena u komunikacijskom kanalu.

Napomenimo da je prilikom svake izmjene aplikacije ponovno potrebno proći korake prilagođavanja za ciljne platforme, prevođenja i prijave aplikacije.

Nakon što je aplikacija prijavljena, moguće je prijaviti ulazne podatke i definirati jedinice posla. Nakon definicije jedinica posla, BOINC preuzima odgovornost da se ona izvrši na nekoj radilici, i rezultati su vidljivi znanstveniku.

2.2. Programski jezik JavaScript

Ispravno ime korištenog jezika je ECMAScript, iako ćemo ime JavaScript koristiti kao sinonim za isti jezik. ECMAScript je originalno dizajniran kao skriptni jezik za Internet stranice omogućujući razvojnim stručnjacima da mijenjaju sadržaj stranice bez njenog ponovnog učitavanja. ECMAScript je nadgradnja na JavaScript koji je nastao 1995. u tvrtki Netscape kao dio Internet preglednika Navigator 2.0. Kasnije je jezik standardiziran i moderniziran, a u trenutku pisanja aktualan je standard ECMAScript-262, peto izdanje[26].

Konkurentnost na tržištu Internet preglednika preslikava se i na konkurentnost između izvršnim okolinama (engl. *engine*) za JavaScript. Najpoznatije takve okoline danas su *SpiderMonkey*, *Nitro* i *V8* napravljeni u organizacijama Mozilla, Apple i Google respektivno.

Zbog otvorenosti koda, dostupnosti dokumentacije i obećavajućim performansnim izvještajima, mi smo se odlučili na korištenje pokretača V8.

Najnovija istraživanja o prevođenju međukoda direktno u podskup jezika JavaScript omogućuju da se drugi programski jezici prevedu u JavaScript[28]. Nadalje, koristeći se ovakvim prevoditeljem znanstvenici su uspjeli programe u jeziku *Python* pokrenuti koristeći izvršnu okolinu za jezik *JavaScript*. Uz sve veći i veći broj matematičkih knjižnica koje se prevode u jezik JavaScript, smatramo da bi ovaj jezik mogao u budućnosti postati popularan i u akademskoj zajednici.

2.3. V8 - JavaScript okolina za izvršavanje

V8 implementiran je u programskom jeziku *C++*, otvorenog je koda i licenciran pod *BSD* licencijom. Izvršna okolina koristi prevođenje *baš kada treba* (engl. *just in time*) i opremljena je raznim heuristikama za optimizaciju programskih isječaka. Budući da je dio Internet preglednika Google Chrome koji je prenosiv među platformama, sam V8 je također dizajniran da bude prenosiv. Ostale značajke ove programske podrške su brzina izvođenja i sigurnost. Pod sigurnosti se smatra otpornost na nepovjerljive programske isječke napisane u JavaScript-u koji bi htjeli naštetiti operacijskom sustavu u kojem se V8 okolina pokreće. Cijela podrška napisana je kao knjižnica funkcija (engl. *library*). Program koji želi koristiti njenu funkcionalnost uključuje *v8.h* zaglavlje i povezuje se (engl. *link*) s prevedenim datotekama knjižnice.

Knjižnicu V8 uspješno smo preveli za Windows, OSX, i Linux operacijske sustave pokrenutim nad *Intel* procesorima. Na operacijskom sustavu Linux, dodatno smo

preveli knjižnicu i za *ARM* i *MIPS* arhitekture.

C++ program koji koristi V8 kako bi učitao datoteku s JavaScript izvornim kôdom i pokrenuo je bez provjera iznimnih stanja sadrži svega petnaestak linija[2] što pokazuje jednostavnost korištenja ove knjižnice.

JavaScript program koji se pokreće pod V8 pokretačem nema dodirnih točaka s okolinom osim globalnog objekta. U taj objekt *C++* razina, odnosno omotač, može umetnuti definicije funkcija ili podataka za koje želi da unutarnji JavaScript program dobije pristup.

[2]https://developers.google.com/v8/get_started

3. Aplikacija omotač za izvršavanje JavaScript programa

Ovo poglavlje predstavlja **jsApp**, razvijeni BOINC omotač za pokretanje JavaScript zadataka. Opisujemo ciljeve, dizajn, implementaciju, ograničenja i prednosti našeg omotača.

Koraci koji su potrebni kako bi se iskoristila BOINC infrastruktura s nativnim aplikacijama opisani su u pododjeljku 2.1.2. Najveće prepreke koje se nameću su prilagodba aplikacije na sve ciljne platforme i nužnost ponovnog prevođenja prilikom izmjene funkcionalnosti aplikacije. Drugi istraživači su već zaključili kako je moguće koristiti virtualizacijsku tehnologiju da se ove prepreke izbjegnu[15, 24, 25]. Iako takva rješenja nude primamljive prednosti poput razvoja aplikacije na jednoj platformi i pokretanja te platforme unutar virtualizacijskog hipervizora na klijentu, sagledavši prednosti i nedostatke potpune virtualizacije postavili smo sljedeće ciljeve za naš omotač:

- Znanstvena aplikacija bi trebala imati samo jedan izvorni kôd koji se može pokrenuti na svim okolinama
- Sva okolina koja je potrebna pri pokretanju znanstvene aplikacije trebala bi se dostaviti klijentu sa zadatkom, odnosno ne želimo pretpostavljati koju programsku podršku volonter već ima ugrađenu
- Količina podataka koja se šalje klijentu ne bi trebala premašiti 100Mb
- Zahtjev za obradu ne bi trebao moći naštetiti klijentskom računalu
- Vrijeme izvršavanja aplikacije unutar omotača ne bi trebalo biti značajno veće od izvršavanja nativne aplikacije s istom funkcionalnošću
- Nove verzije znanstvene aplikacije trebale bi se moći poslati na obradu bez nužnog detaljnog poznavanja BOINC infrastrukture i pristupa ljuski središnjeg poslužitelja

JavaScript je izabran kao jezik izgradnje aplikacija koje će se koristiti na našem

raspodijeljenom superračunalu. Ovaj jezik je nezavisan o arhitekturi i platformi na kojoj se pokreće i iz njega nije moguće napraviti poziv prema jezgri operacijskog sustava koji bi mogao narušiti njegovu sigurnost. Odlučili smo da će svaka znanstvena aplikacija prilikom prijave u sustav biti sažeta u jednu *.js* datoteku koja za ulaznu točku ima funkciju *main*. Funkcija *main* unutar skripte, primit će sve ulazne podatke kao jedan objekt koji je njen jedini argument. Sve izlazne podatke funkcija *main* vratit će kao povratnu vrijednost. Program omotač, odnosno **jsApp**, pobrinut će se za serijalizaciju i deserijalizaciju tih podataka u datoteke na disku i potrebnu komunikaciju s BOINC sučeljem. Jedini kontakt znanstvene aplikacije s okolinom biti će funkcija *boinc_fraction_done* koju će omotač ubaciti u globalni objekt skripte. Ta funkcija služit će kao posrednik prema BOINC knjižnici za objavljivanje napretka u obradi podataka.

Kako bi razvili omotač koji će pokrenuti znanstvenu aplikaciju napisanu u jeziku JavaScript, koristili smo *C++* te BOINC i V8 knjižnice. Cijeli omotač ima manje od 250 redaka izvornog kôda koje sadrže potpuno dojavljivanje iznimnih stanja što potvrđuje jednostavnost njegove izvedbe.

3.1. Implementacijski detalji omotača

Algoritam 1 prikazuje korake koje izvodi omotač. **jsApp** očekuje dvije datoteke u radnoj mapi - **main.js** i **in.json** ili poveznice na njih koje će BOINC knjižnica moći razriješiti. Omotač učitava i parsira obje datoteke. Prva datoteka sadrži JavaScript izvorni kôd znanstvene aplikacije i u njoj tražimo definiciju funkcije *main*. Datoteka **in.json** sadrži serijalizirane ulazne podatke u JSON (engl. *JavaScript object notation*) obliku, koje omotač prevodi u nativni JavaScript objekt. Ako funkcija *main* ne postoji, ili izvorni kôd bilo koje datoteke sadrži sintaksnu grešku, izvršavanje se zaustavlja i greška je prijavljena.

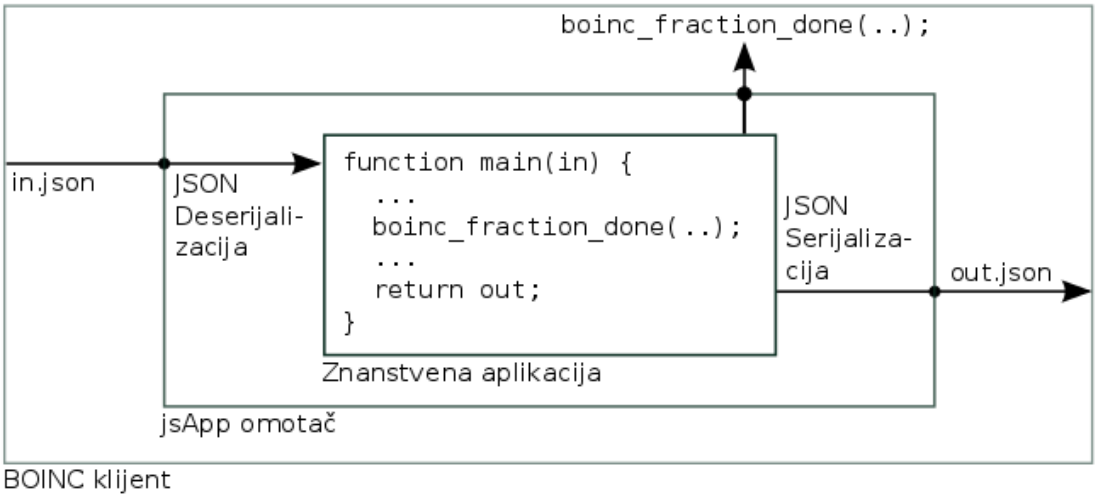
Omotač tada poziva JavaScript funkciju *main* dajući joj deserijalizirane ulazne podatke kao jedini argument, čime počinje znanstveni proračun. Ako se tijekom proračuna dogodi iznimka koja nije obrađena u znanstvenoj aplikaciji, omotač je zaustavlja, prijavljuje grešku i završava obradu. Kada funkcija *main* normalno završi s radom, omotač preuzima kontrolu izvođenja te njenu povratnu vrijednost serijalizira i zapisuje u datoteku **out.json**.

Slika 3.1 pokazuje kako je ulazna datoteka transformirana u izlaznu datoteku pod djelovanjem znanstvene aplikacije unutar omotača kojeg pokreće BOINC klijentski program na volonterskom računalu.

Algoritam 1 jsApp pseudokod

$main\_js := ucitaj\_datoteku(\text{"}main.js\text{"});$
$main := pronadji\_definiciju\_funkcije(\text{"}main\text{"}, main\_js);$

$in\_json := ucitaj\_datoteku(\text{"}in.json\text{"});$
$in := deserijaliziraj(in\_json);$
$out := main(in);$
$out\_json := serijaliziraj(out);$
$zapisi\_datoteku(\text{"}out.json\text{"}, out\_json);$



Slika 3.1: Znanstvena aplikacija unutar omotača kojeg je pokrenuo BOINC klijentski program

Kada klijent od poslužitelja dobije opisnik zadatka, počinje preuzimanje izvršne datoteke **jsApp** za platformu na kojoj se klijent pokreće. Osim izvršne datoteke omotača, **main.js** i **in.json** se također preuzimaju kao dio zahtjeva za obradom.

Kada je preuzimanje svih datoteka uspješno završeno, klijentski BOINC program pokreće preuzetu aplikaciju omotača u zasebnoj radnoj mapi u kojoj su još i poveznice na dvije ulazne datoteke. **jsApp** omotač izvršava gore opisane korake kako bi stvorio **out.json** i završava s radom, nakon čega BOINC klijentski program šalje rezultate središnjem poslužitelju.

Ako aplikacija tijekom izvršavanja pozove globalnu funkciju *boinc_fraction_done*, omotač će preusmjeriti taj poziv BOINC klijentu koji će naposljetku tu informaciju prikazati volonteru kroz grafičko sučelje.

3.2. Prenosivost aplikacije na ciljne platforme

Pri prevođenju **jsApp** aplikacije na razne arhitekture i platforme koristili smo se originalnim uputama BOINC korisničkog priručnika[1].

U trenutku pisanja **jsApp** je preveden i temeljito testiran za platforme Windows, Mac OS X, i Linux na *Intel* procesorima. Osim toga, kako bi provjerili prenosivost, odabrali smo mrežni usmjeritelj na MIPS arhitekturi kao testni uređaj koji ima samo rudimentarni Linux operacijski sustav i nema gotovo nikakve zajedničke knjižnice (engl. *shared libraries*).

Statičko povezivanje izvršne datoteke podrazumijeva da strojni kod sadržan u rezultatu sadrži definicije svih knjižnica koje aplikacija koristi tijekom izvršavanja. Na operacijskom sustavu Linux, pri statičkom povezivanju može doći do problema s nekonzistentnom verzijom C knjižnice (engl. *libc*) i same jezgre operacijskog sustava (engl. *kernel*). Budući da je jezgra sustava potpuno unazadno kompatibilna, rješenje problema je odabir starije implementacije C knjižnice, njeno prevođenje i povezivanje izvršne datoteke s tako prevedenom knjižnicom. Pri prevođenju **jsApp** omotača za operacijski sustav Linux, korištena je slika operacijskog sustava iz rujna 2007. godine[2] što nam omogućuje da se izvršna datoteka može pokrenuti na Linux sustavima čija je verzija osvježeni barem jednom u zadnjih 7 godina.

Iako statičko povezivanje aplikacije implicira povećanu veličinu izvršne datoteke, njihove veličine za svaku platformu ne premašuju 10Mb što pokazuje tablica 3.1.

[1] http://boinc.berkeley.edu/trac/wiki

[2] Verzija Linux jezgre 2.6.18

Tablica 3.1: Veličine izvršne datoteke **jsApp** po platformama

Arhitektura	Platforma	Veličina izvršne datoteke
Intel	i386 Darwin (32 bitni OS X)	8.1 Mb
MIPS	i686 Linux (32 bitni Linux)	7.1 Mb
Intel	i686 Linux (32 bitni Linux)	5.5 Mb
Intel	x86 Windows (32 bitni Windows)	4.5 Mb

Tijekom testiranja nismo pronašli računalo, odnosno uređaj na kojem se aplikacija nije mogla pokrenuti.

Za sada su sve izvršne datoteke prevedene samo za 32 bitne sustave, a 64 bitni sustavi ih mogu pokretati u modu kompatibilnosti. Ako buduća testiranja pokažu da 64 bitni omotač ima bolje performanse na 64 bitnom sustavu uložit ćemo dodatan trud i za prevođenje na takve platforme.

3.3. Primjer pokretanja raspodijeljene *jsApp* aplikacije

Prijavljivanje aplikacije omotača u BOINC sustav potrebno je napraviti samo jednom i u tom trenutku se provode koraci poziva BOINC alata i digitalnog potpisivanja koji zahtijevaju pristup ljusku središnjeg poslužitelja. Budući da je **jsApp** prilagođen za ciljne platforme, aplikacija je prikladno prijavljena u BOINC sustav na poslužitelju *v8boinc.fer.hr*.

Projekt je sada spreman primati zadatke definirane u JavaScript jeziku s ulaznim podacima u JSON obliku. Za prijavu zadataka, znanstvenik koji se koristi sustavom ne mora imati pristup prema ljuski već zadatak može prijaviti koristeći mrežni servis. Isto tako, nove verzije aplikacija ne zahtijevaju dodatne korake budući da su aplikacije dio ulaznih podataka.

Postavljeni mrežni servis je HTTPS poslužitelj koji odgovara na *POST* upite u kojima su **main.js** i **in.json** poslani kao datoteke iz obrasca. Odgovor na ovaj zahtjev je poveznica na kojoj je vidljiv napredak prijavljenog zadatka i na kojoj će po završetku obrade biti moguće preuzeti rezultate.

Ograničenje našeg pristupa u razvoju omotača je nužnost da znanstvena aplikacija bude implementirana u jeziku JavaScript. Ta činjenica će donekle usporiti izvršavanje aplikacije što je prikazano u poglavlju 5.

Slanjem statički povezane izvršne datoteke omotača volonterima ne moramo pretpostavljati ništa o njegovoj ciljnoj okolini i ugrađenoj programskoj podršci što je velik

napredak u odnosu na povezana istraživanja prikazana u poglavlju 6.

Iznenađujuće mala veličina izvršne datoteke omotača neće opteretiti ni klijentsku ni poslužiteljsku mrežu.

Pokretanjem znanstvene aplikacije u omotaču koji nasljeđuje sigurnosne karakteristike Internet preglednika Google Chrome, nije izgledno da bi i promijenjeni oblik znanstvene aplikacije mogao naštetiti računalu volontera.

Svi problemi vezani za heterogenost platformi i arhitektura su riješeni svođenjem znanstvene aplikacije na prenosivi jezik JavaScript.

Isto tako, budući da su znanstvene aplikacije implementirane u programskom jeziku koji potječe iz svijeta Interneta, logično je zapitati se kako je moguće proširiti BOINC raspodijeljeni model da podržava klijente koji koriste isključivo Internet preglednik.

4. Iskorištavanje preglednika u raspodijeljenom računarstvu

Ovo poglavlje opisuje proširenje JavaScript raspodijeljenog računarstva na BOINC platformi koje volonterima omogućava sudjelovanje u znanstvenim projektima koristeći isključivo Internet preglednik.

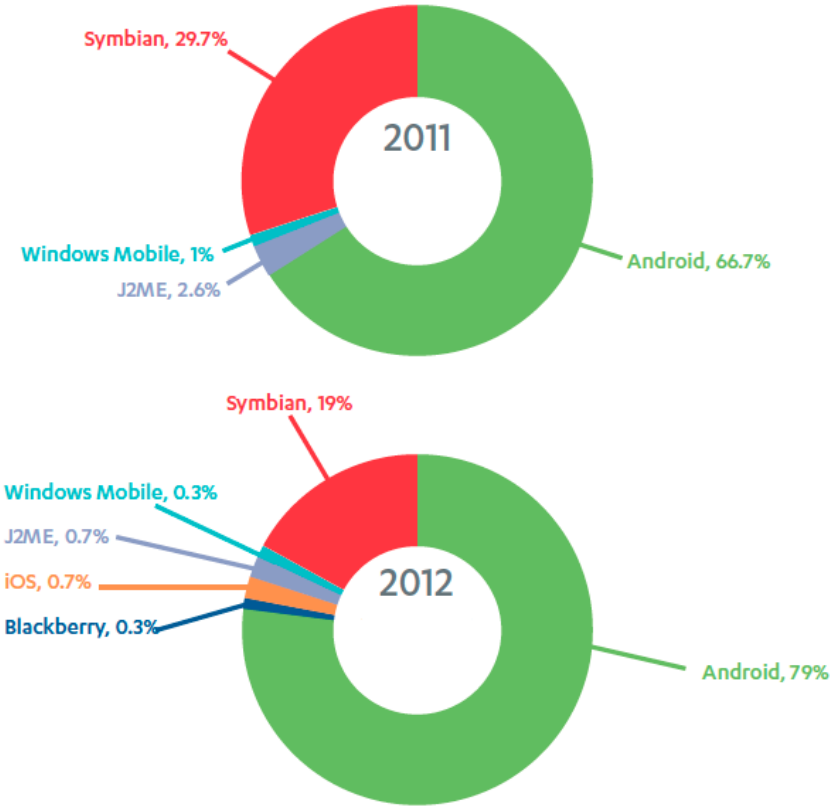
Poglavlje otvaramo navođenjem prepreka koje se mogu ispriječiti volonteru koji želi donirati dio procesorskog vremena znanstvenom projektu. Slijedi opis postojeće rješenja za raspodijeljeno računarstvo u preglednicima GridBee[27] te naš pristup s posrednicima. Poglavlje zatvaramo uspoređivanjem dvaju pristupa i zaključcima.

4.1. Moguće prepreke pri sudjelovanju u znanstvenom projektu

Ugrađivanje BOINC klijentske aplikacije kod korisnika volontera može biti prepreka zbog nekoliko razloga.

Jedan od njih je pravilo propisano od strane BOINC voditelja na sveučilištu u Berkeleyu da volonter smije ugraditi BOINC klijentsku programsku podršku samo na računalima koje on posjeduje ili uz eksplicitnu dozvolu vlasnika računala. Ovo pravilo vrlo striktno odvaja BOINC programe od zlonamjerne programske podrške (engl. *malware*) koja uz razne mehanizme može postati dio korisnikovog operacijskog sustava bez njegova znanja. Po ostalim kriterijima, BOINC klijentski program zaista je sličan zloćudnoj programskoj podrški. Naime preuzimanje izvršnih datoteka i njihovo pokretanje čest je obrazac ponašanja zlonamjernih programa[24].

Osim ove logističke prepreke, postoje i uređaji za koje BOINC klijentski program još uvijek nije preveden. Kao primjer navodimo skup pametnih telefona i pločica (engl. *smartphone, tablet*) proizvedenih u tvrtki Apple. Sigurnosni model njihovog operacijskog sustava *iOS* ne dozvoljava da aplikacija preuzme izvršni strojni kôd s Interneta i pokrene ga lokalno. Jedna od posljedica ovako restriktivne sigurnosne poli-



Slika 4.1: Usporedba broja pronađenih zloćudnih programa na mobilnim platformama Slika je preuzeta s portala *http://www.zdnet.com*.

tike je činjenica da je broj zloćudnih programa na *iOS* operacijskom sustavu zaista malen što pokazuje slika 4.1. Stoga, BOINC klijentski program na ovom operacijskom sustavu ne može obavljati svoju osnovnu funkciju. Istraživači koji su ispitivali performanse *iPhone* uređaja u svrhu raspodijeljenog računarstva koristeći BOINC[10], uređaj su prvo morali otključati (engl. *jailbreak*).

Vjerojatno postoji i frakcija korisnika volontera kojima je ugradnja novog programa na njihovo računalo dovoljna prepreka da se predomisle o donaciji svoje procesorske moći. Mogućnost da korisnik volonter sudjeluje u znanstvenom proračunu jednostavnim otvaranjem Internet stranice nedvojbeno će povećati bazu korisnika volontera.

Dizajnom arhitekture za raspodijeljeno računarstvo u pregledniku pokušali smo se držati sljedećih ciljeva.

Volonter koji sudjeluje u proračunu koristeći preglednik ne bi trebao prouzročiti više opterećenja na središnjem poslužitelju od korisnika koji sudjeluje u proračunu koristeći standardnu klijentsku podršku. Svi resursi na računalu volontera trebali bi se

jednako moći iskoristiti i ograničiti u pregledniku kao i u službenoj klijentskoj implementaciji.

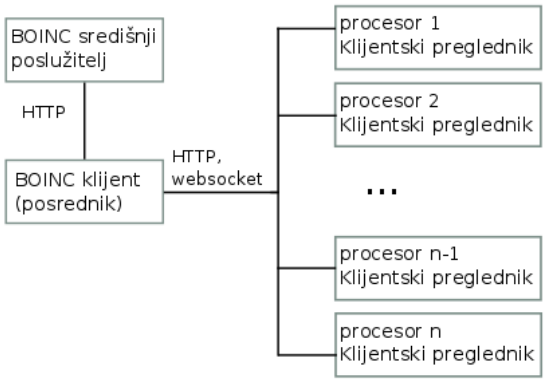
4.2. Postojeća implementacija *GridBee*

Raspodijeljeno računarstvo u preglednicima već je istraživano i nekolicina projekata su dohvatljiva na javnoj mreži. Projekt nazvan *GridBee*[27] nama je zanimljiv jer je nadgradnja na BOINC infrastrukturu. Iako se u literaturi ne spominje iskorištavanje jezgre Internet preglednika kao virtualna okolina za izvršavanje JavaScript zadataka van preglednika, ovaj projekt je vrlo sličan našemu u pogledu raspodijeljenog računanja koristeći Internet preglednike.

GridBee programska podrška implementira većinu funkcionalnosti BOINC klijentskog programa koristeći JavaScript i HTML5 te se pokreće u Internet pregledniku. Uz malu promjenu postavki na središnjem poslužitelju ovaj klijent unutar preglednika može obavljati sve osnovne funkcije BOINC klijentske podrške. Volonter koji želi sudjelovati u projektu čije su znanstvene aplikacije implementirane u JavaScript-u, može otvoriti *GridBee* početnu Internet stranicu, upisati adresu središnjeg poslužitelja, svoje pristupne podatke i sudjelovati u računanju.

Implementacija klijenta unutar preglednika moguća je budući da BOINC koristi HTTP protokol za svu komunikaciju između poslužitelja i klijenta. Unutar preglednika, HTTP upite moguće je odašiljati koristeći AJAX (engl. *Asynchronous JavaScript and XML*) tehnologiju. Budući da svaki klijentski program mora središnjem poslužitelju prijaviti svoju platformu, *GridBee* bira za platformu novo ime **javascript**. Kada naiđe na klijenta koji pokreće **javascript** platformu, središnji poslužitelj zna da se radi o klijentu u pregledniku i dijeli mu samo zadatke izgrađene u JavaScript jeziku. Kada dobije novi zadatak, *GridBee* klijent pokreće njegovo izvršavanje u novoj dretvi koristeći HTML5 tehnologiju *WebWorker*[20].

Iako je projekt pažljivo implementirao dovoljno BOINC komunikacijskog protokola da obavlja osnovne funkcionalnosti raspodijeljenog računanja, neke komponente ipak nisu u potpunosti dovršene. Jedna takva funkcionalnost je mogućnost BOINC klijenata da ne opterećuju središnji poslužitelj ponavljajućim upitima u slučaju iznimnih stanja. Tipičan primjer rješenja ovog problema naziva se eksponencijalno udaljavanje (engl. *exponential back-off*). Također očekujemo da korisnici volonteri iza Internet preglednika pokažu drugačije obrasce ponašanja nego korisnici koji imaju ugrađen BOINC klijent. Na primjer očekujemo mogućnost naglog naviranja korisnika u velikom broju ili često dolaženje i odlaženje sa stranice, na što treba obratiti pažnju.



Slika 4.2: Arhitektura za računanje u pregledniku s posrednikom.

Svi ovi razlozi ukazuju na to da bi *GridBee* pristup mogao imati smanjenu skalabilnost, te smo odlučili krenuti pomalo drugačijim pristupom.

Smatramo da bi rješenje u kojem je originalni BOINC klijent pokrenut u okolini preglednika moglo imati bolje značajke ako se službeni izvorni kôd klijentske programske podrške pomoću *emscripten* prevoditelja prilagodi za JavaScript i HTML tehnologije. Iako smo ova razmatranja ostavili za budući rad, u međuvremenu smo implementirali skalabilan sustav koji omogućuje raspodijeljeno računarstvo u Internet preglednicima, a koristi se novom komponentom u sustavu - posrednikom.

4.3. Raspodijeljeno računarstvo u preglednicima koristeći posrednike

Posrednik (engl. *proxy*) komponenta pokreće standardnu BOINC klijentsku programsku podršku, sa standardnim protokolima komunicira sa središnjim poslužiteljem pri dohvaćanju zadataka, no za njihovu obradu ne koristi vlastite procesorske jedinice već procesore od trenutno spojenih klijenata u preglednicima.

Arhitektura sustava prikazana je na slici 4.2. Posrednik je komponenta koja može i ne mora biti na drugačijoj lokaciji od središnjeg poslužitelja. U sustavu ih može biti više i vatrozidnim pravilima moguće je regulirati koji sve korisnici joj smiju pristupati.

Kao što je već rečeno, posrednik pokreće standardni BOINC klijentski program, no uz pomoć BOINC opcije **anonimne platforme** (engl. *anonymous platform*) on ne pokreće znanstvenu aplikaciju koju preuzima od poslužitelja, već svoju promijenjenu inačicu aplikacije **jsApp**. Posrednik pokreće javno vidljiv HTTP mrežni poslužitelj na koji se klijenti mogu spajati i očekivati zahtjeve za računanje. Broj trenutno spojenih i spremnih veza prema klijentima prijavljuje središnjem poslužitelju kao svoj broj pro-

cesorskih jedinica. Dakle, posrednik odjednom odigrava ulogu i klijenta i poslužitelja.

Kada posrednik dobije opisnik zadatka od središnjeg poslužitelja, njegova promijenjena inačica znanstvene aplikacije preusmjeri zahtjev za obradu jednom od trenutno spojenih klijenata preglednika i zaustavi svoje izvršavanje (engl. *sleep*). Tek aplikacija u pregledniku koja prima zahtjev za obradu istoga i pokreće koristeći novu dretvu. Kada je obrada gotova, aplikacija to dojavljuje posredniku podizanjem izlaznih podataka do njegovog mrežnog poslužitelja. Mrežni poslužitelj izlazne podatke zapisuje u radnu mapu zaustavljene promijenjene znanstvene aplikacije koja je bila zadužena za obradu te jedinice te nastavlja njeno izvršavanje slanjem signala. Znanstvena aplikacija provjeri postojanje izlaznih podatka te završava s radom. Klijentski BOINC program na posredniku sada nastavlja normalan tok izlaznih podataka do središnjeg poslužitelja kao da je znanstvena aplikacija lokalno napravila obradu, bez da je koristila spojene klijente u preglednicima.

U jednoj rečenici, naš pristup temelji se na arhitekturi gdje su klijentski preglednici spojeni na jednog od posrednika koji se prema središnjem poslužitelju ponaša kao i svaki drugi BOINC klijent, no s promjenjivim brojem procesorskih jedinica.

4.4. Usporedba dvaju pristupa

Prednosti našeg pristupa uglavnom su okrenute prema skalabilnosti i sigurnosti.

Umetanjem posredničke komponente između središnjeg poslužitelja i klijenata u preglednicima, rasteretili smo središnji podsustav za raspoređivanje jedinica posla od komunikacije s mnoštvom nepouzdanih klijenata. Nadalje, iako posrednička komponenta može biti preopterećena na isti način koji bi opteretio i središnji poslužitelj, lako je vidjeti da arhitektura podržava podizanje više takvih komponenata. Svaka komponenta tada bi se štitila nekom od tehnika uravnoteživanja opterećenja, poput korištenja višestrukih DNS (engl. *Domain Name Server*) zapisa.

Sigurnosno gledano, smatramo da bi u slučaju većeg broja klijenata u preglednicima bilo dobro postaviti posredničke instance koje će posebno propuštati klijente iz mreže akademske zajednice *CARNet*, ostale klijente iz Republike Hrvatske te ostale klijente s cjelokupnog Interneta. Razlog ovome je što BOINC središnji poslužitelj za svakog klijenta održava u strukturama podataka njegovu mjeru pouzdanosti, odnosno vjerojatnost da će klijent na zahtjev za obradu odgovoriti ispravnim rezultatom. U slučaju pokušaja napada na nekog od posrednika brzim slanjem krivih rezultata ili bilo kojim napadom nedostupnosti, cjelokupni raspodijeljeni sustav se neće zaustaviti jer će središnji poslužitelj ubrzo primijetiti kako jedan od klijenata posrednika prestaje biti

pouzdan i jedinice posla se neće prema njemu dalje dijeliti.

Osim sigurnosnih i skalabilnih značajki, smatramo da je prednost našeg pristupa u odnosu na *GridBee* jednostavnost implementacije i robusnost. Naime, naš pristup ne zahtijeva ponovno implementiranje BOINC protokola već se koristi standardna BOINC klijent-poslužitelj komunikacija.

Trenutno najveća mana u arhitekturi koja koristi posredničke komponente je činjenica da korisnici volonteri nisu autenticirani pred središnjim poslužiteljem što im uskraćuje mjerenje statistika o uloženom procesorskom vremenu za projekt. Nadalje, iako nismo morali nanovo implementirati klijentsku BOINC podršku, naš pristup je puno kompliciraniji u pogledu mrežne administracije.

Pokazali smo kako je BOINC raspodijeljeni model s JavaScript zadacima moguće proširiti s doprinosima od volontera koji nemaju ugrađenu klijentsku BOINC podršku već koriste samo Internet preglednik. Ovo uključuje i volontere s uređajima na koje BOINC klijentski program još nije preveden. Iako slična rješenja za raspodijeljeno računanje u preglednicima već postoje, naš pristup maksimalno iskorištava već napisane i dobro testirane dijelove BOINC programske podrške, dok preglednike iskorištavamo samo kao procesorske jedinice.

Iako je *GridBee* pokazao slična rješenja u izvršavanju raspodijeljenih zadataka u jeziku JavaScript, napominjemo kako naš cjelokupni sustav podržava i volontere koji ne koriste preglednik već standardni BOINC klijentski program. Takvi klijenti biti će primjerice ugrađeni na laboratorijskim računalima jer se tada obrada podataka može pokretati čak i kada korisnik nije prijavljen u sustav.

Drugim riječima, u slučaju **jsApp** omotača opisanog u poglavlju 3, Internet preglednici samo su dodatna platforma prikladna za naš raspodijeljeni model, uvjerljivo nadmašujući prenosivost znanstvene aplikacije iz svih prethodnog istraživanja.

5. Rezultati

Kako bi mogli predstaviti rezultate eksperimentalnih pokretanja raspodijeljenog superračunala odlučili smo razviti jednostavnu, ali korisnu aplikaciju koja crpi njegovu procesorsku moć.

Poglavlje otvaramo odjeljkom 5.1 koje opisuje rezultate vezane za ispitivanje brzine izvršavanja JavaScript jezika.

Odjeljak 5.2 opisuje ciljeve i implementacijske detalje **optJS** programske podrške.

Odjeljak 5.3 prikazuje rezultate koji su dobiveni pokretanjem **optJS** eksperimenata nad razvijenim superračunalom.

5.1. Mjerenje brzine izvršavanja JavaScript programa

JavaScript izvršna okolina V8 koristi prevođenje u strojni kod baš kada treba (engl. *just in time compilation*) i stoga je očekivano da se performansno ponaša bolje od ostalih skriptnih jezika, a sporije od prevođenih jezika.

Mjerenja i usporedbe za brzinu izvođenja svih programskih jezika istraživači su već temeljito izvršili. Iz izvora [16] očitavamo kako je JavaScript jezik prosječno 4 puta sporiji od *C++* jezika, dok je Python prosječno 13 puta sporiji od JavaScript-a.

Kako se pokretačke okoline stalno razvijaju i nastoje biti sve kvalitetnije, očekivano je da će se JavaScript s vremenom približiti vremenu izvršavanja nativnih, odnosno *C++* aplikacija.

Usporenje nativne aplikacije zbog pokretanja unutar potpuno virtualiziranog operacijskog sustava istraženo je u [13]. U radu je vidljivo kako će *C++* aplikacija pokrenuta ispod virtualizacijskog hipervizora i dalje biti mnogo brža od ekvivalentne JavaScript aplikacije.

5.2. *optJS* - raspodijeljeni evolucijski algoritmi

Motivacija za izgradnju ove aplikacije rješavanje je problema odabira parametara za prototip evolucijskog algoritma. Naime, vrlo često se određeni evolucijski algoritam ponaša značajno drugačije s obzirom na početne uvjete ili parametre samog algoritma. Iako se ovaj problem često rješava koristeći iskustvo iz prijašnjih pokretanja sličnih algoritama, mi dajemo jednostavan način da se prototip algoritma implementira u JavaScript programskom jeziku te da se precizno testira njegovo ponašanje s obzirom na različite parametre.

Ciljevi koje smo imali prilikom razvoja aplikacije su:

- Jednostavnost korištenja i intuitivnost sučelja
- Mogućnost da aplikacija od sloja za računanje nad kojim se pokreće zatraži proizvoljno veliku količinu procesorskog vremena
- Mogućnost pokretanja algoritma s mnoštvom različitih parametrima
- Vizualizacija dobivenih rezultata

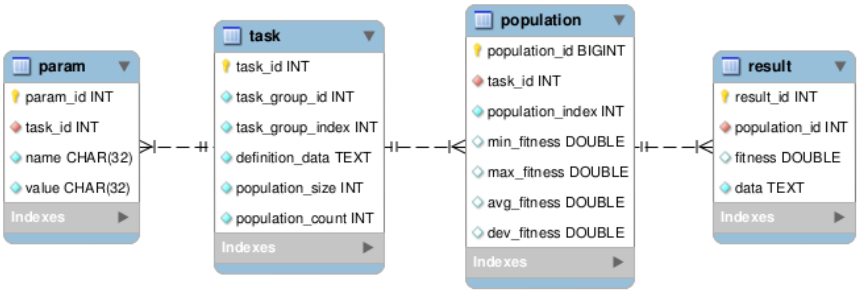
5.2.1. Osnove evolucijskog računarstva

Kako bi opisali računalni model koji koristi razvijena **optJS** aplikacija potrebno je uvesti neke osnovne pojmove o evolucijskim algoritmima.

Evolucijsko računarstvo područje je istraživanja unutar računarske znanosti[14]. Baš kao što ime i naznačuje, inspiraciju dohvaća iz prirodne evolucije. Nije čudno da su znanstvenici pokušali oponašati ovu prirodnu pojavu budući da su njeni rezultati u prirodi impresivni. Evolucijsko računarstvo koristi se specifičnim načinom rješavanja problema koje se temelji na slučajnim procesima, pokušajima i pogreškama.

Jedinka je neki zapis rješenja problema koji se rješava, npr. vektor iz domene višekriterijske funkcije koja se pokušava minimizirati. Za svaku jedinku računa se vrijednost njezine dobrote ili inverzne dobrote. U svakom trenutku evolucijski algoritam pamti populaciju jedinki. Cilj algoritma je nekom perturbacijama jedinki dobiti jedinku s maksimalnom dobrotom, odnosno minimalnom inverznom dobrotom. Perturbacije, tj. operatori koji se koriste inspirirani su prirodnim silama evolucije - selekcija, mutacija, reprodukcija i rekombinacija.

Za opis **optJS** aplikacije, nije potrebno navoditi detalje pojedinog operatora, već je važno imati na umu da se populacija na neki način stalno pokušava poboljšati u vidu dobrote.



Slika 5.1: Dijagram baze kojom se koristi **optJS**

5.2.2. Računalni model *optJS* aplikacije

Entiteti korišteni u **optJS** bazi podataka su: **parametar** (engl. *param*), **algoritam** (engl. *task*), **jedinka** (engl. *result*) i **populacija** (engl. *population*).

Jedno pokretanje algoritma transformira ulaznu populaciju T u T'. Za aplikaciju nije bitno kakva su svojstva novonastale populacije, već samo da takva populacija može biti ponovni ulaz u novu instancu algoritma. Iterativnim pozivanjem istog algoritma na početnoj populaciji stvara se niz populacija $T_0, T_1, \ldots, T_n$.

Pojednostavljeni ER model (engl. *entity-relationship model*) baze podataka aplikacije prikazan je na slici 5.1. Jedinka je JavaScript objekt kojeg zapisujemo pomoću JSON serijalizacije, dok dobrotu pamtimo kao decimalan broj. Algoritam bira kako će se jedinka prikazati u memoriji. Populacija je skup jedinki, i za nju se pamte korisne statističke vrijednosti. Svaka populacija pripada točno jednom algoritmu, odnosno za algoritam se pamti njegov niz populacija. Indeks populacije (engl. *population index*) predstavlja redni broj populacije u tom nizu.

Uz algoritam zapisujemo i parametre koje može koristiti njegova implementacija, broj jedinki u svakoj populaciji i trenutni broj populacija u nizu pokretanja. Implementacija algoritma je JavaScript izvorni kôd sadržan u polju *definition_data*. Da bi algoritam bio valjan, implementacija mora sadržavati dvije funkcije predefiniranog sučelja: *evaluate(Jedinka x)* i *main(Niz<Jedinka> populacija)*. Implementacija funkcije *evaluate* određuje dobrotu neke jedinke, dok funkcija *main* provodi jednu iteraciju algoritma, odnosno transformira populaciju.

task_group_id i *task_group_index* prate grupiranje algoritama. Naime, grupa algoritama nastaje kada jedan algoritam želimo pokrenuti na više različitih parametara ili s različitim početnim populacijama. *task_group_id* je identifikator grupe, dok je *task_group_index* redni broj algoritma u grupi.

Za svaki algoritam vezana je barem jedna populacija koju nazivamo početnom i korisnik je zadaje eksplicitno kroz aplikaciju. Na slici 5.1 nisu prikazana sva polja

zbog jednostavnosti. Neka izostavljena polja upravljaju uvjetima zaustavljanja, i internalnim stanjima cijele aplikacije.

Aplikacija je implementirana kao Internet aplikacija koristeći sljedeće tehnologije: C++, PostgreSQL, pl/pgSQL, PHP (XHP), te HTML5 na klijentskoj strani.

5.2.3. Način korištenja *optJS* aplikacije

Pri stvaranju novog zadatka korisnik aplikacije mora popuniti sljedeća polja obrasca:

1. Opis eksperimenta koji se pokreće na prirodnom jeziku
2. Implementaciju algoritma s funkcijama *main* i *evaluate*
3. Početnu populaciju ili JavaScript izvorni kôd koji je generira
4. Uvjet zaustavljanja algoritma
5. Specifikaciju svih parametara na kojima se algoritam mora pokrenuti

U specifikaciji parametara korisnik može odabrati u kojem rasponu i pomoću koje interpolacije će se parametri generirati. Primjer specifikacije parametara može biti sljedeći:

1. *seed* - sjeme pseudo-slučajnog generatora brojeva - od 0 do 4 uključivo s pomakom 1
2. *pMutation* - od 0.01 do 0.5 uključivo na 10 ravnomjernih uzoraka
3. *mPerc* - od 0.1 do 0.5 uključivo na 5 ravnomjernih uzoraka

Ovakva specifikacija parametara kreirat će ukupno $5 \cdot 10 \cdot 5 = 250$ nezavisnih algoritama. Kada se algoritmi pokrenu, jedinice posla se počinju stvarati i predaju se računarskom sloju ispod aplikacije. Budući da za svaku jedinku zapisujemo njenu dobrotu, za jedan algoritam je moguće nacrtati prosječne dobrote jedinki u populacijama kroz vrijeme, odnosno iteracijama algoritma. No, ako uz to dodamo i mogućnost gledanja grafova za razne parametre, korisnik dobiva odličan alat za empirijsko određivanje prikladnosti pojedinih parametara što je prikazano u odjeljku 5.3.

5.3. Eksperimentalno pokretanje superračunala

Za eksperimentalno pokretanje koristili smo **optJS** aplikaciju u kojoj smo testirali ponašanje genetskog evolucijskog algoritma s raznim parametrima. Algoritam koji smo implementirali rješava problem N kraljica slično kao u povezanom radu[11].

Problem N kraljica kombinatorni je problem u kojem je potrebno postaviti N kraljica na šahovsku ploču dimenzija $N \times N$ tako da se niti jedne dvije kraljice ne napadaju. U trenutku pisanja ovog rada, poznat je ukupan broj različitih načina da se konstruira ovakav razmještaj za sve $N \leq 26$ [21]. Iako se testirani algoritam pokreće za puno veće vrijednosti od N, kao rezultat nas zanima samo jedno rješenje, a ne broj različitih rješenja.

Jedinka koja se pamti u algoritmu je neka permutacija svih prirodnih brojeva manjih od N i ona predstavlja neki razmještaj kraljica na šahovskoj ploči. Ako tu permutaciju označimo sa Π, stupac kraljice u retku i dan je s Π_i. Koristeći ovakav zapis jedinke, sigurni smo da su sve kraljice u različitim redcima i stupcima. No, kraljice će se napadati i onda ako su na istoj glavnoj ili sporednoj dijagonali. Stoga, inverznu dobrotu definiramo kao broj parova kraljica koji se nalaze na istoj dijagonali i tu vrijednost kroz algoritam pokušavamo minimizirati. Ako broj takvih parova padne na nulu, znamo da smo pronašli jedinku koja predstavlja rješenje problema.

Algoritam je implementiran koristeći genetsku optimizaciju s parametrima *pMutation*, *mPerc* i *seed*. Parametar *seed* predstavlja sjeme pseudo-slučajnog generatora brojeva, a koristimo ga kao parametar jer želimo dobiti više pokretanja algoritma nad istim ostalim parametrima zbog statističke značajnosti. Parametri *pMutation* i *mPerc* predstavljaju parametre kojima se upravlja genetskim operatorom mutacije. Točni detalji optimizacijskih značajki algoritma nisu bitni za prikaz rezultata, jer ovim eksperimentom testiramo korisnost prilježećeg raspodijeljenog superračunala.

Jedno pokretanje algoritma dobivenu populaciju veličine 100 jedinki pokušava optimizirati kroz određen broj iteracija nakon čega vraća rezultantnu populaciju. Vrijeme izvođenja jedne takve instance traje između 35 i 2106 sekundi ovisno o uređajima na kojima se pokreće. Velika varijanca vremena trajanja zadatka u volonterskom računarstvu očekivana je zbog velikih razlika između različitih uređaja.

Dozvoljeni broj populacija koji smo dopustili da svaki algoritam konstruira je 200. Vektor parametara smo uzorkovali u ukupno 250 različitih vrijednosti, što znači da smo za prikupljanje svih podatka superračunalo opteretili s ukupno $200 \cdot 250 = 50000$ zadataka. Svaki zadatak prosječno zahtijeva 65 gigaFLOP procesorske snage, odnosno $6.5 \cdot 10^{10}$ decimalnih operacija. Na računalu s jednim Intel(R) Atom(TM) CPU N260 @

1.60Ghz procesorom, vrijeme izvršavanja jedne instance algoritma iznosi 217s. Pokretanjem svih 50000 zadataka, superračunalo smo opteretili s količinom računanja koja je ekvivalenta $3.28 \cdot 10^{15}$ FLOP-a, odnosno 3.28 petaFLOP-a.

Na proračunu je radilo 23 volontera s ukupno 50 procesorskih jezgara. Jedan od tih volontera predstavlja i posrednika za računanje u preglednicima opisanog u poglavlju 4. Stoga, broj procesorskih jezgara mogao je varirati tijekom pokretanja eksperimenta s obzirom na broj spojenih klijenata u preglednicima.

Sekvencijalno izvođenje algoritma na gore spomenutom procesoru trajalo bi otprilike 4 mjeseca, dok je volonterski sustav uspješno završio obradu podataka u 2 dana. Ukupan protok kroz mrežu središnjeg poslužitelja promatrajući samo serijalizirane jedinke iznosio je 10 GB.

Budući da smo uspjeli dobiti sve podatke, i da su sve komponente sustava prikazivale uredne dnevnike rada, smatramo da je pokretanje bilo uspješno.

5.4. Prikupljeni podaci

Svi rezultati nakon pokretanja prikazuju se korisniku u vidu grafova na kojima su inverzne dobrote prikazane na osi ordinate, dok je redni broj populacije prikazan na osi apscisa.

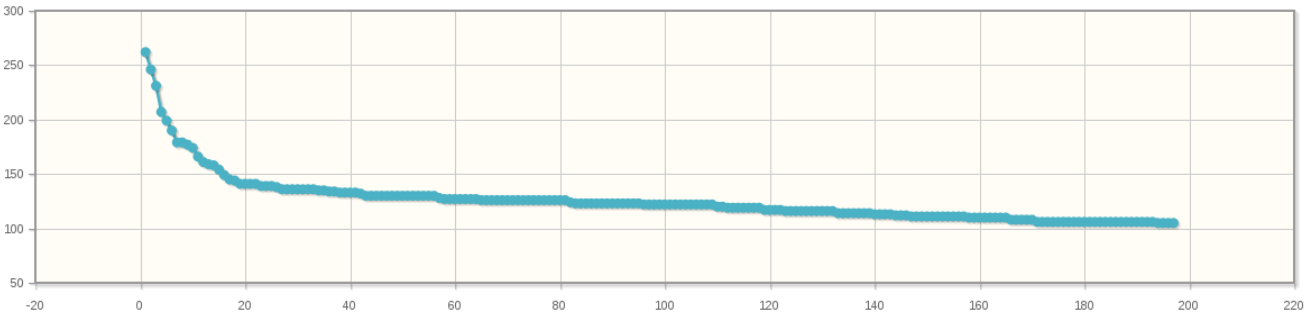
Slika 5.2 prikazuje prosječnu inverznu dobrotu po iteraciji za jednu instancu algoritma sa svim fiksiranim parametrima. Ovaj izgled krivulje dobrote tipičan je za elitističke evolucijske algoritme, no graf sam za sebe korisniku ne odaje informacije o ponašanju algoritma na različitim parametrima.

Slika 5.3 prikazuje sakupljene (engl. *aggregation*) podatke za sve pokrenute algoritme koji imaju fiksirane vrijednosti parametara $pMutation = 0.06\dot{4}$ i $mPerc = 0.3$. Takvih algoritama ima ukupno 5, a imaju različite vrijednosti *seed* parametra. Kako ne bi prikazivali sve njihove grafove sa slike 5.2, prikazane su samo minimalna, prosječna i srednja vrijednost po iteraciji. Slobodni parametar *seed* korisniku služi kao sredstvo za opetovano pokretanje algoritma i primjećivanje varijance u različitim pokretanjima.

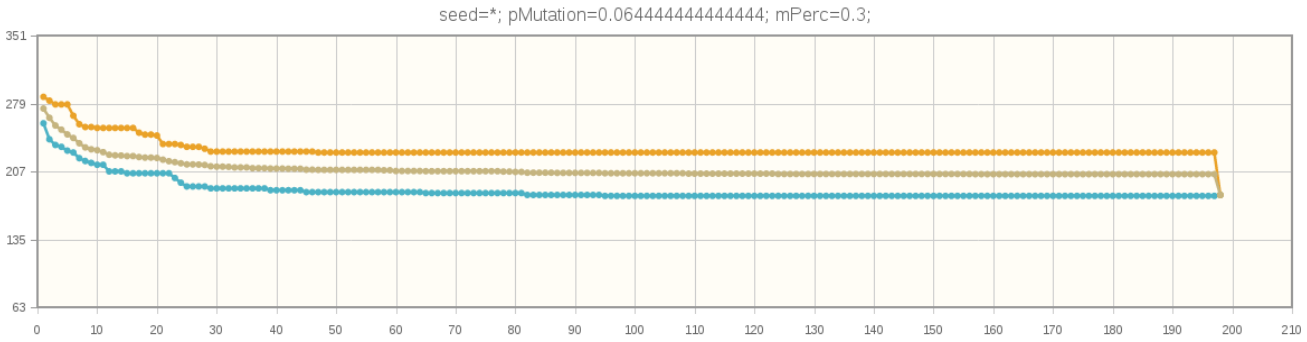
Slika 5.4 na isti način prikazuje podatke za nove vrijednosti dvaju fiksnih parametara. Vidimo da su na ovoj slici sve vrijednosti inverznih dobrota manje nego na prošloj slici po čemu zaključujemo da se algoritam bolje ponaša uz tako odabrane parametre. Slika još prikazuje i dio grafičkog sučelja koje dozvoljava korisniku da kursorom pomiče vrijednosti parametara na što će aplikacija responzivno prikazivati nove odgovarajuće grafove. Klikom na polje za oznaku (engl. *checkbox*), za odgovarajući parametar se uzimaju sve njegove vrijednosti odjednom, kao što je to napravljeno

za parametar *seed* na slikama 5.3 i 5.4.

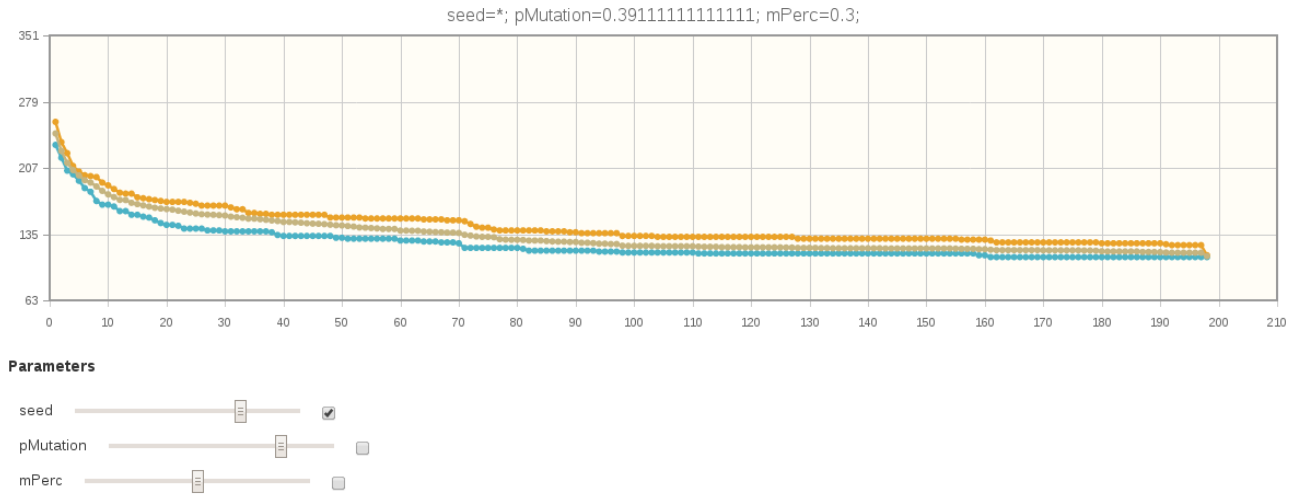
Efektivno, ovo grafičko sučelje na intuitivan i lako čitljiv način prikazuje podatke sakupljene od obrade koja je zahtijevala 4 mjeseca procesorskog vremena, no uz pomoć razvijenog raspodijeljeno superračunala trajala je značajno manje.



Slika 5.2: Očekivani izgled krivulje inverzne dobrote kroz iteracije evolucijskog algoritma



Slika 5.3: Najbolja, prosječna i najlošija pronađena jedinka po iteracijama za 5 instanci algoritma s loše odabrana dva fiksna parametra



Slika 5.4: Najbolja, prosječna i najlošija pronađena jedinka po iteracijama za 5 instanci algoritma s bolje odabrana dva fiksna parametra
Ova slika prikazuje i dio grafičkog sučelja za dinamičko pomicanje vrijednosti parametara.

6. Povezani rad

Ovo poglavlje prikazuje povezani rad u vidu rješavanja problema heterogenih platformi u raspodijeljenom računarstvu. Budući da je problem raznovrsnosti platformi jedan od najvećih izazova u volonterskom računarstvu, nije iznenađujuće da je jako puno autora istraživalo moguća rješenja. Skupni cilj svih ovih istraživanja je ponuditi neki oblik znanstvene aplikacije i mogućeg međusloja koji je može pokrenuti na što više ciljnih računala uz što veću efikasnost.

U odjeljku 6.1 prikazujemo prijašnji rad koji se koristi interpretiranim jezicima za izgradnju platformno nezavisnih aplikacija, dok u odjeljku 6.2 prikazujemo radove koji su koristili virtualizacijske tehnologije kao međusloj.

Poglavlje zaključujemo s referencom na rješenje raspodijeljenog računarstva u pregledniku *GridBee*.

6.1. Interpretirani jezici

Za znanstvene aplikacije napisane u Javi, prevedeni Java međukod može se poslati klijentima koji bi ga onda izvršili koristeći već instaliranu okolinu za izvršavanje Java programa (engl. *JRE, Java Runtime Environment*). Ovaj pristup objašnjen je u BOINC priručniku za korištenje [5, JavaApps]. Iako je u teoriji ovo rješenje potpuno prenosivo, postojeće implementacije za sada rade samo na operacijskom sustavu Windows. Kako bi sudjelovali u projektu, volonteri moraju imati ugrađenu podršku za izvršavanje Jave, *JRE*.

pyMW[19] opisuje okolinu za razvoj i pokretanje raspodijeljenih Python zadataka na različitim sustavima za mrežno računarstvo uključujući i BOINC. Kako bi pokrenuli Python program na klijentskom računalu, autori preporučuju međusloj pod imenom PyBOINC[1] koji spaja BOINC knjižnice s Python-om na sličan način kao što to radi **jsApp** opisan u poglavlju 3. PyBOINC nije dokumentiran i koliko smo uspjeli istražiti ne bori se s problemima raznovrsnih platformi, kao niti sigurnosnim problemima.

[1] https://bitbucket.org/jeremycowles/pyboinc/

González i ostali pokazali su kako je moguće pokrenuti interpretirane skripte u jezicima kao što su Java, R i Matlab na BOINC infrastrukturi[17]. Isto tako, pokazuju kako je virtualizaciju moguće iskoristiti pri rješavanju problema heterogenih platformi o čemu više govorimo u odjeljku 6.2. Njihov pristup koristi se pokretačkom skriptom (engl. *starter script*) koja provjerava postoje li sve potrebne komponente na klijentskom računalu i ako je okolina spremna, glavni program se izvršava. U slučaju da neka od komponenti ne postoji, korisnika volontera se usmjerava na upute kako preuzeti izvršnu okolinu s Interneta. Smatramo da zahtijevanje dodatne programske podrške od dobronamjernih volontera kako bi sudjelovali u znanstvenom projektu može smanjiti njihov broj.

Iako su ideje s interpretiranim aplikacijama slične kao naša ideja, u kojoj je aplikacija implementirana u JavaScript-u, nijedno prethodno rješenje ne šalje izvršnu okolinu sa zadatkom do klijenta. Zbog toga, prijašnja rješenja zahtijevaju dodatne pretpostavke o klijentovoj okolini što naravno smanjuje broj volontera koji mogu sudjelovati u projektu. U ovim pristupima sigurnosna pitanja nisu adresirana.

6.2. Virtualizacijski međuslojevi

Pod potpunom virtualizacijom smatramo pojam gdje se cijeli gostujući operacijski sustav virtualizira što uključuje njegovu jezgru i ljusku. Ovakva virtualizacija u potpunosti rješava probleme prenosivosti kao i sigurnosna pitanja. Zbog toga, nije čudno što su mnogi autori predložili upravo ovakav način rješavanja izazova u volonterskom računarstvu.

Ferreira i ostali prikazali su međusloj s imenom **libboincexec**[15] koji djeluje kao posrednik između znanstvene aplikacije i različitih hipervizora. Implementirana je VMWare i VirtualBox podrška, dok su za ostale virtualizacijske tehnologije pripremili samo sučelja koja treba popuniti. Pokazuje se kako zbog potpune virtualizacije vrijeme pokretanja virtualiziranog operacijskog sustava unosi dodatne latencije u sustav.

Marosi i ostali usporedili su postojeće BOINC međuslojeve navodeći njihove prednosti i mane, te su predstavili svoj generički međusloj (engl. *Generic BOINC Application Client - GBAC*) koji se također koristi virtualizacijom[24]. Atributi po kojima uspoređuju međuslojeve su: podržani programski jezici izgradnje, podržani sustavi za volontersko računarstvo, podržanost pokretanja već implementiranih aplikacija bez njihove promjene, podržanost pokretanja nativnih aplikacija, podržanost zapisivanja stanja (engl. *checkpoint*), izolacija od klijentskog operacijskog sustava, ograničavanje resursa aplikaciji, trud potreban za razvijanje aplikacije, heterogenost platformi te

potrebne pretpostavke o okolini na klijentu. Naše rješenje je po njihovim mjerilima za međuslojeve u rangu s ostalim razvijenim rješenjima.

Buncic i ostali izvještavaju o uloženom trudu za konstruiranje virtualnog računala koji se koristi u LHC (engl. *large hadron collider*) raspodijeljenim eksperimentima[12]. Ovaj rad prvi ukazuje na probleme veličine slike virtualnog računala koja ima vrijednosti od par stotina MB i kako ih ublažiti.

Sva prikazana rješenja koja koriste virtualizaciju zaista rješavaju sve izazove današnjeg volonterskog računarstva, no mjesta za napredak i dalje ima. Naš rad baziran je na mogućnosti da se cijela okolina za izvršavanje pošalje do volontera čak i na platformama gdje potpuna virtualizacija nije moguća. Isto tako, mrežno opterećenje i klijenata i poslužitelja uvelike je smanjeno budući da ne šaljemo cijelu sliku veličine par stotina MB već samo omotač čija veličina ne prelazi 10 MB. Tako smo uvelike povećali broj prigodnih volontera za sudjelovanje u znanstvenim projektima što smatramo korakom naprijed.

Postojeći projekt za računanje u preglednicima *GridBee* detaljno je objašnjen i uspoređen s našom infrastrukturom rješavanja istog problema u poglavlju 4. Glavni nedostatak ovog projekta je nemogućnost iskorištavanja klijenata van preglednika, odnosno nemogućnost korištenja standardnog BOINC klijentskog programa za sudjelovanje u projektu.

Uvođenjem međusloja za virtualizaciju i spajanjem s računanjem u pregledniku, velika je prednost našeg pristupa koja donosi još veći stupanj prenosivosti na različite platforme od prijašnjih radova.

7. Zaključak

Volontersko raspodijeljeno računarstvo efikasan je način da se postojeća infrastruktura računala koja uključuje laboratorijske prostorije, računala djelatnika sveučilišta, kućna računala studenata te prijenosne uređaje na punjenju preko noći iskoristi u svrhe znanstvenih proračuna. Da su takvi proračuni potrebni i da je procesorsko vrijeme znanstvenicima itekako dragocjen resurs govori i zauzetost grozda računala Isabella smještenog na Sveučilišnom računalnom centru. Budući da efikasno iskorištava infrastrukturu koja već postoji, volontersko računarstvo našlo je svoje mjesto na značajnim svjetskim sveučilištima poput sveučilišta na Berkeleyu, Stanfordu i Westminsteru.

Ovaj rad izvještava o trudu uloženom u akademskoj godini 2013./14. u ugradnju jednog takvog sustava na našem sveučilištu po uzoru na postojeća gore spomenuta rješenja. Za vrijeme pisanja ovog rada, sustav je funkcionalan i nadamo se da će u budućnosti moći biti od koristi domaćim znanstvenicima.

Kada je sustav za razašiljanje zadataka na mjestu i kada se pronađu volonteri koji će donirati svoju procesorsku snagu znanstvenicima, postavlja se pitanje maksimalne iskoristivosti međusloja za raspodijeljeno računarstvo. Korišteni međusloj BOINC (engl. *Berkeley Open Infrastructure for Network Computing*) za izgradnju superračunala opisan je u ovom radu. Tijekom implementacije ovog sustava na poslužitelju *v8boinc.fer.hr* susreli smo izazove koji dolaze s korištenjem raspodijeljenog izvršavanja zadataka.

Prvi izazov koji smo susreli raznovrsnost je arhitektura i platformi volonterskih računala, odnosno nužnost osiguravanja prenosivosti znanstvene aplikacije. Drugi izazov bio je maksimizirati broj volontera koji bi mogao sudjelovati u znanstvenom proračunu. Nadalje, željeli smo da je rezultantno raspodijeljeno superračunalo veoma lako za korištenje sa strane znanstvenika.

Ograničavanjem znanstvenih aplikacija na jezik JavaScript, uspjeli smo riješiti mnoge aktualne probleme. U radu je predstavljen **jsApp**, omotač preveden na ciljne platforme koji kao ulaz prima JavaScript aplikacijsku datoteku i ulazne podatke, a na izlazu ostavlja rezultate aplikacije pokrenute nad ulaznim podacima. Ovaj omotač ko-

risti se okolinom za izvršavanje JavaScript zadataka preuzetom iz preglednika Google Chrome i prenosiv je na veći broj platformi od ostalih međuslojeva za virtualizaciju pronađenih u literaturi. Osim toga, velika prednost **jsApp** omotača je mala veličina izvršne datoteke što je čini prigodnom za slanje do krajnjih klijenata sa zahtjevom za obradu. Činjenica da se izvršna okolina šalje sa zadatkom spušta uvjet da volonter mora imati u svoj operacijski sustav ugrađenu dodatnu programsku podršku poput virtualizacijskih hipervizora.

Ovaj rad pokazuje i kako skalabilno riješiti volontersku mogućnost sudjelovanja u računanju koristeći samo Internet preglednik. Imajući tu podršku implementiranu, naše superračunalo može dobiti i još više procesorske snage budući da za korisnika volontera sudjelovanje u projektu postaje krajnje jednostavno - posjet Internet stranici.

Snagu i izdržljivost razvijenog volonterskog superračunala testirali smo izvršavanjem paralelnih evolucijskim algoritmima u aplikaciji koja je razvijena kao dio ovog rada. Dok aplikacija **optJS** pokazuje krajnju jednostavnost i korisnost za znanstvenika koji radi na prototipu evolucijskog algoritma, superračunalo je u eksperimentalnom pokretanju pokazalo i više nego očekivano ubrzanje. U periodu od 2 dana, uspjeli smo napraviti obradu podataka koja je po količini procesorske moći ekvivalentna pokretanju $3.28 \cdot 10^{15}$ operacija s decimalnim brojevima.

U budućnosti ćemo nastaviti prenositi **jsApp** omotač na nove platforme proširujući tako količinu prikladnih volontera. Sagledat ćemo i probleme vezane za ograničenost razvoja aplikacija u jeziku JavaScript, odnosno mogućnost prijenosa i drugih programskih jezika na ovu infrastrukturu sličnim pristupom omotača. Izmijenit ćemo činjenicu da računanje u pregledniku ne čuva statistike o korisnicima što bi za volontere mogla biti velika motivacija pri sudjelovanje u projektu. Radit ćemo s domaćom akademskom zajednicom u procesu upoznavanja s volonterskim računarstvom i informiranja u mogućnostima **v8boinc** raspodijeljenog superračunala.

Iskustvo s drugih sveučilišta nam govori da je snaga volonterskog računarstva neporeciva. Kada bi svaki student Fakulteta elektrotehnike i računarstva donirao procesorsko vrijeme s uređaja koje posjeduje, algoritam konstrukcije rasporeda međuispita koji odgovara svima mogao bi se, za taj fakultet, izvršiti u jednoj noći.

LITERATURA

[1] Great internet mersenne prime search, 1996. URL http://www.mersenne.org/.

[2] Srce: Računalni klaster isabella, Studeni 2012. URL http://www.srce.unizg.hr/proizvodi-i-usluge/racunalni-resursi/racunalni-klaster-isabella/.

[3] Crowd process, 2013. URL https://crowdprocess.com/.

[4] Narasimha R Adiga, George Almási, George S Almasi, Y Aridor, Rajkishore Barik, D Beece, R Bellofatto, G Bhanot, R Bickford, M Blumrich, et al. An overview of the bluegene/l supercomputer. U *Supercomputing, ACM/IEEE 2002 Conference*, stranice 60–60. IEEE, 2002.

[5] David Anderson. Create a virtual campus supercomputing center (vcsc), 2011. URL http://boinc.berkeley.edu/trac/wiki/VirtualCampusSupercomputerCenter.

[6] David P Anderson. Public computing: Reconnecting people to science. U *Conference on Shared Knowledge and the Web*, stranice 17–19, 2003.

[7] David P Anderson. Boinc: A system for public-resource computing and storage. U *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, stranice 4–10. IEEE, 2004.

[8] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, i Dan Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[9] Kiss Balaton Gombás Kacsuk Kornafeld Kovács Marosi Vida Podhorszki. Sztaki desktop grid: a modular and scalable way of building large computing grids. U *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, stranice 1–8. IEEE, 2007.

[10] Michael Black i William Edgar. Exploring mobile devices as grid resources: Using an x86 virtual machine to run boinc on an iphone. U *Grid Computing, 2009 10th IEEE/ACM International Conference on*, stranice 9–16. IEEE, 2009.

[11] Budin Božiković, Golub. Solving n-queen problem using global parallel genetic algorithm, 2003.

[12] Predrag Buncic, Jakob Blomer, Pere Mato, Carlos Aguado Sanchez, Leandro Franco, i Steffen Klemer. Cernvm-a virtual appliance for lhc applications. U *Proceedings of the XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT08)*, 2008.

[13] Patricio Domingues, Filipe Araujo, i Luis Silva. Evaluating the performance and intrusiveness of virtual machines for desktop grid computing. U *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, stranice 1–8. IEEE, 2009.

[14] Agoston E Eiben i James E Smith. *Introduction to evolutionary computing*. springer, 2003.

[15] Diogo Ferreira, Filipe Araujo, i Patricio Domingues. libboincexec: A generic virtualization approach for the boinc middleware. U *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, stranice 1903–1908. IEEE, 2011.

[16] Brent Fulgham i Isaac Gouy. The computer language benchmarks game, 2012. *URL http://benchmarksgame. alioth. debian. org*.

[17] Daniel Lombrana González, F Fernandez de Vega, L Trujillo, G Olague, M Cárdenas, L Araujo, P Castillo, K Sharman, i A Silva. Interpreted applications within boinc infrastructure. U *Ibergrid 2008. 2nd Iberian Grid Infrastructure Conference Proceedings*, stranice 261–272, 2008.

[18] Meuer H. Top 10 supercomputer sites, Studeni 2013. URL `http://www.top500.org/lists/2013/11/`.

[19] Eric Martin Heien, Yusuke Takata, Kenichi Hagihara, i Adam Kornafeld. Pymw-a python module for desktop grid and volunteer computing. U *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, stranice 1–7. IEEE, 2009.

[20] Ian Hickson. Web workers. Candidate recommendation, W3C, Svibanj 2012. http://www.w3.org/TR/2012/CR-workers-20120501/.

[21] The OEIS Foundation Inc. Number of ways of placing n nonattacking queens on n x n board, 2013. URL `http://oeis.org/A000170`.

[22] Amy N Langville i Carl D Meyer. A reordering for the pagerank problem. *SIAM Journal on Scientific Computing*, 27(6):2112–2120, 2006.

[23] Stefan M Larson, Christopher D Snow, Michael Shirts, et al. Folding@ home and genome@ home: Using distributed computing to tackle previously intractable problems in computational biology. 2002.

[24] Attila Marosi, József Kovács, i Peter Kacsuk. Towards a volunteer cloud system. *Future Generation Computer Systems*, 29(6):1442–1451, 2013.

[25] Gary A McGilvary, Adam Barker, Ashley Lloyd, i Malcolm Atkinson. V-boinc: The virtualization of boinc. U *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, stranice 285–293. IEEE, 2013.

[26] ECMA Standard. 262: Ecmascript language specification, december 1999. *Web link: http://www.ecma-international.org/publications/standards/Ecma-262.htm*, 2011.

[27] Imre Szeberényi. Gridbee web computing framework, 2012. URL `http://webcomputing.iit.bme.hu/`.

[28] Alon Zakai. Emscripten: an llvm-to-javascript compiler. U *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, stranice 301–312. ACM, 2011.

[29] Mohammed Javeed Zaki, Wei Li, i Srinivasan Parthasarathy. Customized dynamic load balancing for a network of workstations. U *High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on*, stranice 282–291. IEEE, 1996.

[30] Jakobović Ćupic, Golub. Exam timetabling using genetic algorithm. U *Information Technology Interfaces, 2009. ITI'09. Proceedings of the ITI 2009 31st International Conference on*, stranice 357–362. IEEE, 2009.

v8boinc - Razvoj raspodijeljenog superračunala

Sažetak

Autor: Anton Grbin

Raspodijeljeno se superračunalo za razliku od uobičajenih visoko paralelnih izvedbi temelji na modelu paralelnog računarstva u kojem procesorske jedinice ne pripadaju jednoj ustanovi već procesorsku moć doniraju korisnici volonteri u trenutcima kada ne koriste svoj uređaj.

BOINC (Berkeley Open Infrastructure for Network Computing) je programska podrška ovakvom modelu raspodijeljenog računarstva originalno napravljena za projekt SETI@home koji je u zadnjih petnaestak godina obradio masivne količine podataka.

Budući da su računala koja sudjeluju u znanstvenom proračunu raznovrsna po arhitekturi i platformi, prenosivost znanstvene aplikacije veliki je izazov u njenom razvoju. Ovaj rad predstavlja nov pristup rješavanju problema prenosivosti koji se koristi JavaScript okolinom V8 preuzetom iz Internet preglednika Google Chrome.

Kako bi sudjelovao u proračunu na uobičajenoj BOINC infrastrukturi korisnik mora instalirati klijentski program, dok naš pristup dodatno podržava i mogućnost sudjelovanja koristeći preglednik, odnosno posjećivanjem Internet sjedišta projekta.

Primjenu ovog sustava prikazujemo pomoću razvijene aplikacija koja korisniku znanstveniku omogućava jednostavno pokretanje paralelnih evolucijskih algoritama koja u budućnosti može služiti i u edukacijske svrhe.

Ključne riječi: raspodijeljeno računarstvo, volontersko računarstvo, virtualizacija, prenosivost

v8boinc - Development of distributed supercomputer

Abstract

Author: Anton Grbin

Distributed supercomputer differentiates from traditional highly parallel models by having processing components scattered outside of institution making use of it's power. While scheduling is done on central server, actual computation is running on computers of user volunteers who are donating their idle CPU cycles.

BOINC (engl. *Berkeley Open Infrastructure for Network Computing*) is implementation of this computing model originally developed for SETI@home project which attracted millions of user volunteers to process high volume of signals from outer space.

Infrastructure is designed in such way that each task is an executable that must be able to run on various architectures and platforms. This poses a great portability challenge for scientific applications. Our work consists of a novel approach that is using Google Chrome's JavaScript engine V8 precompiled for each platform and dispatched together with tasks implemented in JavaScript.

To contribute their computing power traditional BOINC users must install client program, but our approach gives an additionally ability to contribute by simply visiting a web page using a browser.

Scientific usage of such system is shown with developed application that runs parallel evolutionary algorithms which can also be used in future educational purposes.

Keywords: distributed computing, volunteer computing, virtualization, portability