SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Igor Grudenić

PRILAGODLJIVO DINAMIČKO RASPOREĐIVANJE SKUPNIH POSLOVA NA GROZDU RAČUNALA

DOKTORSKA DISERTACIJA

Zagreb, 2010.

Doktorska disertacija je izrađena na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave.

Mentor: prof. dr. sc. Nikola Bogunović

Doktorska disertacija ima 156 stranica.

Disertacija br.:

Povjerenstvo za ocjenu doktorske disertacije:

1. Dr.sc. Domagoj Jakobović, docent
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

2. Dr.sc. Nikola Bogunović, redoviti profesor
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

3. Dr.sc. Dragan Gamberger, znanstveni savjetnik
   Institut Ruđer Bošković, Zagreb

Povjerenstvo za obranu doktorske disertacije:

1. Dr.sc. Domagoj Jakobović, docent
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

2. Dr.sc. Nikola Bogunović, redoviti profesor
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

3. Dr.sc. Dragan Gamberger, znanstveni savjetnik
   Institut Ruđer Bošković, Zagreb

4. Akademik dr.sc. Leo Budin, profesor emeritus (u miru)
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

5. Dr.sc. Siniša Srbljić, redoviti profesor
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

Datum obrane disertacije: 03. prosinca 2010. godine

Zahvaljujem se mentoru prof. dr. sc. Nikoli Bogunoviću za vodstvo i motivaciju tijekom izrade doktorske disertacije.

Veliko hvala prijateljima i kolegama koji su mi pomogli savjetima i dobronamjernim kritikama.

Zahvaljujem se roditeljima na bezgraničnom trudu kojeg su uložili tijekom cijelog mog školovanja.

Na kraju, posebna zahvala djevojci Marijani na razumijevanju i podršci.

SADRŽAJ

1. Uvod 1
2. Arhitekture i opterećenja paralelnih sustava 4
2.1. Arhitekture paralelnih sustava 4
2.2. Opterećenja paralelnih sustava 7
2.2.1. Klasifikacija poslova 7
2.2.2. Trajanja i dolasci poslova 9
2.2.3. Arhiva paralelnog opterećenja 10
2.2.4. Karakteristike korištenih opterećenja grozdova računala 11
3. Pregled i analiza srodnih istraživanja 14
3.1. Klasifikacija algoritama za raspoređivanje 14
3.2. Mjere učinkovitosti algoritama raspoređivanja s promjenjivim opterećenjem 15
3.3. Postupci raspoređivanja prekidivih poslova 18
3.4. Postupci raspoređivanja neprekidivih poslova 20
3.4.1. Jednostavni heuristički algoritmi za raspoređivanje neprekidivih poslova na računalnom grozdu 22
3.4.2. Raspoređivanje poslova unazadnim popunjavanjem praznina 24
3.4.3. Modifikacije postupaka unazadnog popunjavanja praznina 31
3.4.4. Složeni heuristički postupci raspoređivanja poslova 32
3.4.5. Postupak unazadnog popunjavanja praznina uz optimizaciju dinamičkim programiranjem 34
4. Sustav za simulaciju i analizu podataka računalnog grozda 38
4.1. Korisnički zahtjevi pri oblikovanju SARG sustava 38
4.2. Arhitektura SARG sustava 39
4.3. Simulator grozda računala 41
4.3.1. Pristupi simuliranju temeljenom na diskretnim događajima 41
4.3.2. Arhitektura simulatora 43
4.3.3. Jezgra simulatora 45
4.3.4. Predlošci simulacijskih entiteta za simulaciju grozda računala 49
4.4. Upravitelj simulacija 56
4.5. Komponente vezane uz prediktivnu statističku analizu 58
5. Metode za dubinsku analizu podataka o opterećenju 61

5.1. Klasifikacija naivnim Bayesovim mrežama 61
5.2. Klasifikacija stablima odlučivanja 62
5.3. Klasifikacija slučajnim šumama 63
6. Postupci raspoređivanja s predviđanjem statusa poslova 65
6.1. Definicija statusa poslova 65
6.2. Potencijalna korisnost predviđanja statusa poslova 67
6.3. Predviđanje statusa statističkim metodama za dubinsku analizu podataka 68
6.3.1. Odabir klasifikatora i optimalne veličine skupa podataka za učenje 69
6.3.2. Dinamički postupak odabira najbolje veličine skupa za učenje metodom stabilne točke 73
6.3.3. Dinamički postupak odabira najbolje veličine skupa za učenje metodom najboljeg prethodnog klasifikatora 76
6.3.4. Jednostavna heuristika na temeljena na prošlom istovjetnom poslu 78
6.3.5. Kombinacija metode najboljeg prethodnog klasifikatora i heuristike temeljene na prošlom istovjetnom poslu 80
6.4. Raspoređivanje poslova na temelju predviđanja statusa poslova 83
7. Postupci raspoređivanja s predviđanjem trajanja poslova 88
7.1. Korisničke procjene trajanja poslova 88
7.2. Postojeći postupci predikcije trajanja poslova 90
7.3. Savršeno predviđanje trajanja poslova 91
7.3.1. Utjecaj korištenja stvarnih vremena trajanja umjesto korisničkih procjena kod algoritma PNP 92
7.3.2. Utjecaj korištenja stvarnih vremena trajanja umjesto korisničkih procjena kod algoritma UPP1 93
7.3.3. Utjecaj umjetnog povećanja trajanja poslova na učinkovitost raspoređivanja algoritmom UPP1 95
7.4. Predviđanje trajanja poslova statističkim metodama za dubinsku analizu podataka 98
7.4.1. Analiza utjecaja veličine skupa za učenje na klasifikaciju trajanja poslova 99
7.4.2. Dinamički postupak klasifikacije trajanja poslova temeljen na Bayesovim mrežama 101
7.5. Utjecaj korištenja predviđenih trajanja poslova kod raspoređivanja modificiranim UPP1 postupkom 103
7.6. Raspoređivanje poslova sa prijavom stupnja dovršenosti 105
8. Zaključak 107
DODATAK A – Uporaba SARG sustava 109

DODATAK B – Programsko sučelje SARG sustava 122
POPIS LITERATURE 147
POPIS INTERNET ADRESA 153
POPIS KORIŠTENIH OZNAKA 154
POPIS KORIŠTENIH KRATICA 155

1. Uvod

Raspoređivanje je proces uparivanja zadataka i raspoloživih resursa u određenim vremenskim trenucima. U većini okruženja taj je proces statičke prirode, što označava da su svi zadaci i resursi poznati unaprijed te je izradu rasporeda potrebno napraviti jednom pri čemu se rezultati mogu primijeniti više puta. U dinamičnijim okolinama, poput planiranja proizvodnje, zadaci nisu poznati unaprijed te je odluku o raspoređivanju potrebno iznova donositi za razne skupove zadataka. Kod planiranja proizvodnje ili raspoređivanja raznih usluga olakotna okolnost koja pomaže smanjenju složenosti problema je umjerena brzina dolazaka zadataka u sustav.

U paralelnim računalnim sustavima dva temeljna izvora neizvjesnosti su vremena dolazaka poslova i obilježja poslova. Proces raspoređivanja poslova u takvim sustavima dodatno je otežan veličinom raspoloživih sustava te velikim brzinama dolazaka poslova. Zbog malog vremenskog razmaka između dolazaka poslova u sustav, raspoređivač poslova treba brzo donositi odluke u izrazito neizvjesnoj okolini. Iako je problem raspoređivanja u takvim sustavima NP kompletan, postoji više heurističkih metoda raspoređivanja koje daju učinkovite rezultate.

Temeljni problem kojim se bavi ova disertacija odnosi se na raspoređivanje poslova u paralelnim računalnim sustavima uz naglasak na raspoređivanje poslova na grozdu računala. Istraživanje je podijeljeno na četiri cjeline koje ujedno čine i doprinose disertacije. Fokus prvog dijela istraživanja odnosi se na mogućnosti simulacije ponašanja različitih aspekata ponašanja paralelnih sustava. Proučene su prednosti i nedostaci postojećih sustava za simuliranje paralelnih sustava te je definirana specifikacija sustava za simulaciju i analizu grozda računala (SARG). Prema toj specifikaciji oblikovana je raspodijeljena arhitektura SARG sustava namijenjena istovremenom izvođenju većeg broja simulacija.

U drugom dijelu istraživanja analizirani su postojeći postupci raspoređivanja poslova u paralelnim sustavima s posebnim osvrtom na postupke namijenjene raspoređivanju na grozdu računala. Eksperimentalno su uspoređene različite jednostavne heuristike raspoređivanja poslova na grozdu računala. Teoretski i praktično razmotrene su varijante najčešće korištenog postupka raspoređivanja poslova zasnovanog na unazadnom popunjavanju praznina. Predložen je izvorni postupak raspoređivanja poslova na grozdu računala temeljen na unazadnom popunjavanju praznina uz optimizaciju dinamičkim programiranjem. Uspoređene su tri inačice izvornog postupka raspoređivanja s obzirom na različit način dodjele prioriteta poslovima.

Treći i četvrti dio istraživanja fokusiran je na uklanjanje neizvjesnosti postupka raspoređivanja poslova koja nastaje zbog nepotpunog poznavanja obilježja pristiglih poslova. Proučavanje načina završetka, odnosno statusa posla, cilj je trećeg dijela istraživanja. U tom dijelu istraživanja raščlanjuju se mogući statusi poslova, razmatra se potencijalna korisnost predviđanja svih vrsta statusa te se vrši predviđanje statusa poslova različitim metodama dubinske analize podataka. Potencijalna korist predviđanja poslova koji završavaju s pogreškom analizirana je kroz modificirane postupke raspoređivanja poslova.

Trajanje poslova najveća je nepoznanica kod raspoređivanja poslova na paralelnim računalnim sustavim i tema je četvrtog dijela istraživanja. Proučeni su postojeći postupci predviđanja trajanja poslova te su navedene njihove točnosti. Potencijalna maksimalna korist uspješnog predviđanja trajanja poslova za njihovo raspoređivanje procijenjena je korištenjem stvarnih vremena izvođenja poslova kod postupka unazadnog popunjavanja praznina. Oblikovan je izvorni postupak predviđanja trajanja poslova temeljen na klasifikaciji poslova u nekoliko intervala trajanja pomoću Bayesovih mreža. Predviđanje trajanja poslova temeljeno na Bayesovim mrežama integrirano je u postupak raspoređivanja poslova unazadnim popunjavanjem praznina i izmjerena je učinkovitost tog postupka.

Disertacija je organizirana u osam poglavlja i dva dodatka. U drugom poglavlju opisane su različite arhitekture paralelnih računalnih sustava, motivacija koja stoji iza njihovog oblikovanja i programski modeli namijenjeni izradi primjenskih programa na takvim sustavima. Dostupne vrste opterećenja paralelnih sustava koje uključuju njihove statističke modele i povijesne podatke korištenja postojećih sustava također su prikazane u drugom poglavlju. Klasifikacija, teoretska i eksperimentalna usporedba postojećih postupaka raspoređivanja poslova i izvorni postupak raspoređivanja poslova temeljen na dinamičkom programiranju i unazadnom popunjavanju praznina opisani su u trećem poglavlju. Arhitektura i neki aspekti ostvarenja sustava za simulaciju grozda računala, kao i opis postojećih simulacijskih sustava slične namjene, prikazani su u četvrtom poglavlju.

Statističke metode dubinske analize podataka koje uključuju Bayesove mreže, stabla odlučivanja i slučajne šume, te su korištene za oblikovanje postupaka predviđanja statusa i trajanja poslova, opisane su u petom poglavlju. Različiti načini završetka poslova, potencijalni značaj njihovog predviđanja, izvorni postupak predviđanja neispravnih poslova i njegova učinkovitost dani su u šestom poglavlju. Postupci automatske procjene trajanja poslova zajedno s izvornim postupkom klasifikacije trajanja posla u različite diskretne intervale prikazani su u sedmom poglavlju. U istom poglavlju eksperimentalno je utvrđen značaj izvornog postupka klasifikacije trajanja poslova na postupak raspoređivanja temeljen na unazadnom popunjavanju praznina.

Primjer razvoja nove metode raspoređivanja poslova SARG sustavom, te mjerenje njene učinkovitosti opisani su u dodatku A. Dokumentacija programskog sučelja osnovnih dijelova SARG sustava dana je u dodatku B.

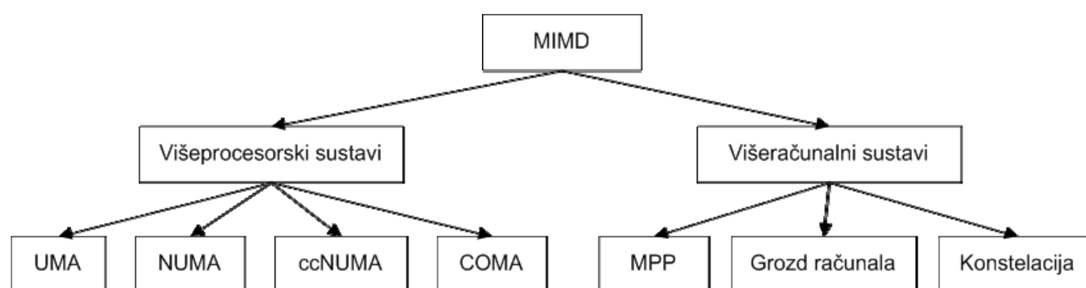
2. Arhitekture i opterećenja paralelnih sustava

Arhitekture i opterećenja paralelnih sustava jedina su dva parametra temeljem kojih se oblikuju postupci raspoređivanja poslova. Vrsta poslova u sustavu uglavnom je određena arhitekturom samog sustava, pa time arhitektura sustava određuje i klasu raspoređivača poslova. Detaljnije oblikovanje raspoređivača unutar klase ovisi o vrsti opterećenja za koju je potrebno ostvariti što veću učinkovitost. Arhitekture i opterećenja paralelnih sustava opisani su u odjeljcima 2.1 i 2.2.

2.1. Arhitekture paralelnih sustava

Arhitekture sustava se mogu klasificirati na nekoliko načina ovisno o promatranom aspektu arhitekture. Flynnovom klasifikacijom [1] paralelni sustavi se dijele s obzirom na broj istovremenih tokova instrukcija i podataka na SISD, SIMD, MISD i MIMD sustave. SISD je sekvencijalno računalo bez paralelizma u podatkovnom ili instrukcijskom toku. SIMD je također sekvencijsko računalo s jednim nizom instrukcija, ali s obradom više tokova podataka. Tipični primjerci SIMD arhitekture su vektorska računala i grafički procesori. MISD arhitektura predstavlja paralelno računalo kod kojeg više instrukcijskih nizova obrađuje jedan tok podataka. Iako za MISD arhitekturu ne postoji praktična primjena ona se radi potpunosti navodi u Flynnovoj klasifikaciji. MIMD arhitektura se sastoji od više instrukcijskih tokova koji obrađuju različite tokove podataka te je najčešće korištena arhitektura paralelnih sustava. Zahvaljujući raznolikosti MIMD sustava moguće je napraviti njihovu detaljniju podjelu kako je naznačeno na slici 2.1.

Vršna podjela MIMD arhitekture zasniva se na načinu prezentacije memorijskog prostora procesorima u sustavu. Kod višeprocesorskih sustava (*engl. multiprocessor systems*) pristup memoriji organiziran je preko jedinstvenog adresnog prostora, pri čemu je interna organizacija memorije proizvoljna. Kod višeračunalnih sustava (*engl. multicomputer*



Slika 2.1 Klasifikacija MIMD paralelnih sustava

systems) svaki procesor može adresirati vlastiti memorijski prostor, a s ostalim procesorima se komunikacija vrši izmjenom poruka. Višeprocesorski sustavi se s obzirom na vrstu pristupa memoriji dijele na UMA, NUMA, ccNUMA i COMA sustave. U UMA (*engl. uniform memory access*) sustavima svi procesori koji pristupaju memoriji to čine na uniforman način, odnosno trajanje pristupa memoriji je jednako za sve procesore. Kod NUMA (*engl. non-uniform memory access*) sustava brzina pristupa određenoj memorijskoj lokaciji nije jednaka za sve procesore u sustavu. Da bi se donekle umanjilo smanjenje učinkovitosti paralelnih primjenskih programa uzrokovano različitim brzinama pristupa memoriji svakom procesoru se dodjeljuje određena količina priručne memorije. Održavanje koherentnosti podataka u takvim sustavima može se ostvariti programski ili sklopovski. Sustavi kod kojih je sklopovski ostvarena koherentnost priručnih memorija nazivaju se ccNUMA sustavi (*engl. cache coherent NUMA*). U COMA (*engl. cache only memory architecture)* sustavima lokalna memorija svakog procesora smatra se njegovom priručnom memorijom. Kod pristupa podacima koji nisu u lokalnoj memoriji, sustav vrši preseljenje podataka u lokalnu memoriju odgovarajućeg procesora.

Višeračunalni sustavi se s obzirom na vrstu veze između procesora dijele na MPP sustave, grozdove računala i konstelacije. MPP (*engl. massively parallel processor*) sustavi sastoje se od procesora s lokalnim memorijama koji su međusobno povezani brzim komunikacijskim vezama. Grozdovi računala se razlikuju od MPP sustava samo u brzini interkonekcije, pri čemu se kod povezivanja procesora na grozdu računala često koriste neke od uobičajenih mrežnih tehnologija. Razlika u brzini komunikacije je značajna sa stanovišta dostupnih programskih modela za MPP sustave i računalne grozdove. Kod MPP sustava su podržani programski modeli temeljeni na dijeljenoj memoriji i izmjeni poruka, dok je kod grozdova računala zbog sporije veze među procesorima moguće učinkovito koristiti samo programski model temeljen na izmjeni poruka.

Konstelacije su grozdovi računala sastavljeni od povezanih MPP sustava kod kojih je broj procesora u jednom MPP sustavu veći od ukupnog broja MPP sustava.

Sa tehnološke strane paralelne sustave je moguće podijeliti na temelju topologije interkonekcijske mreže. Najčešće korištene topologije paralelnih sustava uključuju dijeljenu sabirnicu, stablo, prsten, susjedno povezane procesore (*engl. mesh topology*), hiperkocku i potpuno povezane sustave. Različite vrste MIMD sustava mogu biti povezane s više različitih topologija.

MIMD arhitekture se međusobno razlikuju prema složenosti arhitekture i jednostavnosti programskog modela. Višeprocesorski sustavi, s izuzetkom UMA arhitekture, imaju najveći stupanj složenosti, pri čemu je učinkovito podržan jednostavan programski model temeljen

na dijeljenoj memoriji. Isti programski model podržan je i kod višeprocesorskih MPP sustava, ali sa smanjenom učinkovitošću izvedbe. Grozdovi računala podržavaju samo složeniji programski model temeljen na izmjeni poruka, ali su popularni zbog jednostavne arhitekture koja dovodi do veće dostupnosti takvih sustava. Razvoj programske potpore namijenjene tom modelu moguće je olakšati primjenom različitih formalnih postupaka oblikovanja i verifikacije sustava [2][3].

Među 500 trenutno najsnažnijih računala na svijetu [73] nalazi se 80% grozdova računala, dok ostalih 20% čine MPP sustavi, što povećava naglasak na nužnosti učinkovitog raspoređivanja poslova na grozdu računala. Najpopularniji načini povezivanja grozda računala su gigabitni ethernet [74] u 48,4% slučajeva i Infiniband [4] u 41,4% slučajeva.

2.2. Opterećenja paralelnih sustava

Analiza opterećenja paralelnih sustava [5] čini važan aspekt procesa oblikovanja i prilagođavanja postupaka raspoređivanja poslova. Značajke opterećenja različitih sustava variraju ovisno o namjeni sustava te profilu njegovih korisnika. Različite klase poslova o kojima ovisi odabir postupka raspoređivanja opisane su u odjeljku 2.2.1. Kod simuliranja postupaka raspoređivanja moguće je modelirati opterećenje statističkim metodama ili koristiti opterećenja dobivena promatranjem postojećih sustava. Postupci modeliranja trajanja i vremena dolazaka poslova dani su u odjeljku 2.2.2. Javno dostupna opterećenja dobivena promatranjem računalnih grozdova i temeljne značajke poslova u tim sustavima prikazane su odjeljku 2.2.3. Podskup javno dostupnih opterećenja koji je korišten za evaluaciju postupaka raspoređivanja poslova opisan je u odjeljku 2.2.4.

2.2.1. Klasifikacija poslova

Poslovi u računalnim grozdovima se klasificiraju s obzirom na stupanj paralelnosti, mogućnost prekidanja i preseljenja te s obzirom na interaktivnost [6].

Stupanj paralelnosti posla određen je vrstom zahtjeva na količinu potrebnih procesora. Kod fiksiranih poslova (*engl. rigid*) korisnici unaprijed zatraže određenu količinu računalnih resursa i ta količina se ne mijenja u toku izvođenja posla. Iako je većina paralelnih programa pisana na način da može raditi na različitom broju procesora, korisnici obično traže točno definiranu količinu procesora budući da je većina raspoređivača poslova oblikovana za prihvat fiksiranih poslova. Kod nekih paralelnih programa stupanj paralelnosti ovisi o veličini problema, jer se sve veličine problema ne mogu učinkovito paralelizirati na raznim skupovima procesora.

Poslovi kod kojih su inicijalni zahtjevi za resursima poznati unaprijed, a potrebna količina resursa se može mijenjati kroz vrijeme se nazivaju evoluirajući (*engl. evolving*) poslovi. Evoluirajući poslovi često uključuju izračune koji se sastoje od nekoliko faza, pri čemu za sve faze nije potreban jednak broj resursa, pa se nepotrebni resursi mogu dodijeliti drugim poslovima. Programske biblioteke PVM [7] i MPI 2.0 [75] za oblikovanje paralelnih poslova temeljenih na izmjeni poruka podržavaju implementaciju evoluirajućih poslova.

Modelirajući (*engl. moldable*) poslovi se šalju u sustav s rasponom zadovoljavajućih brojeva procesora potrebnih za izvođenje posla. Nakon dobivanja određene količine procesora, zahtjevi ostaju nepromijenjeni do kraja izvođenja posla. Raspoređivač poslova koji raspoređuje modelirajuće poslove ima veću fleksibilnost kod odlučivanja čime se može postići veća učinkovitost raspoređivanja poslova. Dodatni parametar koji može pomoći u

optimizaciji je promjena ubrzanja posla s obzirom na povećanje broja procesora. Budući da je ta promjena ubrzanja linearna u rijetkim slučajevima, optimalno izvođenje nekog posla uključivalo bi traženje koljena funkcije brzine izračunavanja s obzirom na broj procesora.

Manipulacija promjenjivim poslovima (*engl. malleable*) daje najveću slobodu odlučivanja raspoređivaču poslova, budući da se takvim poslovima može inicijalno dodijeliti proizvoljan broj procesora te se u toku izvođenja procesori mogu dodavati i oduzimati ovisno o ostalim potrebama sustava. Najčešća arhitektura promjenjivih poslova uključuje više jednakih procesnih elemenata koji rješavaju podskup skupa nezavisnih zadataka, pri čemu isključivanje i naknadno uključivanje procesnih elemenata ne umanjuje učinkovitost ostalih dijelova posla.

Poslovi u raspodijeljenim sustavima se s obzirom na vrstu prekidanja i ponovnog nastavljanja posla sa stanovišta raspoređivača poslova dijele na neprekidive, prekidive i mobilne poslove. Neprekidivi poslovi (*engl. non-preemptive*) su poslovi čije izvršavanje raspoređivač poslova ne smije prekinuti i kasnije nastaviti s njihovim izvođenjem. Prekidive poslove (*engl. preemptive*) raspoređivač poslova smije prekinuti u toku izvođenja te uzastopnim prekidanjem i izmjenom poslova koji se izvršavaju može stvoriti privid istovremenog izvođenja više poslova na istom skupu resursa. Mobilni poslovi posebna su vrsta prekidivih poslova kod kojih je, uz prekidanje, omogućeno preseljenje s jednog skupa resursa na drugi skup resursa. Preseljenje paralelnih poslova je tehnički zahtjevan problem budući da je uz preseljenje instrukcijskog i podatkovnog segmenta svih procesa unutar jednog posla, potrebno preseliti i sve otvorene komunikacijske veze među poslovima zajedno s njihovim međuspremnicima (*engl. buffer*).

Mobilnost, a djelomično i prekidivost poslova omogućeni su podrškom za spremanje stanja posla (*engl. checkpointing*). Sustavi u kojima je omogućeno spremanje stanja posla otporniji su na povremene kvarove i nedostupnost resursa, a spremanje stanja posla preduvjet je za ostvarivanje njihove mobilnosti. Spremanje stanja sekvencijalnih poslova može se ostvariti na generički sistemski oblikovan način ili može biti podržano od strane programera koji je oblikovao posao [8]. Sistemski podržano spremanje stanja poslova je preferirani općenitiji pristup problemu koji mora uključiti sve aspekte nekog procesa poput stanja memorije i trenutno otvorenih opisnika datoteka. Zbog kompleksnosti oblikovanja postupka spremanja stanja poslova, uglavnom se koristi sistemski podržano spremanje stanja. Na Unix [76] zasnovanim sustavima podržano je spremanje stanja za velik broj aspekata procesa sekvencijskih poslova [9][10]. Spremanje stanja paralelnih poslova nije omogućeno za postojeće programske biblioteke namijenjene razvoju paralelnih poslova, ali je predložen okvir za nadogradnju MPI sustava [11] sa ciljem omogućavanja tog svojstva.

Sa korisničkog stanovišta pristupu poslovima koji se izvode poslovi se dijele na interaktivne (*engl. interactive*) i skupne (*engl. batch*) poslove. Raspoređivač poslova na temelju vrste posla treba odrediti strategiju raspoređivanja i razinu usluge koju će dodijeliti poslu.

2.2.2. Trajanja i dolasci poslova

Vremena dolazaka poslova u sustav i njihova trajanja temeljne su značajke poslova koje su važne za proces raspoređivanja. Poissonov proces [12] je stohastički model koji opisuje sustave u kojima su događaji međusobno nezavisni i kontinuiranom frekvencijom se pojavljuju kroz vrijeme. Eksponencijalnom distribucijom opisana je vjerojatnost pojavljivanja određene duljine vremenskog intervala između dolaska dva posla u Poissonovom procesu.

Iako se aproksimacija Poissonovim procesom ponekad koristi za modeliranje vremena dolazaka poslova, poslovi u grozdu računala nisu nužno međusobno nezavisni, a i brzina dolazaka poslova u sustav nije konstantna. Dolasci poslova u sustav se tipično dešavaju u toku radnog vremena, dok rezultat prošlih poslova u sustavu može potaknuti korisnika na stvaranje novih poslova. Kod većine raspoređivača poslova ne koristi se statistička procjena vjerojatnosti dolazaka poslova u sustav, a pri simuliranju sustava uglavnom se koriste podaci prikupljeni s različitih grozdova računala.

Vrijeme izvođenja je najvažnija značajka poslova potrebna u procesu raspoređivanja poslova. U procesu simuliranja grozda računala podaci o trajanjima poslova mogu se dobiti na temelju prošlosti postojećih računalnih grozdova ili na temelju statističkih modela. Statističko modeliranje vremena izvođenja se uglavnom provodi traženjem optimalne distribucije za prikaz nekog stvarnog opterećenja grozda računala te podešavanjem parametara te distribucije.

Da bi se dobio što vjerniji model trajanja poslova u sustavu korištene su hiper-eksponencijalna [13], hiper-erlangova [14], log-uniformna [15] i hiper-gama distribucija [16]. Kod modeliranja trajanja poslova hiper-erlangovom distribucijom [14] poslovi su podijeljeni u skupine ovisno o količini traženih resursa te je svaka skupina modelirana zasebno. Veća općenitost temeljna je prednost korištenja modela umjesto stvarnih opterećenja grozda računala. Generatori opterećenja zasnovani na statističkim modelima stabilniji su od opterećenja zabilježenih na grozdovima računala zato jer generirani nizovi poslova ranije postižu postojano srednje vrijeme izvođenja [17].

Kod simulacije procesa raspoređivanja modelirajućih i promjenjivih poslova, očekivano je da trajanje posla ovisi o količini dodijeljenih resursa. Da bi raspoređivač poslova donio

procjenu trajanja posla za različite količine procesora najčešće se koristi linearni model ili downeyev model [15] ubrzanja poslova.

Statističko modeliranje vremena izvođenja ne koristi se u samom procesu raspoređivača poslova, a različite metode predviđanja trajanja poslova čijim korištenjem se povećava učinkovitost raspoređivanja poslova opisane su u poglavlju 7. U izradi ovog rada korištena su opterećenja dobivena s postojećih grozdova računala, a ne statistički modeli, budući da takav pristup olakšava usporedbu s drugim postupcima raspoređivanja i precizan je odraz stvarnih sustava.

2.2.3. Arhiva paralelnog opterećenja

Simulacija postupka raspoređivanja poslova ovisi o vrsti opterećenja sustava. Dolasci i trajanja poslova mogu se simulirati pomoću statističkog modela ili se može upotrijebiti sačuvana prošlost korištenja nekog od postojećih grozdova računala.

Arhiva paralelnog opterećenja (*engl. Parallel Workloads Archive*) [77] sadrži opterećenja 25 računalnih grozdova nastala kroz period od 14 godina. Opterećenja u arhivi očišćena su od povremenih nakupljenih gomila poslova jednog korisnika generiranih unutar kratkog vremena da bi se simulacijom grozda računala pratilo stabilno ponašanje sustava. Za svaki posao u sustavu definirane su značajke prikazane u tablici 2.1.

Sve značajke poslova propisane formatom arhive paralelnog opterećenja se ne nalaze u opterećenjima za svih 25 računalnih grozdova. Procjena trajanja poslova dostupna je za 16 grozdova računala, a zahtjevi za određenom količinom memorije dostupni su za 11 grozdova računala. Identifikatori grupa korisnika također nisu dostupni za sve računalne grozdove u arhivi paralelnog opterećenja. Red poslova i identifikator particije sustava označavaju dijelove sustava koji su rezervirani za provođenje određenih vrsta poslova, a za grozdove

Tablica 2.1 Značajke poslova navedene u arhivi paralelnog opterećenja

Identifikator posla	Zatražena količina radne memorije
Vrijeme dolaska posla	Status završetka posla
Vrijeme čekanja posla na izvođenje	Identifikator korisnika
Trajanje posla	Identifikator grupe korisnika
Broj iskorištenih procesora	Identifikator primjenskog programa
Prosječno vrijeme izvođenja po procesoru	Red poslova
Iskorištena količina radne memorije	Identifikator particije sustava
Zatraženi broj procesora	Identifikator prethodnog posla
Procijenjeno vrijeme izvođenja	Vrijeme razmišljanja proteklo od prethodnog posla

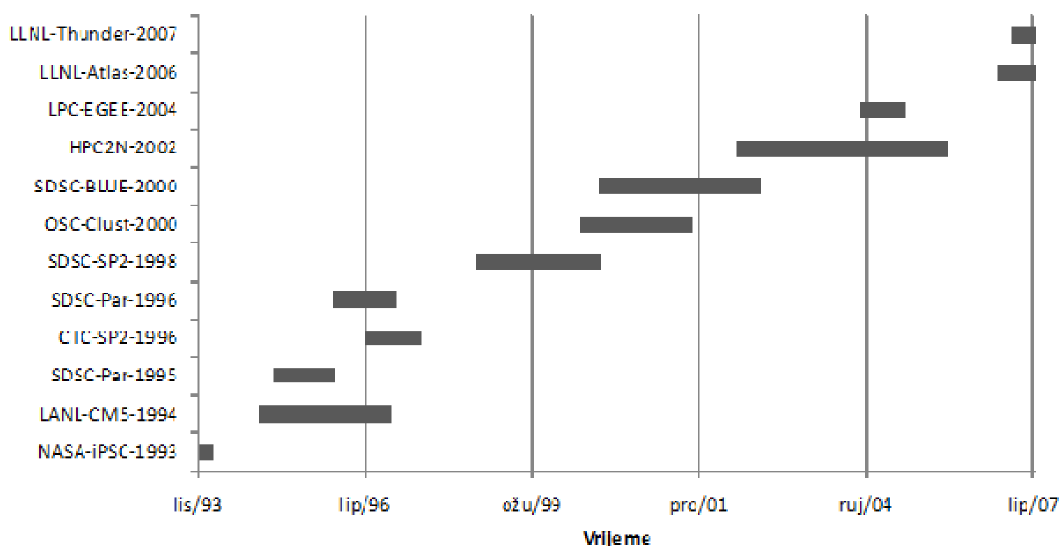
kod kojih se koristi više redova ili particija dan je i njihov detaljan opis. Identifikator prethodnog posla označava posao koji je preduvjet za izvođenje promatranog posla. Vrijeme razmišljanja proteklo od prethodnog posla označava vrijeme proteklo od završetka posla koji je preduvjet za izvođenje promatranog posla do trenutka dolaska promatranog posla u sustav.

Kod korištenja statističkih metoda za dubinsku analizu podataka sa ciljem predviđanja statusa i trajanja poslova dio podataka o poslovima koji nije relevantan je zanemaren. Bez obzira na cilj predviđanja izbačeni su svi podaci koji nisu poznati u trenutku dolaska posla. Podaci koji nisu poznati u trenutku dolaska posla označeni su sivom bojom u tablici 2.1. Identifikator posla je također isključen iz statističkih metoda za dubinsku analizu podataka budući da informacija koju nosi odgovara vremenu dolaska posla, a korištenje zavisnih značajki poslova ne pogoduje nekim metodama klasifikacije poslova.

2.2.4. Karakteristike korištenih opterećenja grozdova računala

Arhiva paralelnog opterećenja sadrži podatke o poslovima 25 računalnih grozdova, pri čemu je podskup podataka korišten u izradi ovog rada. Podaci o opterećenjima 12 računalnih grozdova iz arhive paralelnog opterećenja analizirani su s obzirom na statuse poslova i vrijeme izvođenja te je na njima mjerena učinkovitosti raznih postupaka raspoređivanja poslova. Odabirom podskupa dostupnih opterećenja smanjen je broj potrebnih mjerenja i pojednostavljen je prikaz rezultata.

Popis 12 računalnih grozdova koji su korišteni u eksperimentima, zajedno s vremenskim periodom u kojem je određeni računalni grozd bio u funkciji prikazan je na slici 2.2. Sa slike



Slika 2.2 Periodi korištenja grozdova računala

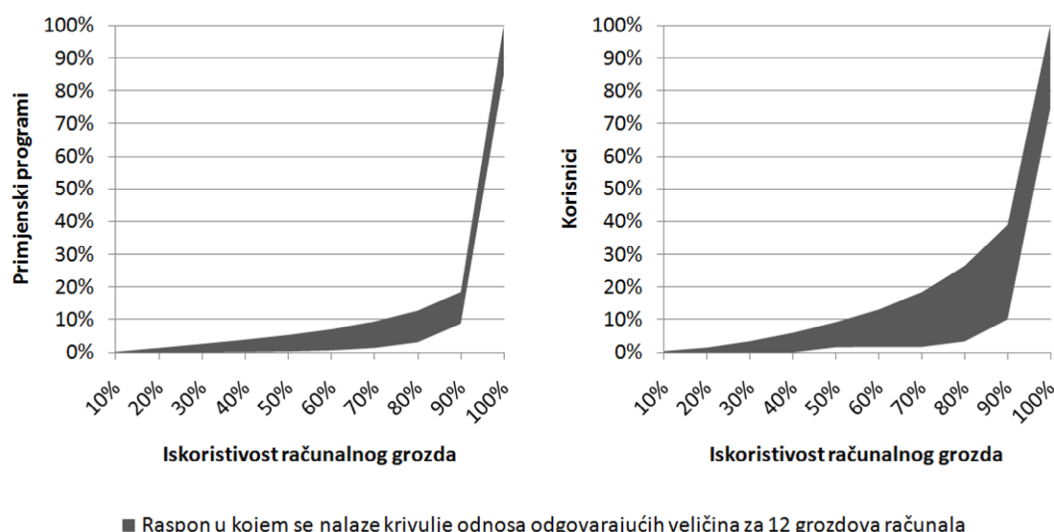
je vidljivo da su računalni grozdovi korišteni kroz period od 14 godina, pri čemu se periodi korištenja različitih računalnih grozdova kreću od tri mjeseca za računalni grozd *NASA-IPSC-1993* do tri godine za računalni grozd *HPC2N-2002*.

Na temelju analize prikazanih računalnih grozdova ustanovljeno je da broj korisnika i različitih primjenskih programa ne ovisi o količini resursa dostupnoj na grozdu računala. Srednja vremena trajanja poslova također ne ovise o veličini računalnog grozda, broju korisnika ili broju različitih primjenskih programa u sustavu.

Iako srednje vrijeme trajanja poslova ne daje informaciju o vrsti računalnog grozda, većinu raspoloživog vremena na računalnim grozdovima koristi vrlo mali broj različitih primjenskih programa. Na slici 2.3 prikazan je odnos broja različitih primjenskih programa i korisnika s obzirom na potrošnju resursa na računalnom grozdu. Na većini računalnih grozdova 70% računalnih resursa iskorištava 4% različitih primjenskih programa, odnosno prosječno 7,5% od ukupnog broja korisnika na računalnom grozdu. Većinu od 90% raspoloživih računalnih resursa koristi manje od 20% primjenskih programa te manje od 25% korisnika, uz izuzetak računalnog grozda *SDSC-BLUE-2000*, gdje 38% korisnika koristi 90% resursa. Računalni grozdovi koriste podatke o potrošnji resursa svakog korisnika pri čemu se podešavaju prioriteti izvođenja poslova.

Kod primjenskih programa koji iskorištavaju 70% resursa sustava, srednje vrijeme trajanja je prosječno 2,11 puta veće od srednjeg vremena trajanja svih primjenskih programa u sustavu. Ta vrijednost se smanjuje na 1,67 ukoliko se promatraju programi koji zauzimaju 90% resursa na 12 računalnih grozdova.

Kod promatranja stupnja paralelnosti primjenskih programa očekivano je da programi



Slika 2.3 Iskoristivost računalnih grozdova s obzirom na primjenske programe i korisnike

duljeg trajanja imaju značajno viši stupanj paralelnosti od ostalih programa, budući da bi se njihovom paralelizacijom dobila najveća ušteda na vremenu. Analizom 12 računalnih grozdova ustanovljeno je da je na računalnom grozdu *LLNL-Atlas-2006* prosječan broj korištenih procesora za primjenske programe koji koriste 70% resursa 40% manji od srednjeg broja korištenih resursa na cijelom sustavu. Za ostale računalne grozdove izmjereno je povećanje stupnja paralelnosti za primjenske programe koji koriste 70% resursa i ono iznosi do maksimalnih 30% u odnosu na stupanj paralelnosti izmjeren u cijelom sustavu.

Povećanje stupnja paralelnosti primjenskih programa ne mora nužno dovesti do povećanja učinkovitosti cijelog sustava, budući da je teško paralelizirati programe uz održavanje linearnog ubrzanja. Najveći problemi paralelizacije programa na grozdu računala vezani su odabir strategije balansiranja opterećenja među procesorima [18][19], te ostvarenja raspodijeljene priručne memorije da bi se optimizirao dohvat podataka u sustavu [20].

Povećanje broja paralelnih primjenskih programa uzrokuje viši stupanj fragmentacije rasporeda izvođenja poslova.

3. Pregled i analiza srodnih istraživanja

Raspoređivanje poslova u paralelnim sustavima je NP težak problem [21] te s obzirom na dinamičnost okoline traženje optimalnog rasporeda nije moguće. Postupci raspoređivanja poslova se klasificiraju s obzirom na karakteristike sustava, poslova i na ciljanu optimizacijsku metriku. Klasifikacija algoritama raspoređivanja prikazana je u odjeljku 3.1. Različite metrike prema kojima se mjeri učinkovitost postupaka raspoređivanja poslova opisane su u odjeljku 3.2. Budući da se za većinu paralelnih sustava koriste slične mjere učinkovitosti, a paralelni sustavi su uglavnom homogeni, postupci raspoređivanja poslova u paralelnim sustavima se najčešće dijele s obzirom na omogućavanje prekidivosti posla. Postupci kod kojih je omogućeno uzastopno prekidanje poslova te funkcioniraju na principu dijeljenja vremena poslovima (*engl. time sharing*) opisani su u odjeljku 3.3. Postupci kod kojih nije dozvoljeno prekidanje poslova (*engl. space sharing*) [22] prikazani su u odjeljku 3.4, gdje je opisan i izvorni postupak raspoređivanja poslova.

3.1. Klasifikacija algoritama za raspoređivanje

Problemi raspoređivanja mogu se klasificirati s obzirom na prirodu resursa koji se koriste za raspoređivanje poslova, na karakteristike poslova koji je potrebno odraditi na tim resursima i na ciljanu optimizacijsku metriku. Zbog trostrukog određenja postupaka za raspoređivanje poslova, standardna notacija za problem raspoređivanja je uređena trojka $\alpha \mid \beta \mid \gamma$ koja opisuje karakteristike resursa (α), poslova (β) i ciljanu metriku (γ).

Algoritmi za raspoređivanje poslova se dijele na algoritme s unaprijed poznatim opterećenjem (*engl. offline scheduling algorithms*) i algoritme s promjenjivim opterećenjem (*engl. online scheduling algorithms*) [23]. Kod algoritama s unaprijed poznatim opterećenjem, svi poslovi su definirani prije pokretanja algoritma i postupkom raspoređivanja poslova se donosi odluka o vremenskim trenucima i resursima na kojima će se ti poslovi odraditi.

U algoritmima s promjenjivim opterećenjem, raspoređivač poslova zaprima poslove koji dolaze u raznim vremenskim trenucima i donosi odluke bez poznavanja budućih poslova. Zbog neizvjesnosti koja je uvedena nepoznatom budućnošću nije moguće garantirati optimalne rasporede za proizvoljne skupove poslova.

Tradicionalno se vrsta algoritma određuje kao parametar poslova pri čemu se definira da je posao nepoznat dok nije poznato vrijeme njegova nastanka što se u $\alpha \mid \beta \mid \gamma$ notaciji bilježi parametrom o_i koji označava da je za posao p_i definirano vrijeme otpuštanja o_i prije kojeg ne može doći do njegovog izvršavanja.

U nekim sustavima je vrijeme izvršavanja poznato u trenutku nastanka posla, dok u nekim slučajevima trajanje posla nije poznato sve do njegovog završetka što se bilježi oznakama *promjenjivo_nt* i *promjenjivo_pt*. Oznaka *promjenjivo* odnosi se na promjenjivo opterećenje sustava koje nije poznato na samome početku, a sufiksi *pt* i *nt* na poznato i nepoznato trajanje. Dodatno poslovi mogu biti prekidivi (*engl. preemptive*) i neprekidivi što proširuje klasifikaciju algoritama za raspoređivanje te se uvodi oznaka posla $\beta = prekidivi$ za sustave s prekidivim poslovima.

Postupak raspoređivanja poslova kod web poslužitelja koji poslužuje statičke web stranice definiran je kao $1 \mid o_i, promjenjivo_pt, prekidivo \mid SrednjiOdziv$ gdje jedinica označava izvršavanje na jednom procesoru, pri čemu je posao dostupan tek u vremenu otpuštanja o_i te je posao prekidiv, a samo opterećenje je promjenjivo uz poznata trajanja poslova. Predvidljivost trajanja posla u uskoj je vezi s veličinom statične web stranice, a srednji odziv sustava je mjera prema kojoj se optimira postupak raspoređivanja.

Raspoređivanje poslova u paralelnim sustavima je skupina različitih optimizacijskih problema, ali u ovom poglavlju će biti dan pregled postupaka raspoređivanja na pretežno homogenim računalnim sustavima (P_m), s prekidivim i neprekidivim poslovima pri čemu su poslovi dostupni za izvođenje tek u trenutku njihovog dolaska u sustav, a trajanja poslova su nepoznata do samog završetka posla. Različite mjere učinkovitosti za takve sustave opisane su u odjeljku 3.2.

Kod sustava $P_m \mid o_i, promjenjivo_nt \mid f$ sa neprekidivim poslovima i nepoznatim vremenima trajanja nije mogući naći strategiju koja će za sve nizove poslova na ulazu dati raspored blizak optimalnom.

3.2. Mjere učinkovitosti algoritama raspoređivanja s promjenjivim opterećenjem

Mjere učinkovitosti algoritama su jedini način usporedbe dva postupka raspoređivanja poslova. Različiti postupci raspoređivanja oblikovani su s drugačijim ciljanim metrikama, pri čemu odabir metrike za raspoređivanje ovisi o vrsti problema.

Usporedba algoritama za raspoređivanje poslova može se provesti teoretskim razmatranjem i eksperimentalno. Kod teoretskih razmatranja algoritama, ciljani postupak raspoređivanja etalonski se uspoređuje s optimalnim postupkom raspoređivanja.

Eksperimentalno uspoređivanje algoritama se provodi tako da se mjera uspješnosti $f(A,I)$ za algoritam raspoređivanja A, izračuna za ograničeni skup ulaznih nizova poslova I. Rezultat takvog mjerenja uspoređuje se s mjerenjima etalonskog algoritma $f(A_E,I)$. Kao etalonski algoritam A_E zbog složenosti vrlo često nije moguće odabrati optimalan algoritam

$P - skup\ svih\ poslova$

$T(p) - trajanje\ posla\ p \in P$

$C(p) - vrijeme\ od\ dolaska\ posla\ p \in P\ u\ sustav\ do\ početka\ njegova\ izvođenja$

$$srednje\ vrijeme\ odziva\ sustava = \frac{\sum_{p \in P} T(p) + C(p)}{|P|}$$

$$srednje\ vrijeme\ čekanja = \frac{\sum_{p \in P} C(p)}{|P|}$$

$$srednje\ usporenje = \frac{\sum_{p \in P} \frac{C(p) + T(p)}{T(p)}}{|P|}$$

$$srednje\ ograničeno\ usporenje = \frac{\sum_{p \in P} Max(\frac{C(p) + T(p)}{Max(T(p), \tau)}, 1)}{|P|}$$

$$srednje\ c - normalizirano\ ograničeno\ usporenje = \frac{\sum_{p \in P} Max(\frac{C(p) + T(p)}{c \times Max(T(p), \tau)}, 1)}{|P|}$$

Slika 3.1 Mjere učinkovitosti raspoređivanja poslova

raspoređivanja, pa se odabire neka od jednostavnih heuristika raspoređivanja poslova, ili neki od češće korištenih složenijih algoritama raspoređivanja s kojima se želi napraviti izravna usporedba.

Budući da je za klasu postupaka raspoređivanja s promjenjivim opterećenjem kod kojih nije dozvoljeno prekidanje poslova ($P_m \mid o_i, promjenjivo_nt \mid f$) obično jednostavno konstruirati niz poslova za koje je učinkovitost dobivenog rasporeda jako daleko od optimalnog, teoretska usporedba s optimalnim rasporedom ne donosi informaciju koja bi značajno razlikovala postupke unutar iste klase.

Mjere *f* kojima se utvrđuje uspješnost algoritma raspoređivanja *A* na nekom nizu poslova trebale bi osigurati brzu konvergenciju u vremenu i na dovoljno ekspresivan način izraziti učinkovitost mjerenog algoritma [17]. Najvažnije mjere koje na ekspresivan način izražavaju učinkovitost postupka raspoređivanja poslova prikazane su na slici 3.1.

Srednje vrijeme odziva sustava (*engl. response time*) je mjera učinkovitosti koja je linearno ovisna o trajanju poslova u sustavu. Konvergencija takve mjere ovisi o konvergenciji trajanja poslova u sustavu, a budući da trajanje poslova može značajno vremenski varirati ta se mjera rijetko koristi za usporedbu postupaka raspoređivanja. Srednje vrijeme čekanja u sustavu (*engl. wait time*) ima bolju sposobnost konvergencije budući da ovisi o svojstvima poslova i načinu raspoređivanja poslova. Poželjna osobina raspoređivača

poslova je da pruži stabilno srednje vrijeme čekanja sa što manjim rasipanjem oko te vrijednosti.

Srednje usporenje (*engl. slowdown*) označava koliko je puta odziv nekog posla veći od trajanja tog posla. Najveći problem kod mjerenja učinkovitosti pomoću srednjeg usporenja leži u naglašavanju značaja kratkih poslova. Kratki posao u trajanju 1s koji na izvođenje čeka 20 minuta ima usporenje 1200, dok posao od 100 sekundi koji također na izvođenje čeka 20 minuta ima usporenje 12. Iako su oba navedena scenarija vjerojatno podjednako loša za korisnika budući da mora čekati 20 minuta za izvođenje relativno kratkih poslova, razlika u usporenju iznosi dva reda veličine.

Da bi se uklonio nepotrebno povećan prioritet kratkih poslova, definirano je minimalno dopušteno trajanje posla τ koje se koristi pri izračunu srednjeg ograničenog usporenja (*engl. bounded slowdown*).

Dodatna zamjerka mjeri usporenja poslova leži u činjenici paralelizacija poslova nije razmatrana. U sustavu koji ima posao s trajanjem 100 sekundi i isti se posao paralelizira tako da radi na 10 procesora i traje 10 sekundi moguće su određene anomalije u metrici usporenja. U slučaju kada se posao od 100 sekundi izvrši odmah, a posao od 10 sekundi bude izvršen s 90 sekundi čekanja poslovi imaju usporenja 1 i 10, a zapravo označavaju jednaku količinu posla odrađenu u ukupno 100 sekundi.

Da bi se uklonila takva anomalija predložena [24] je metrika c-normaliziranog ograničenog usporenja koja ograničeno usporenje dijeli s brojem zatraženih procesora. Takva metrika također nije savršena za usporedbu raspoređivača poslova jer je na neki način potrebno nagraditi korisnički trud paralelizacije poslova, a u obzir nisu uzeti ni efekti nelinearne paralelizacije.

Iskoristivost sustava (*engl. utilization*) i ukupno trajanje rasporeda (*engl. makespan*) su mjere učinkovitosti postupaka raspoređivanja koje nisu primjenjive na algoritme $P_m \mid o_i \mid f$. Razlog tome leži u činjenici da posao p_i u takvom sustavu nije moguće poslati na izvršavanje sve do trenutka o_i, što znači da je donja granica ukupnog trajanja sustava $(o_j \mid \forall o_i \, (o_i \leq o_j))$ kojom su ograničeni svi raspoređivači poslova bez obzira na učinkovitost raspoređivanja. Iskoristivost sustava jednaka je omjeru procesorskog vremena utrošenog na izvođenje poslova i ukupno raspoloživog procesorskog vremena. Budući da je utrošeno na izvođenje poslova jednako sumi trajanja svih poslova koja nije ovisna o postupku raspoređivanja poslova, a ukupno raspoloživo procesorsko vrijeme je jednako ukupnom trajanju rasporeda za koji je definirana uvijek važeća donja granica, iskoristivost sustava spada u mjere koje su invarijantne na postupke raspoređivanja u klasi $P_m \mid o_i \mid f$.

Kod analize učinkovitosti algoritama raspoređivanja u ovom i slijedećim poglavljima srednje vrijeme čekanja i srednje ograničeno usporenje korišteni su kao mjere učinkovitosti raspoređivača poslova budući da semantika srednjeg c-normaliziranog ograničenog usporenja nije u potpunosti dorađena.

Ovako definirane metrike ne uključuju specifične želje pojedinih korisnika, nego preferiraju sustave koji maksimiziraju usrednjenu vrijednost reakcije sustava koja se izražava na razne načine. Da bi korisnici izrazili svoje viđenje važnosti određenog posla moguće je proširiti metrike prikazane na slici 3.1 korisnički definiranim težinskim faktorima poslova. Napredniji način izražavanja korisničkih preferencija prema poslovima uključuje specifikacije funkcija koje izražavaju vremensku ovisnost korisnikovog zadovoljstva s potencijalnim trenucima završetka posla [25].

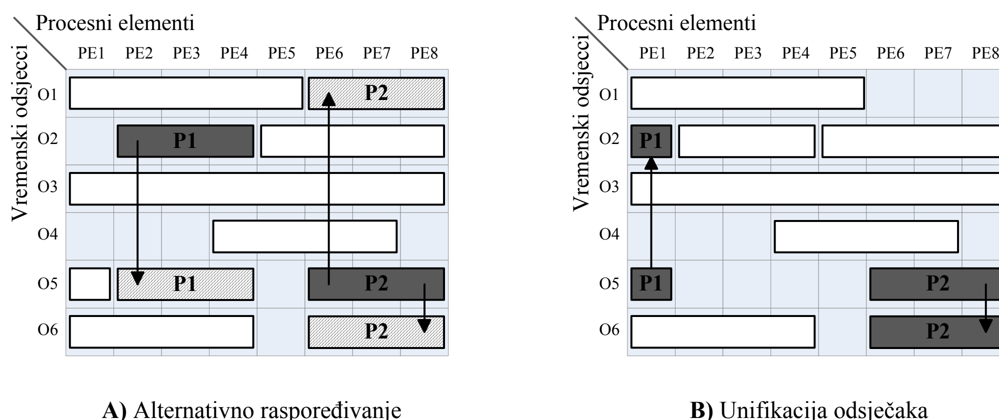
3.3. Postupci raspoređivanja prekidivih poslova

Postupci raspoređivanja prekidivih poslova u paralelnim sustavima $(P_m \mid o_i, promjenjivo_nt, prekidivo \mid f)$ ovise o postojanju mehanizma prekidanja poslova koji treba jamčiti robusnost s obzirom na zaustavljanje i nastavljanje posla. Dodatno se uz zaustavljanje i nastavljanje posla može omogućiti i migracija posla između različitih procesora.

Dostupno vrijeme svih raspoloživih resursa dijeli se na vremenske odsječke pri čemu se svaki vremenski odsječak dodjeljuje nekom poslu. Veličina vremenskog odsječka odabire se s obzirom na trajanje promjene konteksta, odnosno s obzirom na trošak zaustavljanja i ponovnog pokretanja posla. Zbog dijeljenja raspoloživog vremena resursa na vremenske odsječke, ovakvi postupci raspoređivanja nazivaju se postupci dijeljenja vremena (*engl. time sharing*).

Najjednostavniji postupak raspoređivanja poslova zasnovan je na Ousterhoutovoj matrici [26] pri čemu je pretpostavljeno da poslovi ne mogu migrirati. Temeljna ideja postupka leži u činjenici da je u većini slučajeva nužno sve procese jednog posla izvoditi istovremeno da bi taj paralelni posao napredovao. Zbog grupiranja procesa jednog posla i izvođenja cijele grupe zajedno u istom vremenskom odsječku ovakav postupak raspoređivanja naziva se grupno raspoređivanje (*engl. gang scheduling*).

Primjer Ousterhoutove matrice s dva temeljna mehanizma koji omogućuju raspoređivanje dijeljenjem vremenskih odsječaka prikazani su na slici 3.2. Redovi matrice označavaju vremenske odsječke, a stupci matrice označavaju procesne elemente. Raspoređivač poslova u pravilnim vremenskim razmacima mijenja redove, a svaki procesni element podiže prioritet poslu u tom retku. U slučaju u promatranom retku nema



A) Alternativno raspoređivanje **B)** Unifikacija odsječaka

Slika 3.2 Mehanizmi koji omogućuju učinkovito raspoređivanje prekidivih poslova

raspoređenih poslova provodi se alternativno raspoređivanje kako je prikazano na slici. U vremenskom odsječku *O5* na procesne elemente *PE2*, *PE3* i *PE4* se raspoređuje posao *P1* koji originalno pripada vremenskom odsječku *O2*, budući da su ti procesni elementi slobodni. U originalnom algoritmu predloženo je da se procesni elementi pune i poslovima koji se ne mogu u cijelosti prebaciti u ciljani vremenski odsječak, čime bi se omogućilo dvorazinsko raspoređivanje. Na višoj razini raspoređivanja određivao bi se vremenski odsječak, a na nižoj razini bi procesni elementi koji nemaju definiran posao zasebno odabirali poslove iz drugih vremenskih odsječaka.

Kada posao dolazi u sustav, traži se vremenski odsječak koji ima dovoljno slobodnih procesnih elementa. Budući da izbor takvog odsječka ne mora biti jednoznačan, kao i odabir procesnih elemenata unutar odabranog vremenskog odsječka, postoje razne strategije [13][27] za smještanje novog posla u sustav. U slučaju da nema vremenskog odsječka s dovoljno slobodnih procesnih elemenata stvara se novi vremenski odsječak.

U trenutku kada neki posao završava, moguće je da se svi poslovi koji su u istom vremenskom odsječku presele u prazne procesne elemente nekog drugog vremenskog odsječka. Prazan vremenski odsječak se tada može izbrisati (unificirati) iz sustava. Na slici 3.2 B) prikazana je unifikacija vremenskog odsječka *O5*, iz kojeg se poslovi *P1* i *P2* sele u vremenske odsječke *O2* i *O6*.

Postupak grupnog raspoređivanja može se optimirati smanjenjem veličine grupe, pri čemu samo procesi koji iskažu međusobnu interakciju mogu biti u istoj grupi ili korisnici mogu posebnom sintaksom označiti procese jednog posla koji iskazuju veliku međuzavisnost za vrijeme izvođenja.

Postoje različite modifikacije grupnog raspoređivanja poslova koje dozvoljavaju promjenu broja procesa u svakom poslu za vrijeme izvođenja posla [28] zbog povećanja učinkovitosti i smanjenja fragmentacije u sustavu.

Da bi se omogućilo učinkovitije grupno raspoređivanje uz eliminaciju problema sa sinkroniziranom promjenom vremenskog odsječka na jako velikim sustavima, raspoređivanje je moguće vršiti raspodijeljenim hijerarhijskim sustavom kod kojeg hijerarhijska struktura raspoređivača poslova brine o resursima u izdvojenom dijelu sustava [29].

Navedeni postupci grupnog raspoređivanja fokusirani su na optimizaciju vremena procesnih elemenata uz smanjenje fragmentacije. Kod takvog pristupa se može desiti da zbog velikog broja vremenskih odsječaka i poslova koji se izvršavaju na jednom procesnom elementu, taj procesni element ili grupa procesnih elemenata koji dijele istu memoriju ostane bez slobodnog memorijskog prostora. Da bi se spriječio takav scenarij uvodi se kontrola pristupa (*engl. admission control*) [30] koja poslovima ne dopušta ulazak u sustav u slučaju da nije raspoloživa dovoljna količina memorijskog prostora.

Grupno raspoređivanje poslova koristi se u sustavima kod kojih je omogućeno prekidanje više procesnih elemenata na razini sklopovlja. Takvo prekidanje je tipično moguće kod MPP sustava, dok se kod grozdova računala podrška sinkroniziranom prekidanju poslova nužno ostvaruje u programskoj potpori. Zbog kompleksnosti izrade programske potpore prekidanju paralelnih primjenskih programa, postupci raspoređivanja prekidivih poslova se uglavnom ne koriste na grozdu računala.

Temeljna prednost grupnog raspoređivanja je onemogućavanje dugih poslova da izgladnjuju novopridošle kratke poslove, što rezultira smanjenim prosječnim usporenjem. Grupno raspoređivanje omogućuje privid trenutnog vremena izvođenja za interaktivne poslove.

3.4. Postupci raspoređivanja neprekidivih poslova

Postupci raspoređivanja neprekidivih poslova na homogenim paralelnim sustavima $(P_m \mid o_i, promjenjivo_nt \mid f)$ zasnivaju se na različitim heuristikama budući da je za bilo koji postupak raspoređivanja u tako promjenjivoj okolini lako naći primjer opterećenja u kojem će se taj postupak pokazati značajno neučinkovitijim od optimalnog raspoređivača poslova. Temeljni razlog nemogućnosti definicije savršenog rasporeda leži u problemu predviđanja budućnosti pri čemu se odluka o raspoređivanju poslova donosi samo na temelju trenutno poznatog stanja sustava. Budući da je zbog tehnoloških ograničenja raspoređivanje prekidivih poslova teško ostvarivo na grozdu računala, a prirodno odgovara MPP sustavima,

postupci raspoređivanja neprekidivih poslova se gotovo ekskluzivno koriste na grozdovima računala.

Kod raspoređivanja poslova na grozdu računala korisnici poslove šalju centraliziranom raspoređivaču poslova koji donosi odluke o vremenima izvođenja tih poslova. Na različitim grozdovima računala definirani su različiti postupci dodjeljivanja prioriteta korisničkim poslovima. Najprirodniji način određivanja prioriteta poslova jednostavna je funkcija koja u obzir uzima samo redoslijed dolazaka poslova u sustav. Takva strategija nije u skladu s jednolikom podjelom resursa paralelnog sustava među korisnicima, pa se često pri određivanju prioriteta poslova uz vrijeme dolaska posla koriste dodatni kriteriji. Najčešće korišteni dodatni kriteriji pri određivanju prioriteta uključuju intenzitet korištenja sustava od strane korisnika ili grupe korisnika unutar nekog definiranog intervala i stupanj paralelnosti posla.

Da bi se postupak raspoređivanja poslova odijelio od načina dodjeljivanja prioriteta poslovima na temelju odnosa među korisnicima, kod svih postupaka raspoređivanja poslova se pretpostavlja da postoji ukupan uređaj prioriteta poslova na ulazu te da je u skladu s tim prioritetima potrebno donijeti odluke o pokretanju poslova. Različiti raspoređivači poslova na drugačije načine uzimaju u obzir zadanu listu prioriteta da bi se poboljšala ukupna učinkovitost raspoređivanja poslova. Postupak raspoređivanja poslova na paralelnim sustavima se zasniva na traženju najboljeg kompromisa između učinkovitosti i prioriteta definiranih sa strane korisničke politike na grozdu računala.

S obzirom da se nemoguće približiti optimalnom postupku raspoređivanja u zadanoj okolini u ovom odjeljku analizirane su različite heuristike raspoređivanja poslova te su dane usporedbe učinkovitosti najčešće korištenih postupaka. Najjednostavnije heuristike raspoređivanja koje se koriste kod jednoprocesorskih i višeprocesorskih sustava s prekidivim i neprekidivim poslovima opisane su odjeljku 3.4.1 s naznakom njihove upotrebljivosti na grozdu računala. Raspoređivanje poslova unazadnim popunjavanjem praznina te razne varijante postupaka unazadnog popunjavanja praznina prikazane su u odjeljcima 3.4.2 i 3.4.3. U odjeljku 3.4.4 opisani su različiti složeni heuristički postupci raspoređivanja neprekidivih poslova na grozdu računala koji su oblikovani uz različita ograničenja ili sa specifičnim ciljanim metrikama. Izvorni postupak raspoređivanja poslova temeljen na unazadnom popunjavanju praznina uz optimizaciju dinamičkim programiranjem dan je u odjeljku 3.4.5.

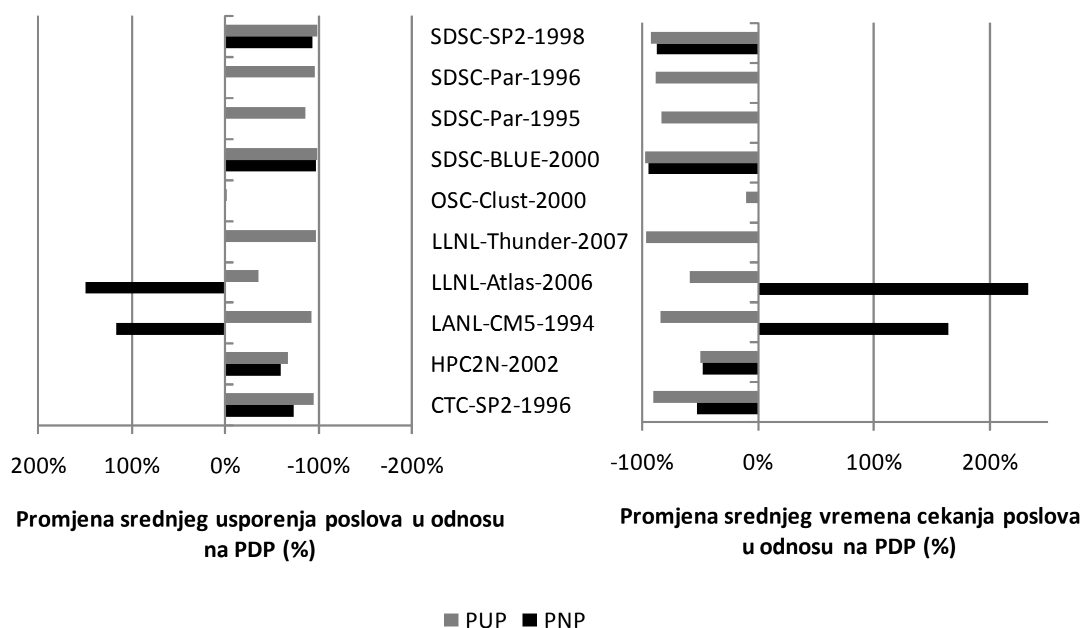
3.4.1. Jednostavni heuristički algoritmi za raspoređivanje neprekidivih poslova na računalnom grozdu

Jednostavni heuristički algoritmi za raspoređivanje poslova mogu se podijeliti u dvije skupine s obzirom na nužnost informacije o trajanju poslova. Postupci kod kojih nije potrebna informacija o trajanju posla uključuju postupak raspoređivanja prema definiranim prioritetima (PDP) [31], postupak prioritiziranja najšireg posla (PŠP) i postupak prioritiziranja najužeg posla (PUP). Kod širine posla podrazumijeva se količina resursa koji su potrebni za izvršavanje posla. Kod postupka prioritiziranja najkraćeg posla (PNP) [31] točna vrijednost ili barem procjena trajanja posla je nužna u trenutku donošenja odluke o njegovom pokretanju.

Postupak PDP se pretvara u postupak raspoređivanja prema redoslijedu prispijeća (*engl. first come – first served*) u slučaju da definirani prioriteti odgovaraju redoslijedu prispijeća. Ovaj postupak raspoređivanja primjenjiv je samo kod neprekidivih poslova, pri čemu se jamči najveća moguća pravednost raspoređivanja uz najveći gubitak učinkovitosti raspoređivanja poslova. U nekim slučajevima se *PDP* postupak koristi u kombinaciji s prekidanjem poslova u sustavu [32] pri čemu visoko paralelni poslovi koji traju dulje od definirane granice prelaze u prekidiv način rada upareni s poslovima koji čekaju na izvršenje. S obzirom da je PDP postupak jednostavan za implementaciju često se koristi kao referentna točka za uspoređivanje postupaka raspoređivanja poslova.

Motivacija za korištenje postupka PNP kod raspoređivanja na grozdu računala potječe iz teoretskog rezultata o ostvarenju minimalnog srednjeg vremena čekanja kod korištenja istog postupka na jednoprocesorskom sustavu s unaprijed poznatim trajanjima svih poslova [33]. Iako je računalni grozd kompleksnija okolina od ovakvog jednoprocesorskog sustava, pretpostavka je da će se korištenjem PNP postupka na računalnom grozdu postići rezultati bliski najmanjem mogućem srednjem vremenu čekanja poslova. Pri korištenju PNP postupka na računalnim grozdovima uočeni su nedostaci poput nepoznavanja trajanja poslova i mogućnosti izgladnjivanja dugih poslova. Varijanta PNP postupka koja se koristi u sustavima kod kojih je moguće prekidanje poslova uključuje povećanje prioriteta poslova s najkraćim preostalim vremenom procesiranja [34].

Na očekivano dobrim rezultatima za srednje vrijeme čekanja koje je moguće ostvariti postupkom PNP zasniva se i postupak prioritiziranja najužeg posla. Temeljna pretpostavka o postojanju korelacije između trajanja posla i količine zatraženih resursa zaslužna je za oblikovanje PUP postupka [35]. Budući da poslovi s manjim zahtjevima na resurse ne završavaju nužno ranije od ostalih poslova korištenje ovog postupka ne rezultira očekivano niskim srednjim vremenima izvođenja poslova.



Slika 3.3 Udio u broju poslova i vremenu izvođenja za poslove sa različitim statusima

Postupak PŠP zasnovan je na preoblikovanju prioritetne liste prema zauzeću resursa pri čemu poslovi s najviše zatraženih resursa imaju najveći prioritet. Temeljna motivacija za korištenje tog postupka dolazi iz problema pakiranja (*engl. bin packing*) gdje se pokazuje da strategija pakiranja predmeta sortiranih padajuće po širini u posudu (*engl. bin*) smanjuje fragmentaciju ukupnog rezultata.

Kod postupaka PUP i PŠP temeljenih na količini zatraženih resursa može doći do izgladnjivanja najširih i najužih poslova.

Usporedba postupka raspoređivanja poslova PNP i PUP s postupkom PDP na deset računalnih grozdova prikazana je na slici 3.3. Kod postupka PNP su umjesto stvarnih trajanja poslova korištene dostupne korisničke procjene. Na slici nije naznačena učinkovitost postupka PŠP, zato jer je za više redova veličine lošija od ostalih postupaka što bi rezultiralo smanjenjem ilustrativnosti usporedbe ostalih postupaka. Primjena postupka PŠP uzrokuje povećanje srednjeg usporenja od 24% do 2302% te povećanje srednjeg vremena čekanja od 27% do 2065% u odnosu na PDP postupak.

Kod usporedbe PNP i PUP postupaka s PDP postupkom raspoređivanja vidljivo je da su oba postupka u većini slučajeva značajno uspješnija u raspoređivanju sa stanovišta srednjeg usporenja i srednjeg vremena čekanja. U iznimnom slučaju kod računalnih *grozdova LLNL-Atlas-2006* i *LANL-CM5-1994* postupak raspoređivanja PNP se pokazao značajno lošijim od PDP postupka čemu su uglavnom razlog iznimno loše korisničke procjene trajanja poslova koje su korištene umjesto stvarnih vremena trajanja poslova. Postupci raspoređivanja PNP i

PDP imaju jednaku učinkovitost raspoređivanja kod računalnih grozdova *LLNL-Thunder-2007*, *SDSC-Par-1995* i *SDSC-Par-1996* budući da je zbog značajnog manjka korisničkih procjena trajanja poslova korištena standardna vrijednost od 55 sati pri čemu postupak PNP konvergira u postupak PDP.

S obzirom da se postupak PUP temelji na pretpostavci korelacije širine posla i njegovog trajanja, a postupak PNP na uspješnosti korisničke procjene trajanja posla temeljem usporedbe oba postupka ustanovljeno je da postupak PUP uspješniji na svim prikazanim računalnim grozdovima za 2% do 292%. To dovodi do zaključka da su korisničke procjene trajanja poslova toliko neprecizne ili manjkave da ih je na svim prikazanim računalnim grozdovima bolje zamijeniti s količinom zatraženih resursa. Detaljnije mjerenje nepreciznosti korisničkih procjena trajanja poslova i postupci unaprjeđenja tih procjena prikazani su u poglavlju 7.

Za računalne grozdove *CTC-SP2-1996*, *HPC2N-2002*, *SDSC-BLUE-2000* i *SDSC-SP2-1998* kod kojih je dana razmjerno precizna korisnička procjena trajanja postupci PNP i PUP imaju sličnu učinkovitost, pri čemu se razlika u poboljšanju usporenja u odnosu na PDP postupak nalazi u intervalu od 1% do 21%.

Kod usporedbe srednjeg usporenja i srednjeg vremena čekanja vidljivo je da metrika srednjeg usporenja pokazuje veća poboljšanja od metrike srednjeg vremena čekanja budući da je osjetljivija na prioritiziranje kratkih poslova.

3.4.2. Raspoređivanje poslova unazadnim popunjavanjem praznina

Analiza u prethodnom odjeljku pokazala je da je algoritam PUP s podizanjem prioriteta uskih poslova najuspješnija jednostavna heuristika raspoređivanja poslova. S obzirom da su temeljni nedostaci te heuristike potencijalno izgladnjivanje širokih poslova i potpuno zanemarivanje prioriteta poslova određenih poslovnom politikom računalnog grozda u većini slučajeva njeno korištenje nije prihvatljivo. Isti zaključak se može primijeniti na sve jednostavne heurističke postupke iz prethodnog odjeljka osim na PDP postupak koji savršeno prati korisnički definirane prioritete poslova, ali ima najnižu učinkovitost raspoređivanja.

Različite varijante postupka unazadnog popunjavanja praznina oblikovane su sa ciljem održavanja visoke učinkovitosti raspoređivanja uz sprečavanje izgladnjivanja poslova. Konzervativni i agresivni postupci unazadnog popunjavanja praznina koji su teoretski najznačajnije metode raspoređivanja poslova na grozdu računala opisani su u odjeljcima 3.4.2.1 i 3.4.2.2.

3.4.2.1 Konzervativni postupak unazadnog popunjavanja praznina

Konzervativni postupak unazadnog popunjavanja praznina (KUPP) omogućava da poslovi nižeg prioriteta zapune praznine u rasporedu nastale raspoređivanjem poslova višeg prioriteta, čak i u slučajevima gdje to dovodi do ranijeg izvršavanja poslova nižeg prioriteta [36]. Pri popunjavanju praznina nije dozvoljeno da posao nižeg prioriteta odgodi izvođenje posla višeg prioriteta. Da bi se omogućilo popunjavanje praznina nužna je informacija o trajanju posla, pri čemu se koriste korisničke procjene trajanja. U slučajevima kod kojih trajanje posla premaši korisničku procjenu dolazi do prekidanja tog posla od strane raspoređivača poslova.

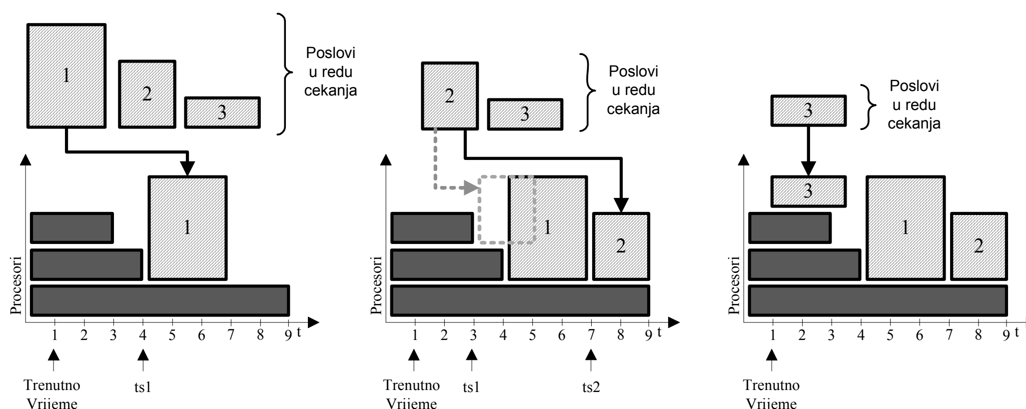
Kod postupaka raspoređivanja neprekidivih poslova na grozdu računala dolazak novog posla i završavanje izvođenja posla koji se trenutno izvodi su dva temeljna događaja koji zahtijevaju obradu raspoređivača poslova. Neki raspoređivači poslova imaju unificirani postupak obrade svakog od ta dva događaja, dok se kod nekih postupaka zbog očuvanja različitih svojstava algoritma raspoređivanja ti događaji tretiraju na različite načine. Kod KUPP postupka raspoređivanja poslova elementarni događaji se obrađuju na različite načine da bi se osiguralo poznavanje minimalnog početka izvođenja za sve poslove od trenutka njihovog ulaska u sustav.

Ostvarenje KUPP postupka zasniva se na dvije strukture podataka. Prva struktura podataka je red pripravnih poslova, a druga je profil očekivanog iskorištenja resursa. U profilu očekivanog iskorištenja procesora za svaki vremenski trenutak dan je popis zauzetih resursa. Postupak obrade dolaska novog posla opisan je na slici 3.4. Algoritam se zasniva na uzastopnom traženju prve točke u kojoj postoji dovoljno raspoloživih resursa za određeni

```
KUPP_Novi_Posao(posao p){
  Ts=PronađiTočkuSidrenja(p)
  Modificiraj profil tako da se
    alociraju potrebni resursi za
    posao p od trenutka Ts
  Ako(Ts=trenutno vrijeme)
    pokreni posao p
}
```

```
PronađiTočkuSidrenja(posao p){
  t=trenutno vrijeme;
  radi(
    Pretraži profil i pronađi prvi vremenski
      trenutak t1>=t u kojem ima dovoljno
      raspoloživih resursa za posao p;
    Pronađi maksimalan t2, takav da t2>t1
      i svi resursi potrebni za izvođenje
      posla p su slobodni u intervalu (t1,t2);
    t=t1;
  )dok(t2-t1<TrajanjePosla(p));
  Vrati t1;
}
```

Slika 3.4 Obrada dolaska novog posla KUPP postupkom



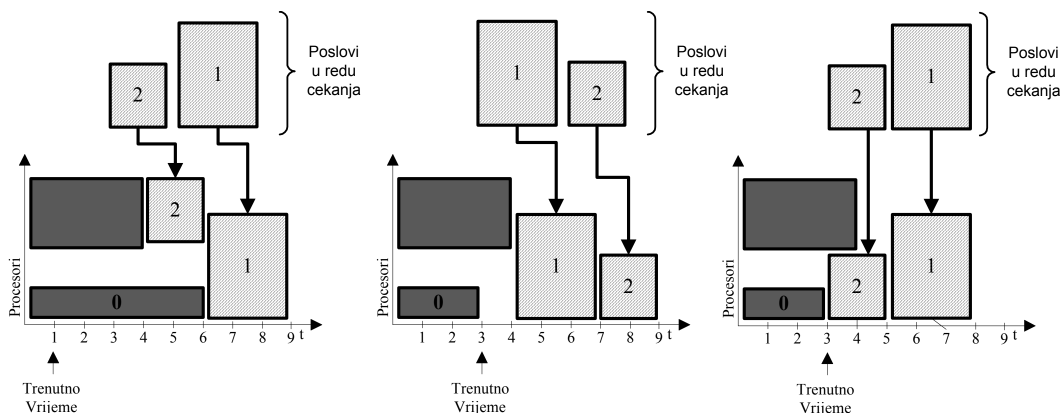
Slika 3.5 Primjer raspoređivanja poslova KUPP postupkom

posao . Takva točka naziva se točkom sidrenja (*engl. anchor point*).

Na slici 3.5 dan je primjer izvođenja postupka obrade novog posla. U vremenskom trenutku *t=4* u redu čekanja se nalaze tri posla s označenim prioritetima pri čemu jedinica označava posao najvećeg prioriteta. Postupak obrade novog posla provodi se posebno za svaki od poslova prema njihovom prioritetu. Za posao s prioritetom *1* trenutak *t=4* je prva točka u kojoj ima dovoljno raspoloživih resursa za izvođenje tog posla. Budući da su resursi raspoloživi od tog trenutka pa sve do isteka trajanja posla ta točka se proglašava točkom sidrenja. Za posao s prioritetom *2* postoje dvije potencijalne točke sidrenja u trenucima *t=3* i *t=7*, pri čemu je prva potencijalna točka sidrenja neodgovarajuća zato jer počevši od te točke nema dovoljno raspoloživih resursa za potpuno izvođenje tog posla. Točka sidrenja posla s prioritetom 3 jednaka je promatranom trenutku *t=4* pa se taj posao odmah pokreće.

Složenost izvedbe ovog postupka za svaki novi posao ovisna je o veličini profila. Količina podataka u profilu razmjerna je broju poslova u sustavu za koje je napravljena rezervacija, a za svaki novi posao se profil pretražuje samo jednom, što uz dobru organizaciju praćenja zauzeća resursa dovodi do linearne složenosti postupka s obzirom na broj poslova.

Kod algoritma KUPP definiran je zasebni postupak obrade završetka svakog posla zato da bi se korisnicima informacija o najgorem mogućem vremenu pokretanja posla pružila u trenutku slanja posla u sustav. Većina poslova na računalnom grozdu završava ranije od procijenjenog vremena trajanja, što otvara mogućnost da se smanje početna vremena nekih rezervacija. Trivijalni način za pomicanje vremena rezervacija prema naprijed je otkazivanje rezervacija svih poslova koji nisu počeli s izvođenjem te provedba postupka *KUPP_Novi_Posao* nad tim poslovima.



Slika 3.6 Prikaz problema očuvanja jamstva najlošijeg početnog vremena posla

Ovakva strategija jamči očuvanje prioriteta svih poslova, odnosno svojstvo da prioritetniji poslovi neće nikad biti odloženi zbog poslova nižeg prioriteta, ali bi dovela do nemogućnosti jamstva najgoreg početnog vremena izvođenja posla. Na slici 3.6 prikazan je scenarij kod kojeg dolazi do narušavanja jamstva početka izvođenja posla.

U vremenskom trenutku *t=1* stvaraju se rezervacije za poslove s prioritetima *1* i *2* pri čemu posao s prioritetom *2* popunjavanja prazninu nastalu rezervacijom posla *1*. Posao s prioritetom *0* završava ranije od procijenjenog vremena trajanja i u trenutku *t=3* dolazi do promjena rezervacija za poslove s prioritetima *1* i *2*. U slučaju uklanjanja rezervacija za sve poslove i njihovim naknadnim postavljenjem prema redoslijedu prioriteta dobiva se konačni raspored vidljiv na srednjem dijelu slike 3.6. Kod takvog rasporeda povećano je očekivano vrijeme početka posla prioriteta *2 s t=4* na *t=7*, ali nije došlo do odgađanja prioritetnijeg posla na štetu posla s nižim prioritetom.

U slučaju da se žele očuvati jamstva najgorih vremena početka provodi se kompresija postojećeg rasporeda. Kompresija rasporeda je postupak kod kojeg se za svaki posao prvo uklanja rezervacija u postojećem profilu i onda se pokušava napraviti nova rezervacija za taj isti posao u najranijem mogućem trenutku pomoću postupka *KUPP_Novi_Posao*. Time se jamči da se inicijalno postavljena vremena početka poslova mogu popraviti, ili će u najgorem slučaju ostati jednaka. Rezultat kompresije rasporeda prikazan je na desnom dijelu slike 3.6, gdje su rezervacije za oba posla pomaknute za jedan vremenski odsječak prema naprijed. Ukupna učinkovitost raspoređivanja na danom primjeru povećana je korištenjem kompresije rasporeda uz očuvanje jamčenih vremena početaka poslova. Takvo povećanje učinkovitosti omogućeno je inverzijom prioriteta poslova.

Složenost postupka kompresije rasporeda je kvadratna s obzirom na broj poslova, budući da je za svaki posao potrebno ukloniti rezervaciju i pokrenuti postupak *KUPP_Novi_Posao* koji je linearne složenosti s obzirom na broj poslova.

3.4.2.2 Agresivni postupak unazadnog popunjavanja praznina (AUPP)

KUPP postupak raspoređivanja poslova redom radi rezervacije za poslove prema padajućem prioritetu pazeći da pritom poslovi manjeg prioriteta ne uzrokuju odgađanje svih poslova većeg prioriteta. Agresivni postupak unazadnog popunjavanja praznina (*engl. easy backfilling*) [37] osigurava rezervaciju samo za najprioritetniji posao dok je pokretanje u trenutku raspoređivanja moguće za sve poslove sve dok se ne mijenja rezervacija najprioritetnijeg posla u redu čekanja.

AUPP postupak koji se sastoji od dva uzastopna koraka detaljno je prikazan na slici 3.7. Prvi korak u postupku je pronalaženja točke sidrenja za najprioritetniji posao u redu čekanja i nepotrebnih resursa pomoću metode *PronađiTočkuSidrenjaINepotrebneResurse*. Nepotrebni resursi su resursi koji bi ostali slobodni nakon izvršenja najprioritetnijeg posla u točki sidrenja. U drugom koraku postupka poziva se metoda *PopuniPraznine* koja pokreće ostale poslove tako da ne naruše mogućnost izvršavanja najprioritetnijeg posla u točki sidrenja.

Kod odabira poslova za popunjavanje praznina pretražuje se cijeli red čekanja prema

```
PronađiTočkuSidrenjaINepotrebneResurse(){
  ls=Lista poslova koji se trenutno
     izvršavaju sortirana prema
     očekivanom vremenu završetka
  resursi=SlobodniResursi()
  p0=posao najvećeg prioriteta iz reda
     čekanja
  pi=Prvi(ls)
  dok(resursi<Zahtjevi(p0)){
    resursi=resursi+Resursi(pi)
    pi=Slijedeći(pi)
  }
  ts=VrijemeZavršetka(Prethodni(pi))
  nepotrebniResursi=resursi-Zahtijevi(p0)
}
```

```
PopuniPraznine(ts, nepotrebniResursi){
  ls=Lista poslova iz reda čekanja
     sortirana prema padajućem prioritetu
     (bez najprioritetnijeg posla)
  pi=Prvi(ls)
  dok(pi<>Kraj(ls)){
    ako(Trajanje(pi)<ts)∧
       (resursi(pi)<=SlobodniResursi()))
      Pokreni(pi)
    inače{
      ako((Trajanje(pi)>ts) ∧
      (resursi(pi)<=Min(SlobodniResursi(),
                         nepotrebniresursi))
        Pokreni(pi)
    }
    pi=Slijedeći(pi)
  }
}
```

Slika 3.7 AUPP postupak raspoređivanja poslova

padajućim prioritetima i pokreću poslovi koji se mogu trenutno pokrenuti. Poslovi koji se mogu trenutno pokrenuti uključuju poslove kojima su trenutno dostupni resursi dovoljni i čije će izvršavanje završiti prije točke sidrenja najprioritetnijeg posla. U slučaju da trajanje posla premašuje točku sidrenja, posao je moguće izvršiti ukoliko njegovi zahtjevi za izvršenjem ne premašuju broj trenutno dostupnih resursa i resursa koji su nepotrebni poslu najvećeg prioriteta u točki sidrenja.

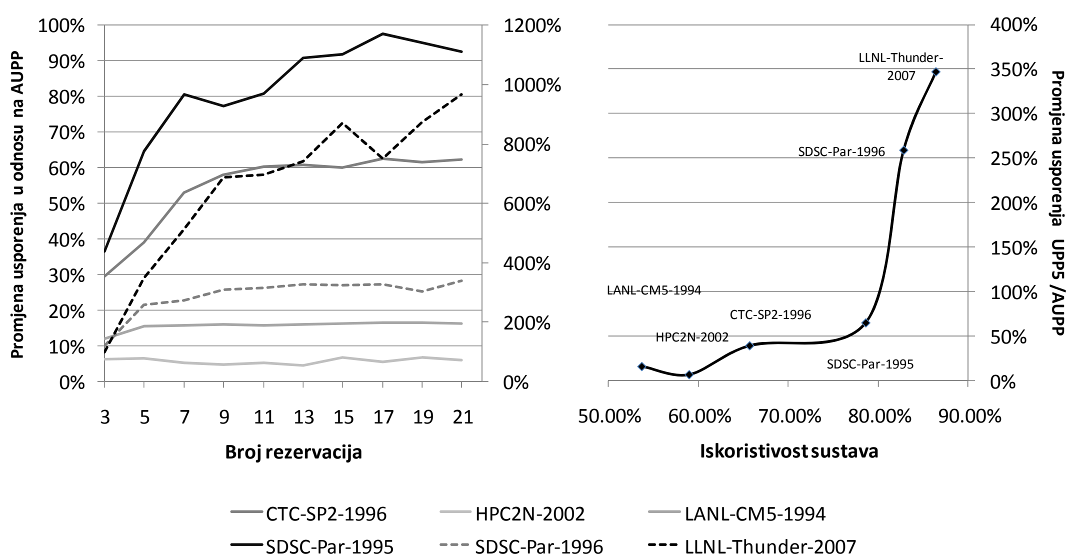
Poslovima koji nemaju najveći prioritet u redu čekanja se može desiti da budu odgađani u korist poslova nižeg prioriteta, s time da je vrijeme odgađanja rezervacije posla p ograničeno umnoškom broja poslova s prioritetom većim od posla p i ukupnim ograničenjem trajanja poslova na računalnom grozdu. Iz ovoga slijedi da kod AUPP postupka raspoređivanja nema izgladnjivanja poslova.

Motivacija kod raspoređivanja poslova postupkom AUPP leži u boljoj iskorištenosti sustava zahvaljujući relaksiranom pridržavanju prioriteta posla. Zbog povećanja učinkovitosti raspoređivanja postupak AUPP implementiran je u vodećim sustavima za raspoređivanje poslova koji uključuju MOAB [78], Maui[79], OpenPBS[80], Sun GridEngine [81] i IBM LoadLeveler [82]. U analizi 50 najsnažnijih paralelnih sustava s top 500 liste [73] iz 2004 godine ustanovljeno je [38] da se u 60% slučajeva koristi AUPP postupak raspoređivanja poslova.

AUPP postupak raspoređivanja poslova se vrlo često neznatno modificira da bi se povećalo poštivanje prioriteta uz neznatno smanjenje učinkovitosti raspoređivanja poslova. Izmjena se provodi uvođenjem fiksnog broja rezervacija za najprioritetnije poslove, pri čemu se taj broj rezervacija može konfigurirati. Ovakav unazadni postupak popunjavanja praznina označen je s UPPX pri čemu X predstavlja broj fiksnih rezervacija što znači da je UPP1 sinonim za postupak AUPP.

Na slici 3.8 prikazana je učinkovitost raspoređivanja poslova UPPX algoritma na 6 grozdova računala za različiti broj rezervacija poslova. Na lijevom grafikonu prikazana je promjena usporenja s obzirom na UPP1 postupak raspoređivanja poslova. Za prikaz podataka koriste se dvije osi ordinata zbog velikih razlika u promjeni učinkovitosti na različitim računalnim grozdovima, pri čemu su vrijednosti za promjenu usporenja računalnih grozdova označenih s isprekidanom linijom prikazani na desnoj osi.

Kod svih prikazanih računalnih grozdova vidljiv je porast srednjeg usporenja poslova s brojem korištenih rezervacija pri raspoređivanju poslova. Najveći porast ostvaren je za do 5 rezervacija najprioritetnijih poslova, a za više od pet rezervacija brzina promjene učinkovitosti opada. Za veliki broj rezervacija poslova ovaj postupak odgovara KUPP



Slika 3.8 Utjecaj broja rezervacija na učinkovitost UPPX postupka

postupku raspoređivanja poslova bez kompresije rasporeda, odnosno s ponovnim izračunom cijelog rasporeda u trenutku završetka nekog od poslova.

Različita iskoristivost računalnih grozdova je temeljni razlog postojanja velikih razlika promjene učinkovitosti nastale korištenjem različitog broja rezervacija. Na desnom grafikonu slike 3.8 prikazan je odnos iskoristivosti sustava i promjene srednjeg usporenja postupka UPP5 u odnosu na postupak UPP1. Kod računalnih grozdova *LANL-CM5-1994* i *HPC2N-2002* kod kojih je iskoristivost sustava manja od 60% promjena učinkovitosti zbog korištenja pet rezervacija umjesto jedne je manja od 15%. Za računalne grozdove *CTC-SP2-1996* i *SDSC-Par-1995* promjena srednjeg usporenja je manja od 64% pri čemu je najveća iskoristivost sustava 78%. Broj rezervacija jako utječe na raspoređivanje poslova u sustavima kod kojih je iskoristivost veća od 80%, pri čemu se prosječno usporenje može promijeniti i za nekoliko puta. Računalni grozd *LLNL-Thunder-2007* ima iskoristivost od 86% pri čemu je zbog korištenja postupka UPP5 srednja vrijednost usporenja povećana za 3.46 puta u odnosu na postupak UPP1.

Kod odabira konfiguracije UPPX postupka preporuča se što veći broj rezervacija na sustavima s vrlo malom iskoristivosti. Na sustavima sa srednjom iskoristivosti (od 60% do 75%) korisno je koristiti manji broj fiksnih rezervacija, dok se kod sustava koji rade s velikim opterećenjem preporuča korištenje UPP1 postupka da bi se spriječilo višestruko povećanje srednje vrijednosti usporenja poslova.

3.4.3. Modifikacije postupaka unazadnog popunjavanja praznina

Budući da je postupak popunjavanja praznina najčešće korišteni postupak raspoređivanja neprekidivih poslova razvijene su razne varijante tog postupka u svrhu povećanja učinkovitosti ili pravednosti s obzirom na zadane prioritete.

Uvođenjem relaksacije u UPP postupak pri čemu se prioriteti poslova zanemaruju, a rezervacije se provode samo na poslovima koji su predugo čekali na izvođenje moguće je smanjiti srednje usporenje do 70% [39]. Predugo čekanje definirano je s obzirom na prosječna usporenja poslova u sustavu, što znači da ako neki posao u sustavu čeka toliko da bi njegovo trenutno izvođenje narušilo prosječno usporenje sustava tada se za njega stvara rezervacija. Da bi se spriječilo favoriziranje kraćih poslova korištenjem metrike usporenja, predloženo je razdvajanje praćenja usporenja različitih klasa poslova te odlučivanje o postavljanju rezervacija za poslove čije odgađanje narušava srednje usporenje poslova unutar iste klase.

Sličnim pristupom modifikacije postupka UPP koji također uključuje odjeljivanje više klasa poslova [40] ostvarena su smanjenja srednjeg usporenja na tri računalna grozda za 20% do 70%, dok je na jednom računalnom grozdu usporenje povećano za 60%. Izmjena postupka UPP kod kojeg su ostvareni ovi rezultati odnosi se na podjelu reda čekanja poslova na četiri manja reda čekanja s obzirom na procijenjeno trajanje poslova, uz podjelu resursa u sustavu posebno za svaku klasu trajanja poslova. Kod postupka raspoređivanja dozvoljeno je da se pokrenu poslovi iz svakog reda čekanja sve dok se poštuju rezervacije za prvi posao u tom redu. Prilikom popunjavanja praznina dozvoljeno je posuđivanje resursa namijenjenih za druge klase poslova.

Temeljni postupci unazadnog popunjavanja praznina KUPP i AUPP provode rezervacije za poslove slijedno, pri čemu je redoslijed stvaranja rezervacija određen prioritetima poslova. Reorganizacija načina stvaranja rezervacije poslova pri čemu se promatra više poslova odjednom može dovesti do poboljšanja učinkovitosti raspoređivača poslova [41] s 30% smanjenjem srednjeg usporenja. Raspoređivanje više poslova odjednom na optimalan način je NP težak problem, ali se može pojednostavniti na polinomnu složenost uz uvođenje određenih ograničenja.

Svi navedeni postupci funkcioniraju za poslove kod kojih je zabranjeno prekidanje poslova pri čemu su resursi potrebni za izvršavanje nekog posla poznati unaprijed. Kod nekih vrsta poslova unaprijed je poznat raspon dodijeljenih resursa ili su poslovi u toku izvođenja sposobni mijenjati zahtjeve za resursima. Modifikacija UPP postupka [42] uključuje korištenje linearnog modela paralelizacije poslova što dovodi do smanjenja

srednjeg vremena odziva sustava od 50% uz pretpostavku da je 25% poslova nema fiksne zahtjeve na količinu resursa.

Kod AppLes raspoređivača poslova (*engl. Application Level Scheduler*) [43] korisnici pri slanju posla u sustav mogu odabrati nekoliko različitih konfiguracija poslova s obzirom na zahtijevane resurse. Raspoređivač poslova na temelju dobivenih konfiguracija simulira moguće rasporede poslova i kao relevantnu konfiguraciju uzima onu koja ostvaruje minimalno vrijeme završetka posla. AppLes raspoređivač poslova smanjuje srednje vrijeme odziva sustava za 30%, pri čemu su moguća odstupanja ovisno o opterećenju sustava.

SCOJO-P raspoređivač poslova [44] kod određivanja resursa potrebnih za izvršenje posla u obzir uzima sve poslove u trenutnom redu čekanja uz statističko modeliranje dolazaka budućih poslova. Takvom strategijom raspoređivanja postiže se 70% manje srednje vrijeme odziva sustava u odnosu na AUPP postupak raspoređivanja poslova. U usporedbi s AppLes postupkom raspoređivanja poslova, SCOJO-P postupak postiže 59% manje srednje vrijeme odziva sustava. Procijenjeno je da korišteno statističko modeliranje budućih poslova ne unapređuje značajno postupak raspoređivanja poslova.

3.4.4. Složeni heuristički postupci raspoređivanja poslova

U ovom odjeljku opisani su postupci raspoređivanja poslova koji se mogu primijeniti na računalnom grozdu, ali zbog načina oblikovanja ili ciljane metrike nisu izravno usporedivi s postupcima unazadnog popunjavanja praznina.

Načelni cilj postupaka raspoređivanja poslova je održati pravednost među korisnicima uz održavanje visoke iskorištenosti dostupnih resursa. Pravednost među korisnicima obično je definirana prioritetnim redom poslova u kojem poslovi korisnika koji su u prošlosti manje koristili grozd računala imaju viši prioritet. Korisničko ocjenjivanje poželjnosti izvršavanja određenih poslova i stupanj hitnosti nisu naznačeni.

Kod tržišno orijentiranog postupka raspoređivanja poslova [45] korisnici na aukciji određuju cijenu određenih vremenskih intervala za izvođenje poslova. Pobjednik aukcije virtualnim novcem plaća za izlicitirani vremenski interval na računalnom grozdu. Svakom korisniku se s vremenom povećava količina raspoloživog virtualnog novca. Takvim pristupom omogućeno je da korisnici koji rjeđe koriste grozd računala ili su u izrazitoj žurbi dođu ranije do potrebnih resursa za izvođenje posla. Nedostatak takvog pristupa je mogućnost lažnog licitiranja u svrhu umjetnog povećanja vrijednosti neželjenog intervala za izvođenje na štetu ostalih korisnika.

Postupak grupnog spajanja (*engl. gang matching*) [46] namijenjen je kompliciranijim raspodijeljenim sustavima koji nisu nužno u zajedničkoj administrativnoj domeni, s

raznolikijim vrstama resursa te potencijalnim ograničenjima kod korištenja resursa. Raznolike vrste resursa uključuju standardne procesore i memoriju, uz dodatke programske opreme, programskih licenci te različitih dodatnih uređaja koji mogu biti uključeni u računalnu mrežu ili spojeni na neki od računala. Neki od resursa poput programskih licenci mogu biti vezane za druge resurse poput skupova računala na kojima je moguće koristiti te licence.

Kod postupka grupnog spajanja definirane su uloge čijim izvođenjem dolazi do izgradnje rasporeda poslova. Tri uloge koje se koriste su podnositelj zahtjeva (*engl. requestor*), pružatelj usluge (*engl. provider*) i spajatelj (*engl. matchmaker*). Podnositelj zahtjeva i pružatelj usluge oglašavaju svoja svojstva spajatelju. Svaki oglas sadrži spojne točke (*engl. ports*) koje odgovaraju ponudi i potražnji. Posao spajatelja je slaganje različitih oglasa u jednu cjelinu pri čemu sve spojne točke moraju biti povezane. Nakon određivanja cjeline, spajatelj svim podnositeljima zahtjeva i pružateljima usluge šalje informaciju o načinu spajanja. Podnositelji zahtjeva prema dobivenoj informaciji samostalno kontaktiraju odgovarajuće pružatelje usluga. U slučaju da su pružatelji usluga u međuvremenu promijenili stanje raspoloživosti resursa cijeli proces oglašavanja se ponavlja.

Usko grlo postupka grupnog spajanja je pronalaženje veza između spojnih točaka. Pronalaženje veza obavlja se algoritmom koji rekurzivno prolazi spojnim točkama i pokušava naći kandidate za uspješno spajanje. Za svaki uspješno spojeni oglas potencijalno se povećava broj slobodnih spojnih točaka čije se daljnje spajanje provodi istim postupkom. Eliminacija uskog grla i uzastopnog pretraživanja slobodnih spojnih točaka i odgovarajućih oglasa provodi se indeksnim stablima koja smanjuju broj potencijalno dobrih spojnih točaka u svakom stupnju rekurzije.

Genetski algoritmi se uglavnom koriste kod statičkog određivanja rasporeda pri čemu trajanje postupka nije od kritične važnosti. Genetski algoritam s ograničenjem veličine ulaznog problema (GAOVU) [47] oblikovan je sa ciljem raspoređivanja poslova na grozdu računala. Temeljni koraci genetskog algoritma [48] uključuju stvaranje inicijalne populacije, selekciju, križanje i mutaciju, pri čemu se selekcija, križanje i mutacija odvijaju u definiranom broju koraka ili dok dobrota (*engl. fitness*) najbolje jedinke u populaciji ne dostigne traženu vrijednost.

Svaka jedinka populacije predstavlja jedan mogući raspored poslova u sustavu pri čemu su jedinke kodirane tako da su brojevi poslova poredani u nizu i omeđeni graničnicima. Poslovi između dva graničnika izvršavaju se slijedno na jednom procesoru. Inicijalna populacija odabrana je slučajnim raspoređivanjem dijela poslova na procesore dok je ostatak raspoređen redom na procesore s najmanjim ukupnim opterećenjem. Dobrota jedinke

određena je udaljenošću rasporeda koji je njome definiran od optimalnog rasporeda. Selekcija roditelja se odvija postupkom ruleta gdje vjerojatnost odabira jedinke ovisi o njenoj dobroti. Postupkom cikličkog križanja dvaju roditelja nastaju dvije nove jedinke. Kod mutacije neke jedinke slučajno odabrani dugi posao s najopterećenijeg procesora se mijenja s kratkim poslom nekog drugog procesora.

Postupak raspoređivanja GAOVU se provodi na podskupu poslova u redu čekanja, pri čemu se veličina podskupa odabire s obzirom procjenu vremena raspoloživog do trenutka u kojem će se pojaviti prvi slobodni procesor. Učinkovitost raspoređivanja ovog postupka uspoređena je s postupkom kružnog raspoređivanja poslova (*engl. round robin*) i strategijom dodjeljivanja posla najmanje opterećenom procesoru (*engl. lightest load first*) pri čemu su ostvarena do 50% kraća trajanja rasporeda. Takva mjera učinkovitosti omogućena je u slučaju da su svi poslovi u sustavu poznati unaprijed. S obzirom da se kod GAOVU postupka promatra samo podskup poslova u redu čekanja pretpostavljeno je da se takvim pristupom postiže dobra aproksimacija raspoređivanja na grozdu računala.

3.4.5. Postupak unazadnog popunjavanja praznina uz optimizaciju dinamičkim programiranjem

Postupak unazadnog popunjavanja praznina uz optimizaciju dinamičkim programiranjem (UPPDP) oblikovan je s namjerom iskorištavanja dobrih svojstava AUPP algoritma uz dodatno optimiranje postupka popunjavanja praznina. Kod AUPP postupka rezervacija resursa se provodi za najprioritetniji posao, a praznine koje eventualno nastaju rezervacijom se redom popunjavaju ostalim poslovima pri čemu je redoslijed popunjavanja određen prioritetima poslova. Vrlo je lako konstruirati primjere u kojima poštivanje prioriteta pri popunjavanju praznina dovodi do smanjene iskoristivosti raspoloživih resursa.

Prilikom popunjavanja praznina nastalih rezervacijom najprioritetnijeg posla, moguće je na različite načine kombinirati poslove iz reda čekanja. Postupak UPPDP koji je prikazan na slici 3.9 temelji se na uzastopnom testiranju različitih kombinacija poslova da bi se što bolje popunili raspoloživi resursi. Navedeni se postupak provodi u slučaju dolaska novog posla u sustav ili kod završetka posla. Brisanje postojeće preostale rezervacije posla prethodi ponovnom izračunu rezervacije za najprioritetniji posao. Stvaranje rezervacije provodi se traženjem točke sidrenja na isti načina kao i kod KUPP postupka, pri čemu je taj dio izostavljen sa slike zbog pojednostavljenja prikaza.

Nakon stvaranja rezervacije najprioritetnijeg posla definiraju se liste pripravnih poslova i lista trenutno slobodnih resursa te se poziva postupak *OptimizirajPraznine* kojim se traži najbolji način ispunjavanja trenutno dostupnih resursa. Rekurzivni postupak

```
UPPDP(){
  ObrisiRezervacije()
  p0=najprioritetniji posao iz reda čekanja
  NapraviRezervaciju(p0)
  lr=lista trenutno slobodnih resursa
  lp=lista pripravnih poslova bez posla p0 kojima su dovoljni resursi iz skupa lr
  OptimizajPraznine(1,lp,lr,{},konačniRaspored)
  PokreniPoslove(konačniRaspored)
}

OptimizajPraznine(indeksPosla,lp,lr,privremeniRaspored,konačniRaspored)
  ako(SlobodniResursi(privremeniRaspored)< SlobodniResursi(konačniRaspored)){
    konačniRaspored=privremeniRaspored
  }
  i=indeksPosla
  dok(i<|lp|){
    valjaniResursi={r|(r∈lr)∧(Raspoloživost(r)>=Trajanje(lp(i)))}
    ako(Resursi(lp(i))<=|valjaniResursi|{
      rezerviraniResursi=podskup od valjaniResursi pri čemu se odabiru resursi sa
                         najmanjom raspoloživošću
      privremeniRaspored.Dodaj(lp(i),rezerviraniResursi)
      OptimizirajPraznine(i+1,lp,lr-rezerviraniResursi,privremeniRaspored,
                          konačniRaspored)
      privremeniRaspored.Oduzmi(lp(i),rezerviraniResursi)
    }
    i++
  }
}
```

Slika 3.9 UPPDP postupak raspoređivanja poslova

OptimizirajPraznine redom prolazi listom pripravnih poslova i za svaki posao u listi provjerava da li postoji dovoljno raspoloživih resursa. U slučaju da postoji dovoljno raspoloživih resursa odabire se njihov podskup koji je minimalno potreban za izvođenje posla te se opet rekurzivno ponavlja isti postupak pri čemu se mijenja početni posao i umanjuju dostupni resursi. U svakom pozivu funkcije *OptimizirajPraznine* provjerava se da li trenutno složeni raspored *privremeniRaspored* troši više raspoloživih resursa od do sada poznatog najboljeg rasporeda *konačniRaspored* te se po potrebi trenutno složeni raspored proglašava najboljim rasporedom.

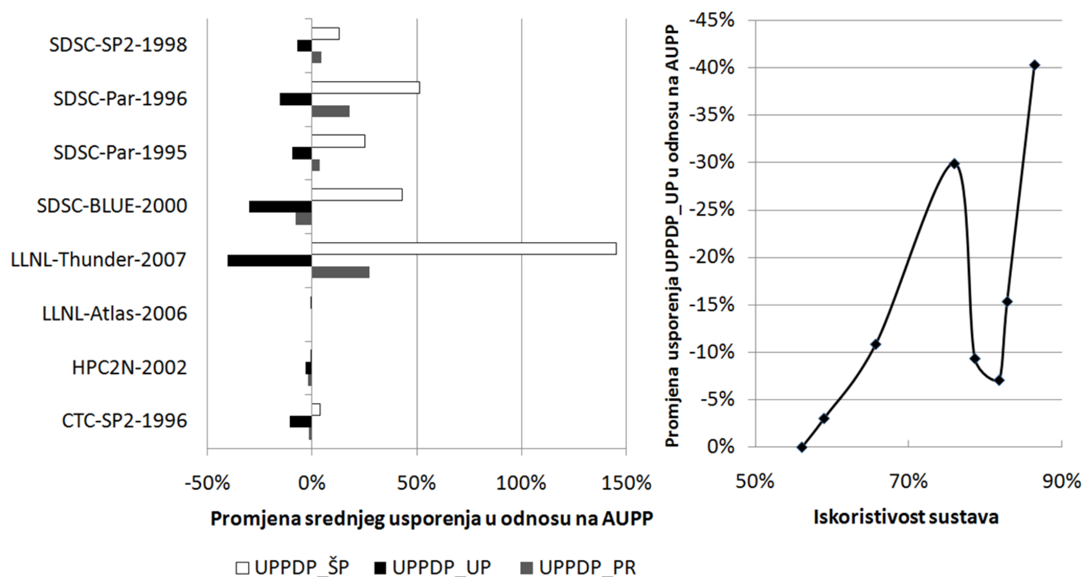
Odabir između dva različita privremena rasporeda koji zauzimaju jednak broj resursa i trajanje UPPDP postupka raspoređivanja poslova su temeljni problemi pri korištenju ovog načina raspoređivanja poslova. Postoji više mogućih načina selekcije između privremenih rasporeda pri čemu se može ciljati veća pravednost ili bolja učinkovitost sustava. Da bi se

omogućio odabir između ova dva cilja izgrađene su tri metode selekcije. Selekcijska metoda UPPDP_PR između dva alternativna rasporeda odabire onaj s prioritetnijim poslovima, dok selekcijske metode UPPDP_UP i UPPDP_ŠP preferiraju poslove uske i široke poslove. Selekcija rasporeda prioritiziranjem uskih poslova trebala bi dovesti do smanjenja srednjeg usporenja sustava, budući da kod nekih grozdova računala postoji korelacija između broja zatraženih resursa i trajanja poslova. Dodatna korist koja se dobiva korištenjem uskih poslova je mogućnost pokretanja više takvih poslova na ograničenoj količini resursa. Motivacija korištenja širokih poslova leži u činjenici da je potencijalno bolje iskoristiti prazninu u sustavu da bi se rasporedio posao koji zahtijeva veću količinu resursa nego čekati da taj posao postane najprioritetniji te da se tada nužnom rezervacijom resursa poveća fragmentacija u sustavu.

Sve navedene selekcijske metode se jednostavno uključuju u opisani postupak tako da se na odgovarajući način sortira lista pripravnih poslova *lp* prije poziva funkcije *OptimizirajPraznine*. Trajanje UPPDP postupka može se ograničiti brojem dozvoljenih rekurzivnih poziva funkcije *OptimizirajPraznine* uz korištenje najboljeg pronađenog rješenja u tako limitiranom vremenu izvođenja. Kod mjerenja učinkovitosti postupka raspoređivanja broj rekurzivnih poziva je ograničen na sto tisuća pri čemu nije zabilježeno značajnije produljenje duljine simulacije u odnosu na AUPP postupak. Do značajnijeg produljenja trajanja postupka raspoređivanja nije došlo zato jer skupovi pripravnih poslova i trenutno slobodnih resursa često sadrže mali broj elemenata, a u slučajevima kada je taj broj značajniji koristi se definirano ograničenje.

Usporedba učinkovitosti UPPDP postupaka s različitim kriterijima selekcije privremenih rasporeda s AUPP postupkom prikazana je na slici 3.10. Postupak UPPDP_ŠP s prioritiziranjem širokih poslova je najmanje učinkovit postupak raspoređivanja poslova, pri čemu je kod računalnog grozda *LLNL-Thunder-2007* srednje usporenje poraslo za 145%. Srednja vrijednost usporenja kod UPPDP_PR postupka je u nekim slučajevima bolja, a u nekim slučajevima lošija od AUPP postupka raspoređivanja poslova. Optimalno slaganje na trenutno slobodnim resursima sa ciljem minimizacije slobodnih resursa ne dovodi nužno do globalno najboljeg rasporeda, budući da je trajanje poslova važan faktor kod stvaranja dobrog rasporeda izvođenja poslova.

Postupak raspoređivanja UPPDP_UP postiže najmanju srednju vrijednost usporenja od svih testiranih postupaka, uključujući i referentni AUPP postupak. Smanjenje srednjeg usporenja se kreće od minimalnih 0,3% na računalnom grozdu *LLNL-Atlas-2006* do maksimalnih 40,3% na računalnom grozdu *LLNL-Thunder-2007*. U pokušaju da se ustanovi uzrok razlikama u promjeni usporenja na desnoj strani slike 3.10 prikazan je odnos



Slika 3.10 Učinkovitost UPPDP postupka raspoređivanja

iskoristivosti mjerenog sustava i promjene srednjeg usporenja UPPDP_UP postupka u odnosu na referentni AUPP postupak raspoređivanja poslova. Sa slike je vidljivo da kod sustava *LLNL-Atlas-2006* i *HPC2N-2002* koji imaju nisku iskoristivost dolazi do manjeg poboljšanja usporenja poslova korištenjem postupka UPPDP_UP. Kod sustava s niskim iskorištenjem ne javlja se intenzivna potreba za popunjavanjem praznina, pri čemu se vrlo često u redu čekanja nalazi vrlo mali skup poslova koji ograničava mogućnosti optimizacije postupka UPPDP_UP.

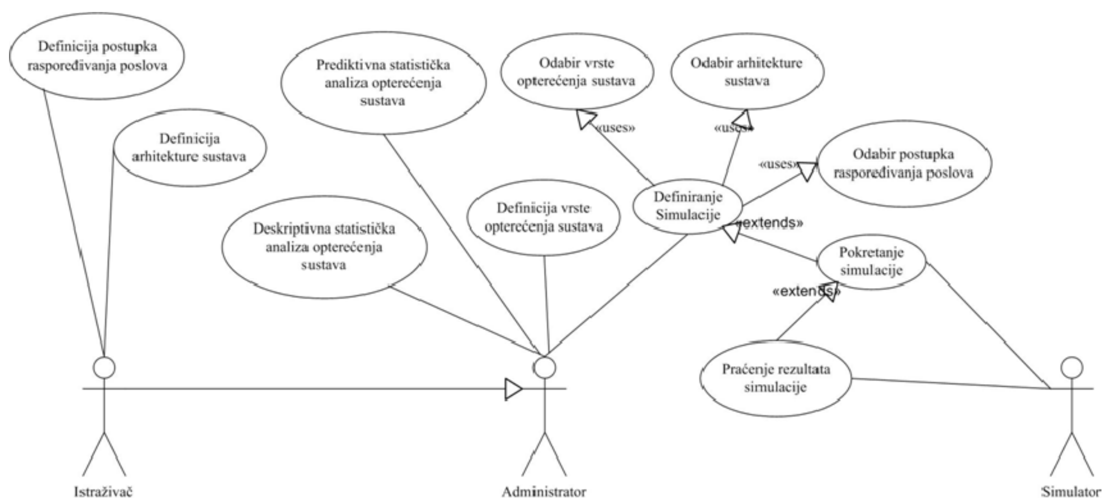
Kod sustava s visokom iskoristivosti postupak UPPDP_UP pokazuje očitu dominaciju nad AUPP postupkom raspoređivanja. Kod nekih računalnih grozdova ta dominacija nije izrazito visoka, a razlog tome leži u vrsti opterećenja računalnog grozda koja kod nekih grozdova ne daje mnogo mogućnosti za optimizaciju slobodnih resursa. Postupak UPPDP_UP učinkovitošću nadmašuje AUPP postupak raspoređivanja poslova uz jednako jamstvo da neće doći do izgladnjivanja poslova. Postupak UPPDP_UP je NP vremenske složenosti s obzirom na broj poslova u redu čekanja za koje postoji dovoljna količina raspoloživih resursa za trenutno izvođenje, ali zbog male veličine tog skupa i mogućnosti ograničenja broja iteracija postupka prikladan je za korištenje na grozdu računala.

4. Sustav za simulaciju i analizu podataka računalnog grozda

Sustav za simulaciju i analizu podataka s računalnog grozda (SARG) oblikovan je kao potpora istraživanju postupaka raspoređivanja poslova na računalnom grozdu. SARG je osmišljen kao proširiva platforma za razvoj diskretnih simulacija opće namjene uz dodatak simulacijskih entiteta nužnih za modeliranje ponašanja grozda računala. Zahtjevi prema kojima je oblikovan SARG sustav navedeni su u odjeljku 4.1. Temeljne komponente arhitekture SARG sustava i njihova međusobna povezanost opisani su u odjeljku 4.2. Simulator koji je najzahtjevnija komponenta SARG sustava te model osnovnih simulacijskih entiteta prikazani su u odjeljku 4.3. U odjeljku 4.4 opisana je komponenta upravitelj simulacija koja kontrolira simulacije čije izvođenje je omogućeno na različitim računalima. Komponente vezane uz prediktivnu statističku analizu koje također čine dio SARG sustava prikazane su u odjeljku 4.5.

4.1. Korisnički zahtjevi pri oblikovanju SARG sustava

SARG sustav [49] prvenstveno je namijenjen istraživačima postupaka raspoređivanja na računalnom grozdu i administratorima računalnih grozdova. Dijagram obrazaca uporabe [50] koji opisuje uloge i načine korištenja SARG sustava prikazan je na slici 4.1. Administrator grozda računala može koristiti SARG sustav da bi povećao učinkovitost raspoređivanja poslova na grozdu računala kojim upravlja. Simuliranje raspoređivanja poslova na grozdu računala sastoji se od tri koraka. U prvom koraku se definira simulacija, odnosno odabere se opterećenje sustava koje će se simulirati, odabire se arhitektura simuliranog grozda računala i postupak raspoređivanja poslova. Ukoliko administrator sustava nije zadovoljan unaprijed



Slika 4.1 Dijagram obrazaca uporabe SARG sustava

definiranim vrstama opterećenja sustava, omogućeno mu je definiranje vlastite vrste opterećenja sustava. Pokretanje simulacije opcionalno je omogućeno za svaku definiranu simulaciju, s time da je praćenje rezultata moguće ukoliko je simulacija pokrenuta. Deskriptiva statistička analiza opterećenja grozdova računala nužna je za razumijevanje ponašanja sustava te može služiti kao osnova za podešavanje različitih parametara raspoređivača poslova. Prediktivna statistička analiza podataka omogućava procjenu mogućnosti predviđanja događaja i parametara poslova u sustavu. Ukoliko se prediktivna statistička analiza pokaže uspješnom na zadanom problemu moguće ju je ugraditi u postupke raspoređivanja poslova.

Temeljna razlika između administratora i istraživača je u mogućnosti definiranja novih arhitektura sustava i postupaka raspoređivanja poslova. Nije nužno da administratori sustava dodaju nove arhitekture ili nove postupke raspoređivanja budući da su komercijalno dostupni postupci raspoređivanja poslova i najčešće korištene arhitekture grozdova dio SARG sustava. Definicije novih arhitektura sustava, novih postupaka raspoređivanja poslova i novih vrsta opterećenja sustava uključuju programsku nadogradnju sustava, pri čemu se poštivanjem odrednica razvoja novih komponenti omogućava interakcija s ostalim dijelovima sustava.

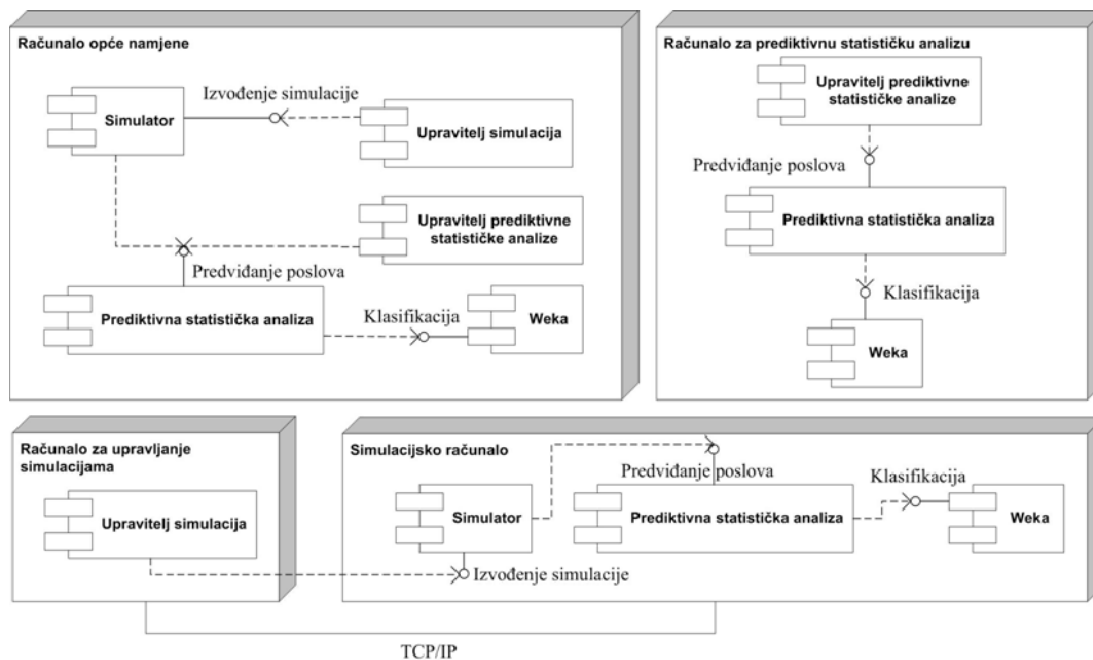
Simulator je zasebni dio SARG sustava koji sudjeluje u pokretanju simulacije te praćenju rezultata za vrijeme i nakon simulacije. Kod istraživanja postupaka raspoređivanja poslova nužne su česte simulacije novih vremenski složenih postupaka koje je potrebno ispravno parametrizirati i isprobati učinak tog postupka na različitim vrstama opterećenja. Da bi se veliki broj simulacija uspješno proveo u prihvatljivom vremenu potrebno je omogućiti paralelno izvođenje simulacija. SARG sustav omogućava paralelno izvođenje više simulacija na grozdu računala, pri čemu se svaka simulacija izvršava sekvencijalno.

Budući da operacijski sustav većine grozdova računala čini neka od inačica Linux-a, potrebno je omogućiti izvođenje simulacija na tom operacijskom sustavu.

4.2. Arhitektura SARG sustava

Kod oblikovanja arhitekture sustava nužno je voditi računa o korisničkim zahtjevima, učinkovitosti i iskoristivosti sustava, jednostavnosti implementacije te mogućem održavanju sustava. Arhitektura SARG sustava koja je oblikovana uz poštivanje korisničkih zahtjeva prikazana je na slici 4.2.

SARG sustav sastoji se od pet temeljnih komponenti koje se u različitim kombinacijama mogu izvršavati na jednom ili više računala. Temeljna komponenta sustava je *Simulator* koji omogućava izvedbu jedne simulacije računalnog grozda s obzirom na zadano opterećenje. U



Slika 4.2 Arhitektura SARG sustava

procesu simulacije dozvoljeno je korištenje prediktivne statističke analize koja je implementirana u komponenti *Prediktivna statistička analiza*. Komponenta *Prediktivna statistička analiza* u osnovi predstavlja sučelje prema Weka [51] programskom alatu za statističku dubinsku analizu podataka.

Upravitelj prediktivne statističke analize je komponenta koja na temelju podataka o opterećenju grozda računala vrši analizu i usporedbu različitih metoda predikcije različitih značajki poslova.

Postoji nekoliko mogućih razmještaja komponenti koje čine SARG sustav s obzirom na raspoloživost resursa i obrazac uporabe. Sve komponente i simulacije moguće je provoditi na samo jednom računalu (*Računalo opće namjene*). Istraživanja vezana uz prediktivnu učinkovitost različitih klasifikatora nezavisna su od simuliranja grozda računala te se dozvoljava njihovo izvođenje na zasebnom računalu. Budući da je simuliranje grozdova računala računski najintenzivniji posao, moguće je paralelno izvršavanje nekoliko različitih simulacija na skupu simulacijskih računala. Upravljanje raspodijeljenim sustavom za simuliranje moguće je vršiti s izdvojenog računala (*Računalo za upravljanje simulacijama*) koje je računalnom mrežom spojeno sa simulacijskim računalima.

SARG sustav ostvaren je u C# programskom jeziku na Microsoft .Net platformi [83]. Povezivanje komponenti SARG sustava s *Weka* komponentom oblikovanoj u Javi ostvareno je pomoću IKVM.NET [84] implementacije Jave za .Net okolinu. Izvršavanje SARG sustava na Linux operacijskom sustavu omogućeno je Mono [85] implementacijom .Net okoline.

Iako se sve komponente SARG sustava mogu izvoditi na Linux operacijskom sustavu, preporuča se da se komponenta *Upravitelj Simulacija* izvodi na računalu sa Windows operacijskim sustavom budući da su primijećene manjkavosti u Mono implementaciji GUI komponenti.

4.3. Simulator grozda računala

Simulator grozda računala je temeljna komponenta SARG sustava. Simulatorom grozda računala je moguće upravljati pomoću upravitelja simulacijama, a njegova se funkcionalnost proširuje pomoću alata za statističku dubinsku analizu podataka. Različiti pristupi diskretnoj simulaciji te razni alati za simulaciju grozda računala opisani su u odjeljku 4.3.1. Arhitektura simulatora grozda računala razvijenog kao dio SARG sustava dana je u odjeljku 4.3.2. Jezgra simulatora koja čini temeljni sloj izgrađenog simulatora opisana je u odjeljku 4.3.3. U odjeljku 4.3.4 prikazani su predlošci simulacijskih entiteta koji ostvaruju osnovno ponašanje temeljnih komponenti grozda računala. Na temelju predložaka simulacijskih entiteta izgrađuju se potpuno funkcionalni simulacijski entiteti koji se koriste u simulaciji grozda računala.

4.3.1. Pristupi simuliranju temeljenom na diskretnim događajima

Simulacija temeljena na diskretnim događajima namijenjena je modeliranju sustava čija funkcionalnost se može prikazati kao niz događaja koji se ostvaruju u kronološki poredanim vremenskim trenucima. Grozd računala se prirodno može modelirati na takav način pri čemu su temeljni događaji u sustavu dolazak novog posla, završetak posla, kvar na nekom od resursa i donošenje odluke raspoređivanja.

Generalno se postupku simuliranja može pristupiti korištenjem gotovih simulacijskih alata, korištenjem programskih biblioteka koje ostvaruju funkcionalnost diskretne simulacije i razvojem cjelokupnog simulacijskog sustava. Korištenje gotovih simulacijskih alata omogućuje modeliranje i simuliranje sustava na najvišoj razini apstrakcije. Takav pristup često ograničava količinu dostupnih svojstava simulacijskog sustava, pri čemu je postavljen strogi okvir razvoja modela te je dan ograničen skup temeljnih komponenti koje se mogu koristiti pri modeliranju. Ograničeni skup dostupnih komponentni može otežati modeliranje sustava ukoliko je potrebno simulirati kompleksne interakcije između simuliranih entiteta.

Simulacijske biblioteke jedna su od alternativa za provođenje diskretnih simulacija. Osnovne simulacijske operacije i entiteti dostupni su kroz programsku biblioteku kao podrška procesu modeliranja. Simulacijske biblioteke pružaju veći stupanj kontrole simulacije za razliku od gotovih simulacijskih alata, ali povećavaju složenost izrade

simulacije. Unatoč povećanom stupnju kontrole nad izgradnjom modela, neki od simulacijskih primitiva poput otkazivanja događaja često nisu podržani simulacijskim bibliotekama, a mogu biti potrebni kod izrade modela. Uglavnom se pažljivim modeliranjem može izbjeći korištenje takvih primitiva, ali uz povećanu složenost izgrađenog modela.

Izrada cjelokupnog simulacijskog sustava je vremenski najzahtjevniji pristup diskretnom simuliranju koji omogućava potpunu kontrolu svih aspekata izrade modela. Kod takvog pristupa nužno je detaljno poznavanje temeljnih zahtjeva i razvojnih principa diskretnih simulacijskih sustava.

Postoji nekoliko razvijenih simulacijskih sustava namijenjenih simuliranju različitih aspekata računalnih grozdova i računalnog grida, pri čemu je većina simulatora grozdova računala izvedena kao dio simulatora računalnog grida. Većina grid simulatora poput Optorsim-a [52], SimGrid-a [53], GridSim-a [54], i Bricks-a [55] usmjerena je na simulaciju replikacije podataka i preseljenja procesa između različitih računalnih grozdova pri čemu je zanemarena detaljna simulacija postupka raspoređivanja unutar računalnog grozda.

SimGrid simulator oblikovan je s namjerom izvođenja simulacije računalnog grida na sklopovskoj opremi ekvivalentnoj onoj na računalnom gridu u svrhu točnog predstavljanja interakcija između paralelnih primjenskih programa i vjernog oponašanja ulazno izlaznih operacija. GridSim je najčešće korišteni simulator računalnog grida koji je zasnovan na simjava [56] programskoj biblioteci. Njime je definiran proširivi model alokacije resursa na različitim računalnim grozdovima te su ponuđene implementacije temeljnih strategija raspoređivanja prekidivih i neprekidivih poslova unutar grozda računala.

Beosim [86] simulacijski sustav namijenjen je detaljnom simuliranju računalnog grozda, ali izvorni kod i detalji izvedbe nisu javno dostupni.

Zbog potreba precizne simulacije računalnih grozdova većina istraživača samostalno gradi modele temeljene na nekom od dostupnih simulacijskih sustava ili programskih biblioteka opće namjene. Takvi simulacijski sustavi su ciljani za specifičnu primjenu na konkretnoj metodi raspoređivanja poslova što otežava nepristranu usporedbu različitih postupaka raspoređivanja poslova unutar iste okoline.

Zbog nedostatka jednostavno proširivog simulatora namijenjenog simuliranju računalnih grozdova i potrebe za usporedbom različitih strategija raspoređivanja poslova izgrađen je cjelokupni simulator računalnog grozda temeljen na diskretnim događajima.

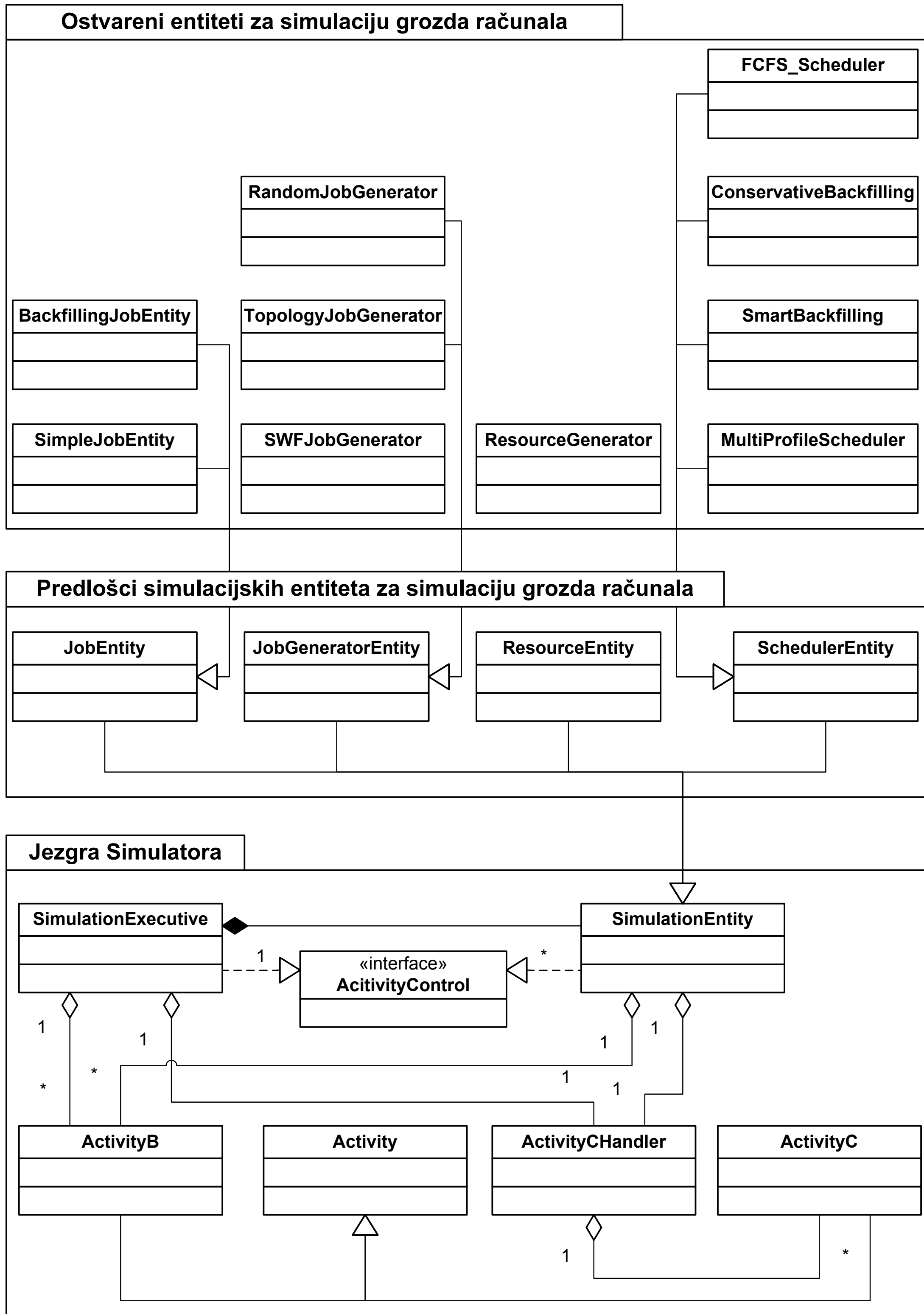
4.3.2. Arhitektura simulatora

Simulator koji čini temeljnu komponentu SARG sustava ostvaren je kao proširiva programska biblioteka namijenjena izvršavanju na .Net platformi. Arhitektura simulatora ostvarena je u više komponenata, pri čemu svaka komponenta pruža određeni skup funkcionalnosti ostalim komponentama. Komponente simulatora sa pripadajućim dijagramima razreda prikazane su na slici 4.3. Iz dijagrama razreda izostavljen je velik broj pomoćnih razreda čiji prikaz nije nužan za razumijevanje funkcionalnosti simulatora.

Komponenta *Jezgra simulatora* ostvaruje temeljnu funkcionalnost upravljanja vremenom i događajima u sustavu. Na temelju tako definirane jezgre simulatora moguće je izgraditi simulaciju temeljenu na diskretnim događajima za proizvoljno područje primjene, a ne isključivo za simulaciju grozda računala. U jezgri simulatora definirani su razredi *SimulationExecutive*, *SimulationEntity*, *Acitivity, ActivityB, ActivityC, ActivityCHandler* i sučelje *ActivityControl*. Razredi *Activity, ActivityB i ActivityC* predstavljaju aktivnosti koje se trebaju izvršiti u određenom trenutku ili kada se ostvare određeni uvjeti u simulaciji. Razred *SimulationEntity* ostvaruje funkcionalnost simulacijskog entiteta koji može generirati određene aktivnosti, pri čemu aktivnosti unutar određenog simulacijskog entiteta trebaju biti vezane uz praćenje ili promjenu stanja tog entiteta. Aktivnosti koje uključuju podatke iz više simulacijskih entiteta ostvarene su zasebno, a za njihovo izvođenje brine se izravno razred *SimulationExecutive*. S obzirom na različitu semantiku aktivnosti, aktivnosti koje se odnose na samo jedan simulacijski entitet sadržane su unutar razreda *SimulationEntity*, a aktivnosti čije izvršavanje uključuje više entiteta dio su razreda *SimulationExecutive*. Klasifikacija aktivnosti i način njihove obrade detaljnije su opisani u odjeljku 4.3.3. Transparentnost dodavanja novih aktivnosti, te njihovog brisanja neovisno o njihovoj povezanosti sa razredima *SimulationEntity* i *SimulationExecutive* osigurana je sučeljem *ActivityControl* koje implementiraju oba navedena razreda.

Predlošci osnovnih simulacijskih entiteta izdvojeni su kao zasebna komponenta simulatora koja čini osnovu simulacije grozda računala. Predlošci se sastoje od minimalnog skupa razreda nužnih za simulaciju grozda računala poput računalnih resursa (razred *ResourceEntity*), poslova (razred *JobEntity*), generatora poslova (razred *JobGeneratorEntity*) i raspoređivača poslova (razred *SchedulerEntity*). U predlošcima osnovnih simulacijskih entiteta nije ostvarena njihova potpuna funkcionalnost, nego su definirane temeljne aktivnosti koje provodi određena vrsta entiteta te događaji za koje treba ostvariti određene akcije.

Na temelju predložaka simulacijskih entiteta izgrađene su detaljne implementacije devet različitih postupaka raspoređivanja poslova, slučajni, XML i SWF (*engl. Standard Workload*



Slika 4.3 Arhitektura simulatora

Format) generator poslova te posao i procesor. Devet različitih postupaka raspoređivanja poslova ostvareno je parametrizacijom razreda *FCFS_Scheduler*, *ConservativeBackfilling*,

SmartBackfilling i *MultiProfileScheduler*. Dva temeljna tipa posla koji se mogu raspoređivati implementirana su razredima *SimpleJobEntity* i *BackfillingJobEntity* pri čemu razred *SimpleJobEntity* generira samo jednu aktivnost vezanu uz završetak posla, dok razred *BackfillingJobEntity* ostvaruje poslove koji generiraju dvije aktivnosti koje uključuju završetak posla i istek korisničke procjene trajanja posla.

Prije početka simulacije u sustavu je potrebno stvoriti sve resurse, odnosno odrediti trenutke dolazaka resursa u sustav. Razredom *ResourseGenerator* ostvarena je funkcionalnost inicijalizacije svih resursa u sustavu.

Slučajni generator poslova ostvaren je razredom *RandomJobGenerator* pri čemu su vremena između dolazaka poslova i trajanja poslova preuzeta iz uniformne razdiobe na temelju zadanog intervala. SWF generator poslova ostvaren razredom *SWFJobGenerator* namijenjen je stvaranju poslova prema podacima iz paralelne arhive opterećenja dobivenih na temelju promatranja raznih grozdova računala. Generiranje poslova omogućeno je na temelju opisa opterećenja zadanog u XML formatu pomoću razreda *TopologyJobGenerator*.

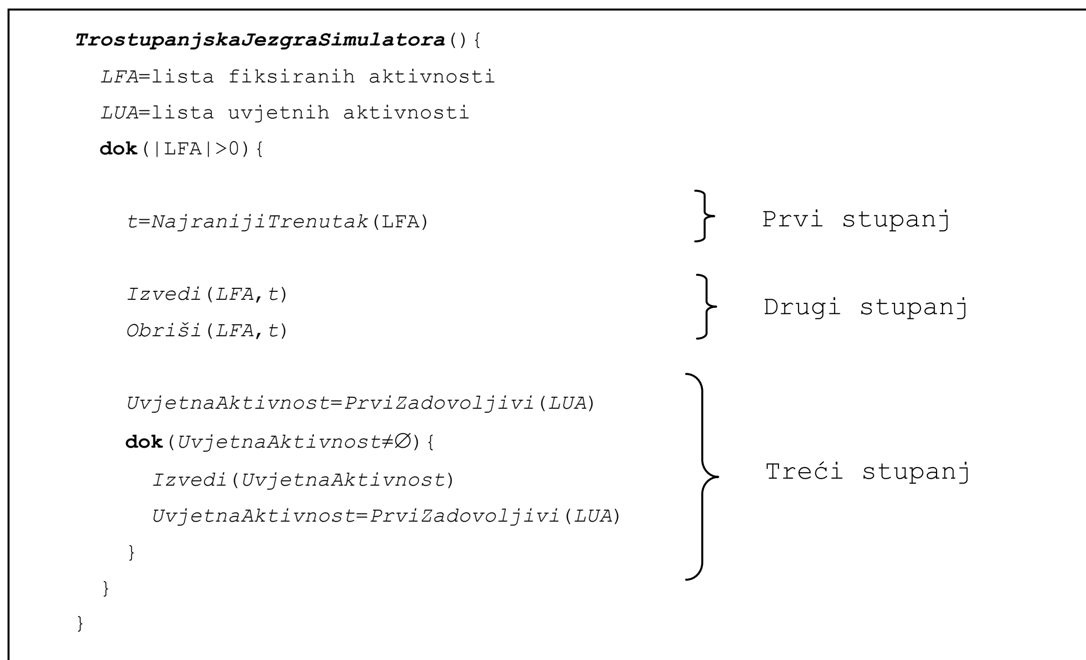
Administratorima računalnih grozdova omogućeno je testiranje različitih postupaka raspoređivanja poslova na temelju zadanog specifičnog opterećenja računalnog grozda oblikovanog prema zahtjevima SWF ili XML generatora poslova. Proširenje simulatora omogućeno je izradom novih entiteta koji se mogu temeljiti na bilo kojem od navedenih razreda, ovisno o specifičnostima modeliranog sustava.

4.3.3. Jezgra simulatora

Jezgra simulatora (*engl. Simulation executive*) kontrolira događaje i aktivnosti koje se provode tijekom simulacije [57]. Postoje različite vrste jezgre simulatora koje se razlikuju s obzirom na kompleksnost jezgre i lakoću izrade simuliranog modela. Odabir vrste jezgre simulatora opisan je u odjeljku 4.3.3.1, a detalji ostvarenja takve jezgre simulatora dani su u odjeljku 4.3.3.2.

4.3.3.1 Odabir vrste jezgre simulatora

Jezgra simulatora zasnovanog na diskretnim događajima prati vremenski tok simulacije uz održavanje popisa trenutaka u kojima je potrebno izvesti određene aktivnosti. Aktivnosti u diskretnim simulacijama se dijele na fiksirane i uvjetne aktivnosti. Fiksirane aktivnosti bezuvjetno se obavljaju u točno definiranim vremenskim trenucima, dok se uvjetne aktivnosti obavljaju samo u trenucima kad su zadovoljeni odgovarajući uvjeti. Izvođenjem fiksiranih aktivnosti mijenja se stanje sustava, koje dovodi do ostvarenja prava na izvođenje uvjetnih aktivnosti. Uvjetne aktivnosti također mijenjaju stanje simuliranog sustava i mogu



Slika 4.4 Trostupanjska izvedba jezgre simulatora

omogućiti ili spriječiti izvođenje drugih uvjetnih aktivnosti. Fiksirane aktivnosti u sustavu su obično vezane uz otpuštanje resursa, dok su uvjetne vezane za potraživanje resursa.

Iako većina simulatora slijedno izvodi sve aktivnosti, neke od njih zakazane su za isti simulirani vremenski trenutak, tako da je potrebno simulacijom ostvariti prividno paralelno izvođenje aktivnosti pomoću sekvencijalnog postupka. Zbog potrebe ostvarenja prividnog paralelizma, razvoj diskretnih modela sličan je razvoju paralelnih programa, zato jer često nema jamstva na redoslijed izvođenja skupa aktivnosti zakazanih u istom trenutku. Zbog postojanja uvjetnih aktivnosti i neodređenog redoslijeda njihova izvođenja pojava potpunog zastoja je jedan od problema simulacije temeljene na diskretnim događajima.

Problem nejednoznačnosti koja može nastati zbog nedefiniranog redoslijeda istovremenih aktivnosti i problem potencijalnog zastoja rješavaju se na različite načine ovisno o vrsti jezgre simulatora. Jezgre simulatora klasificiraju se s obzirom na mogućnost razlikovanja fiksiranih i uvjetnih aktivnosti. Kod jezgri simulatora kod kojih su podržane samo fiksirane aktivnosti, sve potrebe uvjetnog izvođenja ostvarene su kao dio fiksiranih aktivnosti. Takav pristup povećava složenost ostvarenja fiksiranih aktivnosti, budući da na kraju fiksirane aktivnosti mora biti implementirana logika svih uvjetnih aktivnosti za koje postoji mogućnost izvođenja. Takvo smanjenje modularnosti otežava izradu modela, ali pojednostavljuje ostvarenje jezgre simulatora.

Kod izrade simulatora za SARG sustav, jedan od temeljnih zahtjeva je olakšanje izrade modela sustava, pa je izgrađena jezgra simulatora kod koje je podržano izvođenje fiksiranih i

uvjetnih aktivnosti. Jezgra simulatora SARG sustava oblikovana je trostupanjskim postupkom (*engl. three-phase simulation executive*) prikazanim na slici 4.4. U prvom stupnju postupka pronalazi se najraniji vremenski trenutak za koji postoji zakazana fiksirana aktivnost. U drugom stupnju postupka izvode se fiksirane aktivnosti zakazane za taj vremenski trenutak. Nije dozvoljeno da se u toku izvođenja fiksirane aktivnosti generira nova fiksirana aktivnost s istim ili manjim trenutkom izvođenja. U slučaju da se pokuša generirati fiksirana aktivnost s jednakim trenutkom izvođenja, ona se ne zapisuje u listu fiksiranih aktivnosti nego se izvodi kao dio izvorišne aktivnosti. Na kraju drugog stupnja se iz liste fiksiranih aktivnosti brišu sve izvedene aktivnosti.

U trećem stupnju jezgre simulatora u listi uvjetnih aktivnosti se pronalazi prva aktivnost čiji uvjet je zadovoljen i ona se izvodi. Postupak se iterativno ponavlja sve dok se može pronaći uvjetna aktivnost čije je izvođenje dopušteno. Iterativnost postupka je nužna budući da izvođenje jedne uvjetne aktivnosti može promijeniti stanje sustava tako da neka nova uvjetna aktivnost bude omogućena.

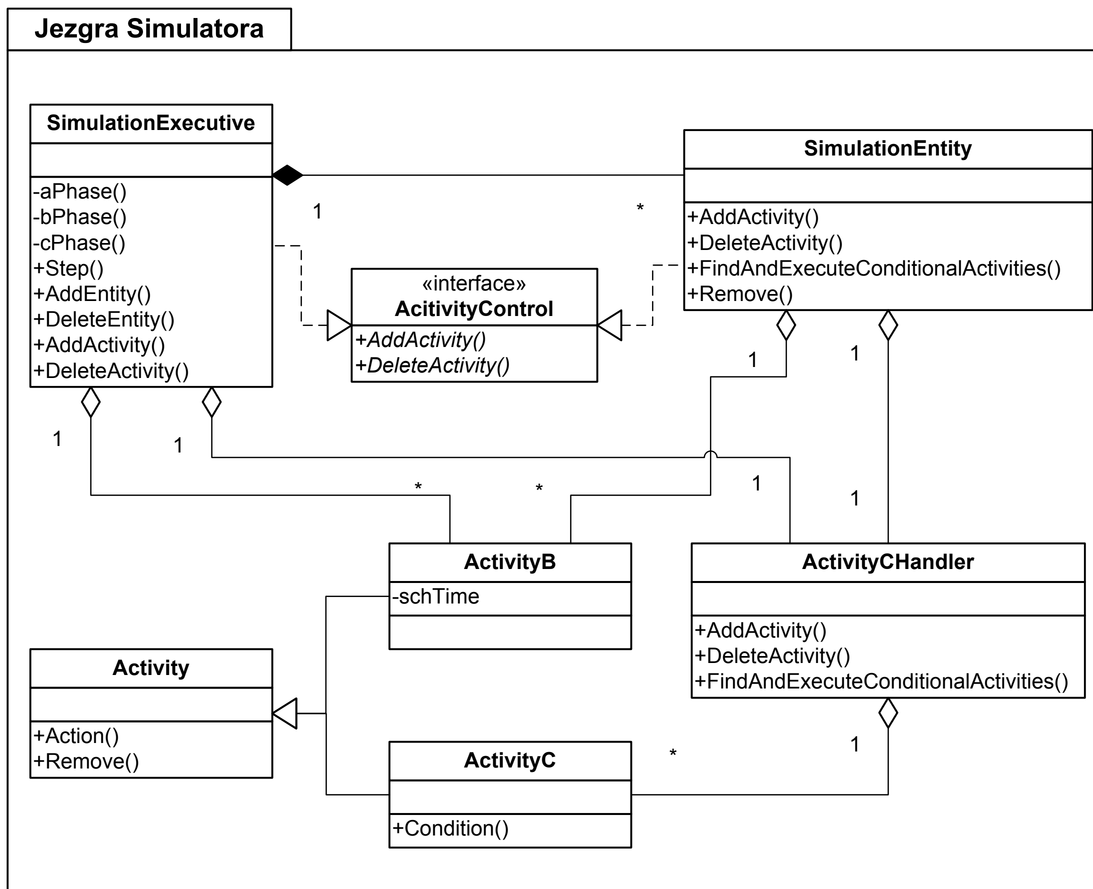
Redoslijed obrade fiksiranih aktivnosti koje se trebaju izvesti u istom trenutku i skupa uvjetnih aktivnosti za čije su izvođenje stvoreni preduvjeti ostvariv je pomoću određivanja prioriteta aktivnosti i sortiranjem odgovarajuće liste aktivnosti. Budući da se sve fiksirane aktivnosti koje obično označavaju otpuštanje resursa provode prije uvjetnih aktivnosti za čije izvođenje su potrebni resursi smanjuje se mogućnost potpunog zastoja, čemu dodatno pogoduje iterativno testiranje uvjetnih aktivnosti.

Kod procesno zasnovane jezgre simulatora (*engl. process-based simulation executive*) također je ponuđeno razlikovanje fiksiranih i uvjetnih aktivnosti, ali na implicitan način. Svaki entitet u simulaciji temeljnoj na procesno zasnovanoj jezgri sastoji se od nekoliko nezavisnih procesa čije izvođenje određuje tijek simulacije. U svakom procesu dozvoljene su temeljne operacije čekanja na određeni vremenski trenutak ili blokirajuće čekanje na ostvarenje nekog uvjeta. Iako je procesno orijentirani pristup modeliranju jednostavniji za učenje, analiza potencijalnog potpunog zastoja i opće razumijevanje stvorenog modela značajno su kompleksniji nego kod trostupanjskog pristupa simuliranju.

Jezgra simulatora SARG sustava ostvarena je trostupanjskim postupkom da bi modelirani entiteti bili što transparentniji sa stanovišta analize simuliranog sustava.

4.3.3.2 Ostvarenje jezgre simulatora

Trostupanjska izvedba jezgre simulatora ostvarena je objektno orijentiranom paradigmom koja prirodno ogovara toj izvedbi [58]. Dijagram razreda jezgre simulatora sa naznačenim najznačajnijim metodama prikazan je na slici 4.5. Razred *SimulationExecutive*



Slika 4.5 Dijagram razreda jezgre simulatora

ostvaruje temeljnu funkcionalnost trostupanjske jezgre simulatora. Metodama *aPhase, bPhase* i *cPhase* redom se ostvaruju stupnjevi jezgre simulatora, pri čemu je izvođenje svih faza moguće pokrenuti metodom *Step* koja vraća broj aktivnosti koje su odrađene u jednom vremenskom trenutku. U slučaju kada poziv funkcije *Step* vrati vrijednost nula simulacija je završena budući da više nije moguće izvršiti niti jednu aktivnost.

Budući da razred *SimulationExecutive* ostvaruje temeljnu funkcionalnost jezgre simulatora, u njemu su sadržani svi entiteti simulacije (razred *SimulationEntity*), aktivnosti koje pripadaju tim simulacijskim entitetima, te aktivnosti koje su nevezane uz simulacijske entitete.

Sve aktivnosti u simulaciji su implementirane razredom *Activity* koji se sastoji od metoda *Action* i *Remove*. Metoda *Action* je apstraktna metoda koju je potrebno ostvariti u nekome od izvedenih razreda što je zadatak osobe koja modelira simulirani sustav. Poziv metode *Action* inicira razred *SimulationExecutive* u slučaju da su ostvareni uvjeti za izvođenje određene aktivnosti. Metodom *Remove* omogućena je deaktivacija aktivnosti, pri čemu se ona uklanja iz njoj pridijeljenog simulacijskog entiteta i jezgre simulatora. Iako brisanje aktivnosti nije

dio standardnih trostupanjskih simulatora, te se semantika brisanja aktivnosti može ostvariti odgovarajućom kombinacijom uvjetnih aktivnosti, u SARG sustavu je omogućeno brisanje aktivnosti da se olakša izrada modela sustava koji se simulira.

Razredi *ActivityB* i *ActivityC* implementiraju fiksirane i uvjetne aktivnosti u modelu. U razredu *ActivityB* koji ostvaruje fiksirane aktivnosti bilježi se samo vrijeme (atribut *schTime*) u kojem je potrebno izvršiti aktivnost, dok je kod razreda *ActivityC* definirana apstraktna metoda *Condition* koju je potrebno ostvariti u nekom od izvedenih razreda ovisno o uvjetima potrebnima u konkretnoj simulaciji. Iako se fiksirane aktivnosti mogu implementirati kao poseban slučaj uvjetnih aktivnosti pri čemu je uvjet vezan samo uz vrijeme izvođenja, u ostvarenju trostupanjskog simulatora one su izdvojene zbog povećanja učinkovitosti jezgre simulatora.

Razred *ActivityCHandler* omogućava grupiranje i izvođenje cijele grupe uvjetnih aktivnosti. Dodavanje i oduzimanje aktivnosti iz grupe omogućeno je metodama *AddActivity* i *DeleteActivity* dok je izvođenje svih aktivnosti iz grupe čiji uvjet je zadovoljen u nekom trenutku simulacije omogućeno metodom *FindAndExecuteConditionalActivities.*

Fiksirane i uvjetne aktivnosti mogu pripadati razredima *SimulationExecutive* i *SimulationEntity,* pri čemu fiksirane aktivnosti (razred *ActivityB*) pripadaju izravno odgovarajućem razredu, a uvjetne aktivnosti (razred *ActivityC*) su tim razredima pridružene posredno preko njihovog manipulatora *ActivityCHandler*. Smještaj aktivnosti u razrede *SimulationExecutive* i *SimulationEntity* ovisi o semantici same aktivnosti. Aktivnosti koje za izvršavanje trebaju podatke iz različitih simulacijskih entiteta ili uzrokuju promjenu različitih simulacijskih entiteta smještaju se u razred *SimulationExecutive*, dok se aktivnosti čije je izvršavanje ograničeno unutar jednog simulacijskog entiteta smještaju u razred *SimulationEntity*, odnosno u njemu izvedeni razred. Ovakva organizacijska aktivnosti čuva svojstvo enkapsulacije objektnih jezika.

Prikazano ostvarenje jezgre simulatora omogućava izvođenje više različitih simulacija u samo jednoj dretvi izvođenja, pri čemu entiteti i aktivnosti mogu pripadati samo jednoj simulaciji, odnosno samo jednoj instanci razreda *SimulationExecutive*.

4.3.4. Predlošci simulacijskih entiteta za simulaciju grozda računala

Identifikacija entiteta i resursa najvažniji je dio modeliranja diskretnih sustava. Entiteti diskretne simulacije su individualni elementi čije ponašanje se simulira i prati posebno za svaki individualni element. Resursi su također zasebni elementi simulacije, ali se njihovo ponašanje ne simulira, nego se promatraju agregatno.

Za svaki element koji sudjeluje u simulaciji odluka se donosi na temelju uloge elementa s obzirom na zadani simulacijski cilj. Temeljni elementi kod simuliranja računalnog grozda su poslovi, računalni resursi i raspoređivač poslova. Dodatno je potrebno ostvariti generator poslova koji omogućava dolazak poslova u različitim vremenskim trenucima. Računalni resursi i poslovi potencijalni su kandidati za modeliranje putem simulacijskih resursa, ali budući da se neka od njihovih ponašanja poput prijave dovršenosti posla i kvara na resursu trebaju zasebno pratiti, računalni resursi i poslovi su modelirani kao simulacijski entiteti.

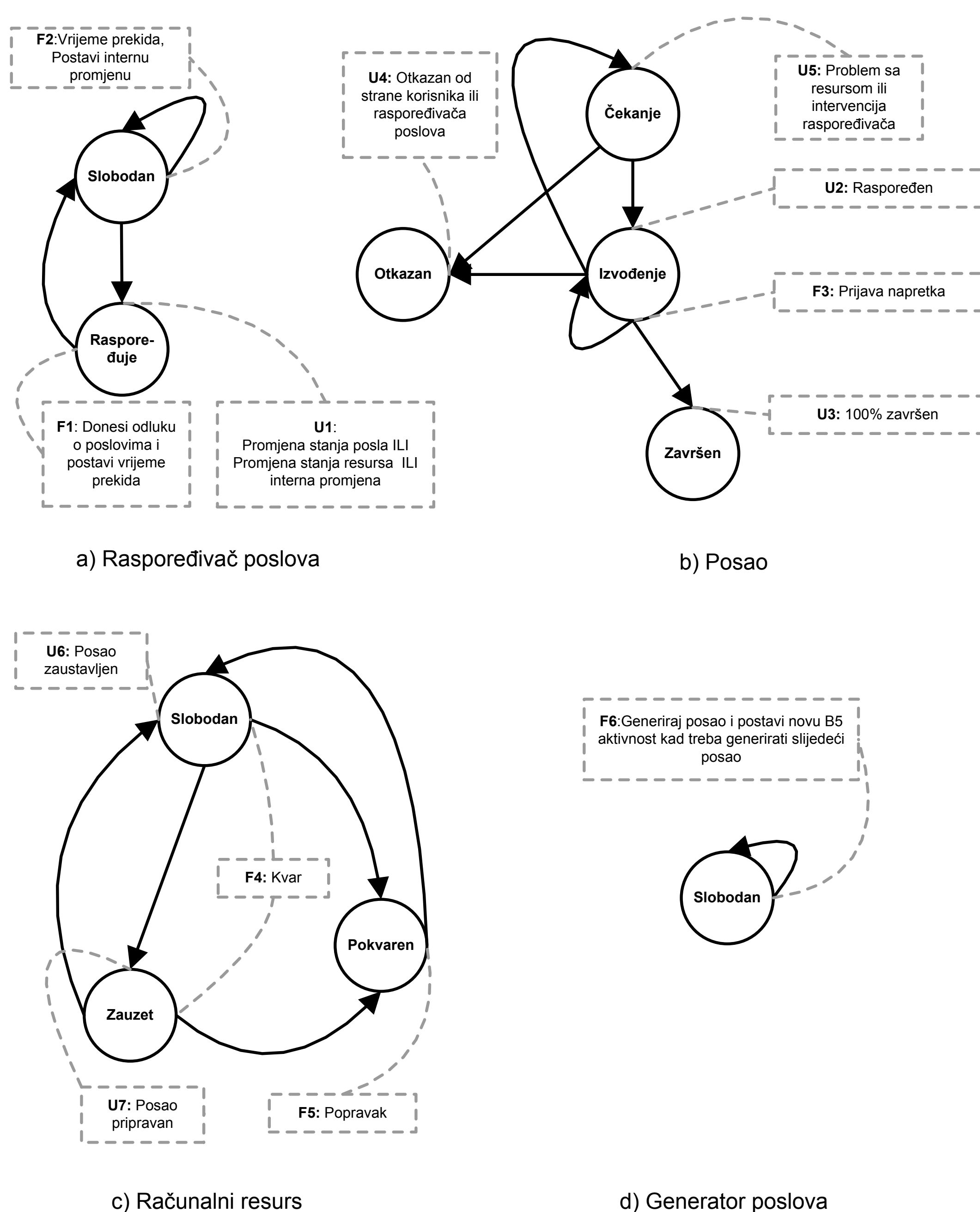
Prikaz funkcionalnosti predložaka simulacijskih entiteta dan je pomoću dijagrama stanja koji opisuju semantiku entiteta, te dijagramima razreda kojima je naznačeno na koji način izvršiti nadogradnju tih predložaka. Dijagrami stanja predložaka simulacijskih entiteta namijenjenih simulaciji grozda računala dani su u odjeljku 4.3.4.1. Ostvarenje predložaka tih simulacijskih entiteta prikazano je u odjeljku 4.3.4.2.

4.3.4.1 Dijagrami stanja predložaka simulacijskih entiteta za simulaciju grozda računala

Različiti entiteti u simulaciji grozda računala mogu se nalaziti u različitim stanjima, pri čemu je u stanjima moguće izvršiti različite aktivnosti koje mogu uzrokovati prijelaze između stanja. Stanja i aktivnosti predložaka simulacijskih entiteta raspoređivača poslova, poslova, računalnih resursa i generatora poslova prikazana su na slici 4.6. Raspoređivač poslova zadužen je za iniciranje slanja podataka na odgovarajuća računala i donošenje odluka o pokretanju poslova. Budući da poslovi u računalnim sustavima mogu biti prekidivi i neprekidivi, raspoređivač poslova je modeliran tako da može raspoređivati obje vrste poslova. Raspoređivač poslova može se nalaziti u stanju *Slobodan* i *Raspoređuje*. Prijelazi između stanja uvjetovani su različitim aktivnostima. Inicijalno se raspoređivač poslova nalazi u stanju *Slobodan*. U slučaju da se promijeni stanje nekog posla u sustavu ili stanje nekog od resursa dolazi do prijelaza iz stanja *Slobodan* u stanje *Raspoređuje* koji je ostvaren uvjetnom aktivnosti *U1*.

Kod većine raspoređivača poslova proces donošenja odluke se smatra trenutačnom aktivnošću, što u dobroj mjeri modelira stvarnost u kojoj je bitno da je brzina donošenja odluka veća od frekvencije dolazaka događaja za koje treba pokrenuti proces raspoređivanja poslova. S obzirom je kod nekih heuristika raspoređivanja, poput genetskih algoritama [47], složenost izvođenja nezanemariva entitet raspoređivača poslova oblikovan je tako da omogući simulaciju trajanja donošenja odluke o raspoređivanju.

Trajanje donošenja odluke modelirano je aktivnošću *F1* pri čemu se u trenutku izvođenja te aktivnosti provodi odluka raspoređivanja donesena za vrijeme izvođenja aktivnosti *U1*. Trajanje odlučivanja, odnosno vremenski razmak između aktivnosti *U1* *i F1* ovisi o vrsti



Slika 4.6 Predlošci simulacijskih entiteta za simulaciju grozda računala

algoritma koja se koristi za raspoređivanje poslova, količini resursa i broju poslova koji se trenutno nalazi u sustavu.

Za simulaciju postupaka raspoređivanja koji imaju izrazito kratko trajanje donošenja odluka s obzirom da frekvenciju događaja u sustavu, oblikovan je pojednostavljeni raspoređivač poslova kod kojeg je uklonjeno stanje *Raspoređuje* i fiksirana aktivnost *F1*. Kod pojednostavljenog raspoređivača poslova donošenje i provedba odluke raspoređivanja provode se za kao sastavni dio aktivnosti *U1*.

Kod simuliranja procesa raspoređivanja prekidivih poslova, interval prekidanja poslova i donošenja odluka raspoređivanja određen je aktivnošću *F2*, za vrijeme koje se bilježi interna promjena stanja raspoređivača poslova. Interna promjena stanja uzrokuje aktivaciju aktivnosti *U1* koja prebacuje raspoređivač poslova u stanje *Raspoređuje*. Nakon provođenja odluke o raspoređivanju poslova za vrijeme izvođenja aktivnosti *F1* dodatno se stvara nova aktivnost *F2* kojom je određen slijedeći trenutak prekidanja svih poslova i donošenja odluke o raspoređivanju. U slučaju da prijelaz u stanje *Raspoređivanje* uzrokovan promjenom stanja posla ili resursa otkazuje se zakazana *F2* aktivnost, donosi se odluka o raspoređivanju poslova te se nakon provođenja odluke o raspoređivanju definira nova aktivnost *F2* čije izvođenje u budućnosti će omogućiti slijedeći ciklus raspoređivanja poslova.

Poslovi su u simulaciji sustava predstavljeni kao zaseban simulacijski entitet koji u sustavu nastaje djelovanjem generatora poslova. Stvoreni posao inicijalno je u stanju čekanja, pri čemu se u trenutku stvaranja novog posla izvodi ciklus donošenja odluke raspoređivača poslova. U slučaju da raspoređivač poslova donese odluku o izvođenju posla, uvjetna aktivnost *U2* mijenja stanje posla u *Izvođenje* i stvara novu *B3* aktivnost koja je nužna za prijavu stupnja dovršenosti posla. Prijava stupnja dovršenosti posla može se implementirati kod nekih poslova, dok se za većinu poslova provodi samo prijava potpune dovršenosti posla. Nakon prijave potpune dovršenosti posla aktivira se uvjetna aktivnost *U3* koja uzrokuje prijelaz iz stanja *Izvođenje* u stanje *Završen*.

Poslovi mogu biti prekinuti ili otkazani iz niza različitih razloga na temelju aktivnosti ostalih entiteta u simulaciji. Prekidanje posla uključuje povratak posla u stanje *Čekanje*, pri čemu uzrok može biti kvar nekog od računalnih resursa ili odluka raspoređivača poslova. Prekidanje posla odvija se uvjetnom aktivnosti *U5* koja provjerava potencijalne uzroke prekida. Prekinuti poslovi mogu se nastaviti od pozicije u kojoj su zaustavljeni ili se mogu pokrenuti iznova, što ovisi o razlogu prekida posla i vrsti posla.

Otkazivanje poslova se razlikuje od prekidanja s mogućnosti nastavka ili ponovnog pokretanja posla, pri čemu se otkazani poslovi ne mogu ponovno pokrenuti. Otkazivanje poslova obično provode korisnici, a u rjeđim slučajevima raspoređivač poslova. Budući da je prijelaz u stanje *Otkazan* uzrokovan drugim simulacijskim entitetima, ostvaren je uvjetnom aktivnošću *U4*.

Računalni resursi u simuliranom sustavu mogu se nalaziti u stanjima *Slobodan*, *Zauzet* i *Pokvaren*. Inicijalno se resursi nalaze u stanju *Slobodan* s aktivnošću *F4* koja definira prijelaz u stanje *Pokvaren* nakon isteka definiranog vremena. Nakon prijelaza računalnog resursa u stanje *Pokvaren* stvara se nova aktivnost *F5* koja definira vrijeme popravka određenog resursa te se njenom aktivacijom računalni resurs vraća u stanje *Slobodan*.

U sustavima u kojima nije potrebno simulirati kvar i popravak, odnosno redovno održavanje, definirani predložak računalnog resursa potrebno je smanjiti izuzimanjem stanja *Pokvaren* i fiksiranih aktivnosti *F4* i *F5*. Prijelaz računalnog resursa iz stanja *Slobodan* u stanje *Zauzet* pomoću aktivnosti *U7* uvjetovan je promjenom stanja posla kojeg raspoređivač poslova šalje na izvođenje. Završetak, odgađanje ili otkazivanje posla dodijeljenog nekom računalnom resursu uvjetuje njegov prijelaz u stanje *Slobodan* što je ostvareno aktivnošću *U6*.

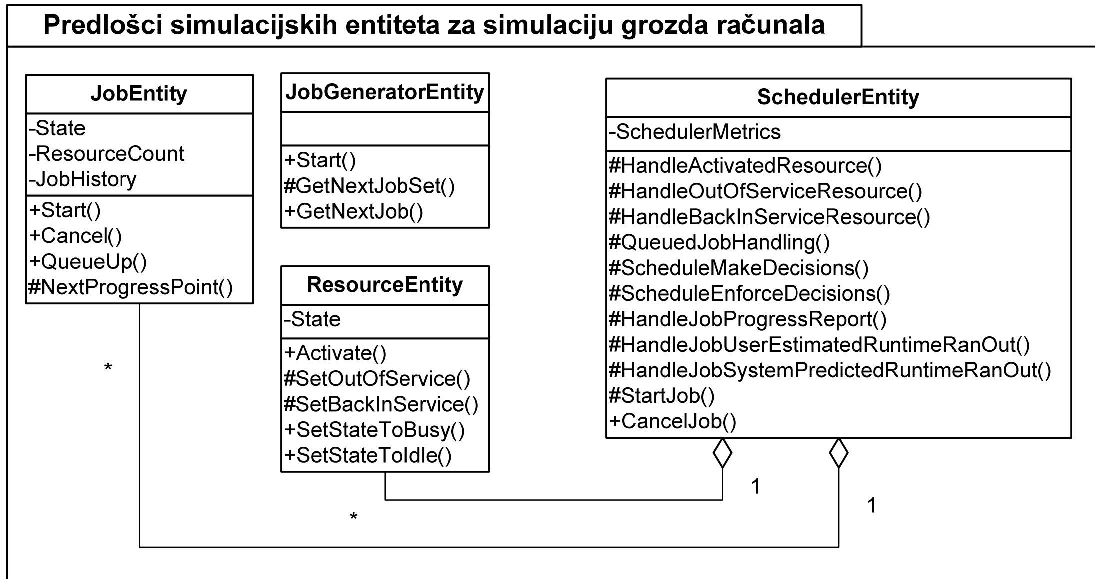
Generator poslova sastoji se od samo jednog stanja *Slobodan* unutar kojega je definirana fiksirana aktivnost *F6*. Fiksirana aktivnost određuje prvi slijedeći trenutak u kojem u simulirani sustav dolaze jedan ili više poslova. Kod izvođenja aktivnosti *F6* stvaraju se novi poslovi koji ulaze u sustav u tom trenutku i zakazuje se nova *F6* aktivnost koja određuje slijedeći vremenski trenutak u kojem će doći novi poslovi u sustav.

Sve navedene aktivnosti nabrojanih predložaka simulacijskih entiteta izvode se prema semantici koja je opisana u ovom odjeljku. Nadogradnja definiranih predložaka omogućena je proširenjem definiranih aktivnosti. Nova strategija raspoređivanja poslova ostvaruje se kroz proširenje aktivnosti *U1* u kojoj je potrebno donijeti odluku o raspoređivanju poslova i aktivnosti *F1* u kojoj se ta odluka provodi.

4.3.4.2 Ostvarenje predložaka simulacijskih entiteta za simulaciju grozda računala

Predlošci simulacijskih entiteta ostvaruju temeljne funkcionalnosti osnovnih elemenata potrebnih za simulaciju grozda računala. Na slici 4.7 prikazan je dijagram razreda predložaka simulacijskih entiteta te su navedene metode kojima su implementirani dijagrami stanja iz prethodnog odjeljka. Dodatno su naznačene metode koje čine osnovu za izgradnju potpune simulacije grozda računala, odnosno čijom implementacijom je moguće ostvariti različite raspoređivače poslova, vrste poslova, generatore poslova te različite tipove resursa.

Razredom *JobEntity* ostvaren je predložak posla na računalnom grozdu. Svaki posao dodjeljuje se jednom raspoređivaču poslova (razred *SchedulerEntity*). Moguće je da u jednom grozdu računala postoji više raspoređivača poslova koji mogu raspoređivati poslove na različitim skupovima resursa, ali nije dozvoljeno da jedan posao bude dodijeljen većem broju raspoređivača poslova. Svaki posao interno bilježi podatke u kojem se stanju nalazi, koliko mu je resursa potrebno, te povijest zbivanja za određeni posao, pri čemu su ti podaci pohranjeni kao atributi *State, ResourceCount* i *JobHistory*. Povijest zbivanja posla uključuje promjenu svih stanja tog posla, zajedno sa popisom vremenskih trenutaka u kojima su se desile promjene stanja.



Slika 4.7 Dijagram razreda predložaka simulacijskih entiteta za simulaciju grozda računala

Generator poslova (razred *JobGeneratorEntity*) stvara poslove u sustavu pri čemu rad započinje pozivom metode *Start*. U svakom ciklusu generacije poslova poziva se funkcija *GetNextJobSet* koja dobavlja skup poslova *P* koji je potrebno generirati u vremenskom trenutku *t*. Nakon dobavljanja tog skupa generira se fiksirana aktivnost koja se zakazuje za trenutak *t* u kojem se stvaraju poslovi iz skupa *P* i započinje novi ciklus dohvata skupa poslova metodom *GetNextJobSet*. Prilikom simulacije stvarnog sustava potrebno je ponuditi implementaciju metode *GetNextJob* koja funkciji *GetNextJobSet* javlja informaciju o slijedećem poslu koji dolazi u sustav.

Generator poslova (razred *JobGenerator* entity) prilikom stvaranja novog posla poziva metodu *QueueUp* razreda *JobEntity*. Ta metoda stvara novu fiksiranu aktivnost čijim izvođenjem se taj posao šalje raspoređivaču poslova. Kad raspoređivač poslova donese odluku o pokretanju poslova poziva metodu *Start* razreda *JobEntity*. U slučaju otkazivanja izvođenja posla raspoređivač poslova poziva funkciju *Cancel* istog razreda.

Stupanj dovršenosti nekog posla, odnosno tok izvođenja posla od njegovog pokretanja kontroliran je pomoću metode *NextProgressPoint*. Metoda *NextProgressPoint* nije ostvarena u razredu *JobEntity*, već je njenu implementacija potrebno izvesti implementacijom novog izvedenog razreda pri čemu je nakon što je prethodni stupanj dovršenosti nekog posla odrađen potrebno odrediti vrijeme i razinu slijedećeg stupnja dovršenosti posla. Na taj način se modeliraju poslovi sa prijavom stupnja dovršenosti i stupnjevanim izvršavanjem. U slučaju gdje se posao izvršava bez stupnjeva potrebno je izvesti jednostavnu metodu *NextProgressPoint* koja vraća ukupno trajanje posla i stupanj dovršenosti od 100%.

Računalni resursi ostvareni su razredom *ResourceEntity*. Aktivacija resursa u sustavu obavlja se pomoću metode *Activate*. Kvarovi i popravak resursa modelirani su funkcijama *SetOutOfService* i *SetBackInService*, pri čemu je definirana osnovna funkcionalnost promjene stanja resursa i prijavka promjene stanja pripadajućem raspoređivaču poslova. Pokretanje poslova, odnosno rezervacija resursa provodi se metodom *SetStateToBusy*, a oslobađanje resursa metodom *SetStateToIdle*.

Raspoređivač poslova, ostvaren razredom *SchedulerEntity*, implementira nekoliko grupa funkcionalnosti koje uključuju manipulaciju resursima i poslovima, donošenje odluka o raspoređivanju poslova te praćenje različitih statistika nužnih za ocjenu učinkovitosti raspoređivača poslova. Manipulacija resursima ostvarena je pomoću funkcija *HandleActivatedResource*, *HandleOutOfServiceResource* i *SetBackInServiceResource* čijom implementacijom je potrebno odrediti ponašanje raspoređivača poslova u trenucima kada određeni računalni resurs postane aktivan te u slučajevima kvara i popravka resursa.

Manipulacija poslovima uključuje prihvat novog posla metodom *QueuedJobHandling* i prijavu dovršenosti posla metodom *HandleJobProgressReport*. Dodatno je za neke vrste poslova omogućena obrada isteka korisničke i sistemski generirane procjene trajanja posla koja se ostvaruje implementacijom predložaka metoda *HandleJobUserEtimatedRuntimeRanOut* i *HandleJobSytemPredictedRuntimRanOut*. Sistemski generirana procjena trajanja posla se dobiva na temelju analize prošlosti sustava.

Svaka od promjena u sustavu koja se odnosi na poslove i resurse automatski dovodi do pokretanja metode *ScheduleMakeDecisions* unutar koje se donose odluke o raspoređivanju, otkazivanju i odgađanju poslova. Budući da proces donošenja odluka može trajati neko vrijeme, provođenje donesenih odluka ostvaruje se u metodi *ScheduleEnforceDecisions*. Trajanje procesa odlučivanja jedna je od izlaznih vrijednosti metode *ScheduleMakeDecisions.* Pokretanje poslova može inicirati samo raspoređivač poslova metodom *StartJob*, dok se otkazivanje poslova metodom *CancelJob* može provesti iz bilo kojeg razreda.

Budući da razred *SchedulerEntity* implementira predložak raspoređivača poslova, metode koje su navedene podržavaju samo osnovne mehanizme koje trebaju podržati svi raspoređivači poslova poput automatskog poziva metode *ScheduleMakeDecisions* kada dođe do promjene u sustavu i poziva funkcije *HandleJobProgressReport* kada neki posao prijavi status dovršenosti. Razred *SchedulerEntity* dodatno generira nekoliko temeljnih događaja na koje se ostali razredi mogu pretplatiti u svrhu nadziranja simulacije u toku njenog izvođenja. Temeljni događaji uključuju aktivaciju novog resursa, pokretanje posla na nekom skupu resursa, završetak i otkazivanje posla.

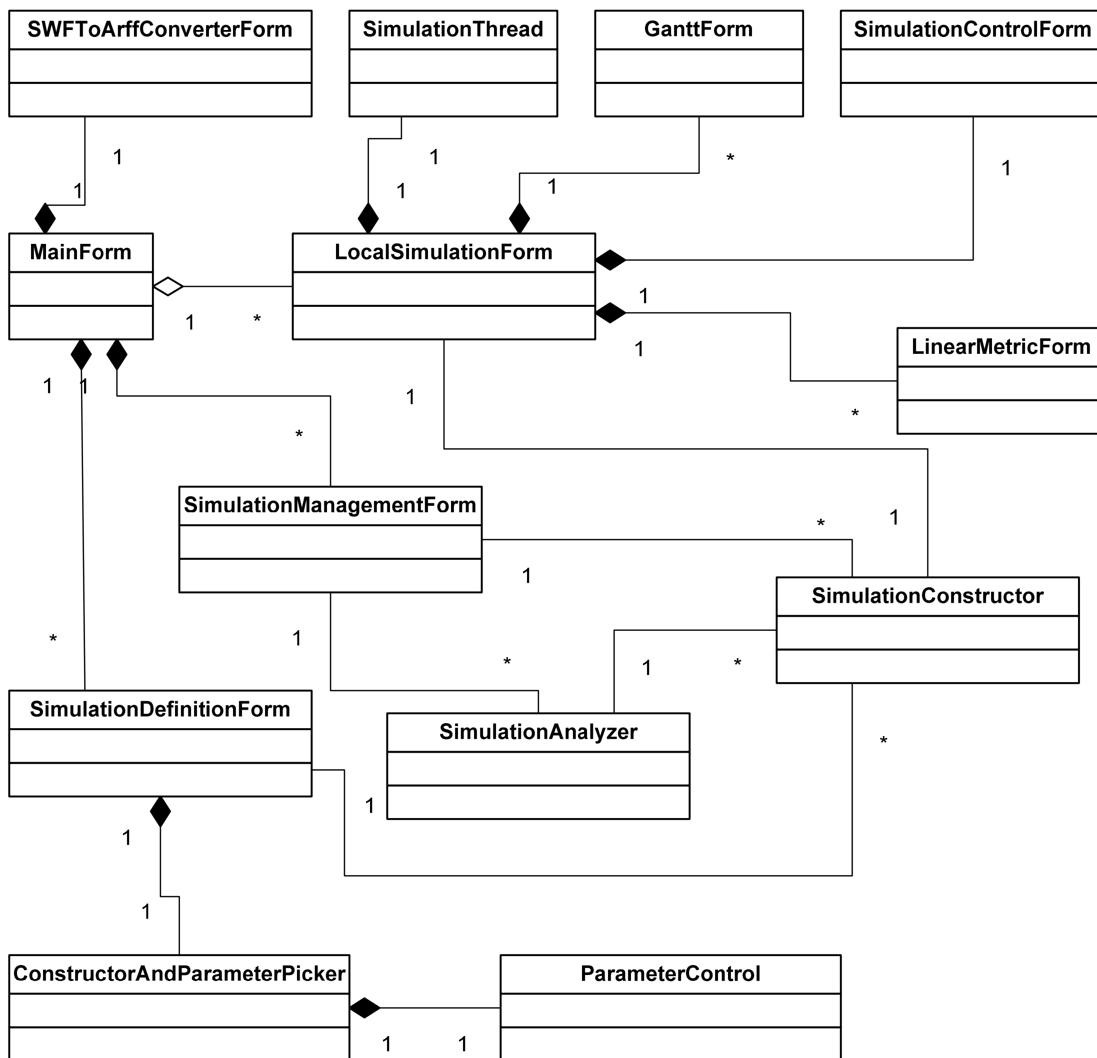
Unutar razreda predložaka simulacijskih entiteta za simulaciju grozda računala implementirani su podrazredi koji ostvaruju odgovarajuće fiksirane i uvjetne aktivnosti. Izvođenje tih aktivnosti uvjetuje pokretanje metoda razreda kroz vrijeme, pri čemu se uzima u obzir i trenutno stanje sustava. Podrazredi nisu prikazani na slici 4.7 zbog pojednostavljenja prikaza. Detaljnija dokumentacija razreda dostupna je u dodatku B.

4.4. Upravitelj simulacija

Upravitelj simulacija je komponenta SARG sustava koja omogućava kontrolu nad većim brojem simulacija koje se mogu izvoditi na različitim računalima, pri čemu je udaljeno izvođenje simulacija transparentno za korisnika sustava. Dodatno upravitelj simulacija podržava usporedbe različitih parametara i rezultata završenih simulacija, kao i detaljno praćenje svakog koraka svih simulacija koje se izvode na istom računalu kao i upravitelj simulacija.

Detaljni prikaz arhitekture upravitelja simulacija dan je na slici 4.8. Razred *MainForm* ostvaruje početni prikaz upravitelja simulacije i omogućava odabir svih funkcionalnosti koje su podržane upraviteljem simulacija. Razred *SimulationDefinitionForm* omogućava definiranje novog skupa simulacija temeljem odabira različitih postupaka raspoređivanja poslova, različitih generatora poslova te različite vrste računalnih resursa. Kod definiranja novog skupa simulacija automatski su ponuđene sve implementacije raspoređivača poslova (razred *SchedulerEntity*), generatora poslova (razred *JobGeneratorEntity*) i generatora resursa (razred *ResourceGeneretor*). Parametriziranje postupaka omogućeno je odabirom konstruktora odgovarajućeg razreda, te odabirom skupa vrijednosti iz kojih treba popuniti određene parametre konstruktora. Podrška odabiru konstruktora i odgovarajućih parametara ostvarena je u razredima *ConstructorAndParameterPicker* i *ParameterControl*.

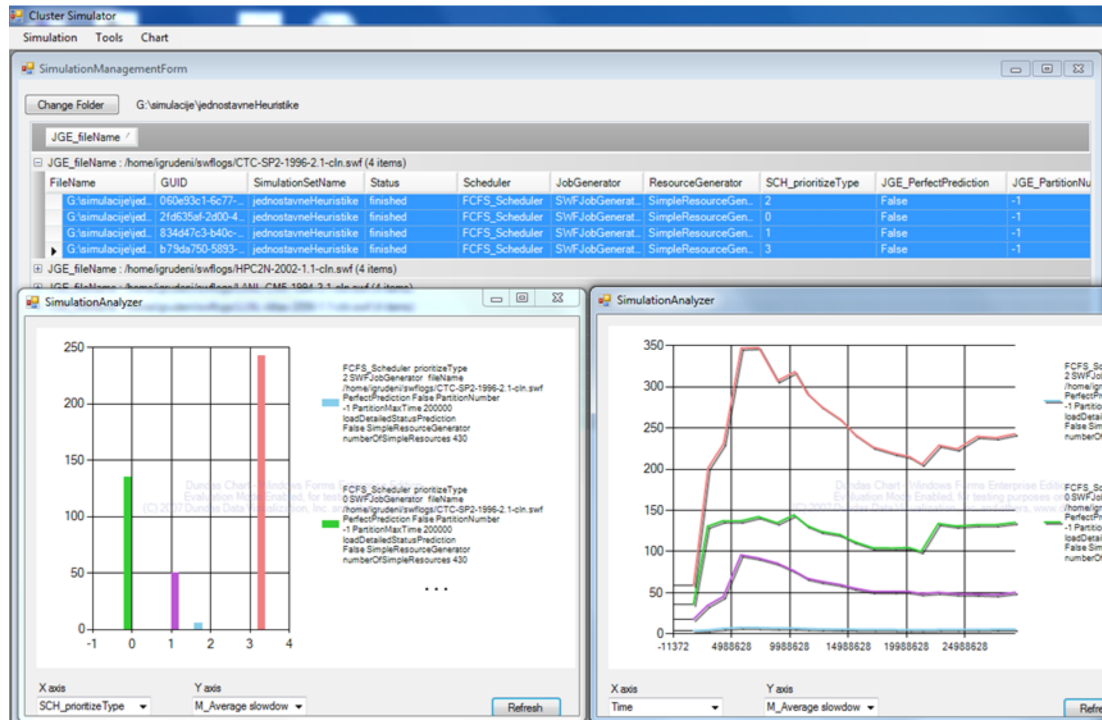
Rezultat definicije simulacije je skup objekata razreda *SimulationConstructor* koji objedinjuje sve podatke potrebne za izvođenje simulacije. Razred *SimulationConstructor* podržava serijalizaciju podataka u datoteku i taj se proces provodi nakon definiranja novih simulacija. Jednom stvorenim simulacijama upravlja se pomoću razreda *SimulationManagementForm* koji daje prikaz svih definiranih simulacija i njihovih statusa. Simulacija grozda računala može biti inicijalizirana, pokrenuta na lokalnom računalu, pripravna u redu čekanja udaljenog računala, pokrenuta na udaljenom računalu i završena. Razred *SimulationManagementForm* omogućava kontrolu izvođenja simulacija na raznim računalima te omogućava usporedbu rezultata simulacija u tekstualnom obliku. Vizualna usporedba rezultata skupa simulacija omogućena je razredom *SimulationAnalyzer*.



Slika 4.8 Dijagram razreda upravitelja simulacija

Izvođenje simulacija na istom računalu na kojem se izvodi upravitelj simulacija (lokalne simulacije) omogućeno je razredom *LocalSimulationForm* i pripadajućim pridruženim razredima. Razred *SimulationThread* vrši kontrolu izvođenja dretve u kojoj se izvodi simulacija. Kontrola nad izvođenjem simulacije korisniku je pružena putem grafičke komponente ostvarene razredom *SimulationControlForm*. Grafički prikaz izvođenja lokalne simulacije omogućen je razredima *GanttForm* i *LinearMetricForm*, pri čemu razred *GanttForm* daje prikaz zauzeća resursa od strane poslova na grozdu računala, a razredom *LinearMetricForm* moguće je pratiti promjenu različitih mjera učinkovitosti raspoređivača poslova kroz cijeli tok simulacije.

Izvođenje simulacija na udaljenom računalu ne uključuje pokretanje cijelog upravitelja simulacije, već je oblikovan zaseban primjenski program u kojem su izostavljeni svi grafički elementi. Zasebni program sastoji se od dijela funkcionalnosti razreda *LocalSimulationForm*



Slika 4.9 Prikaz upravljanja i analize simulacija pomoću komponente upravitelj simulacija

i razreda *SimulationThread*. Ovakav pristup odabran je zbog problema sa kompatibilnošću grafičkih komponenti na različitim operacijskim sustava, odnosno da se omogući izvođenje simulacije na Linux zasnovanim operacijskim sustavima.

U upravitelju simulacije dodatno je omogućena konverzija podataka koji predstavljaju opterećenje grozda računala iz SWF formata u ARFF format pogodan za analizu podataka u WEKA alatu za dubinsku analizu. Pretvorba podataka omogućena je razredom *SWFToArffConverterForm*.

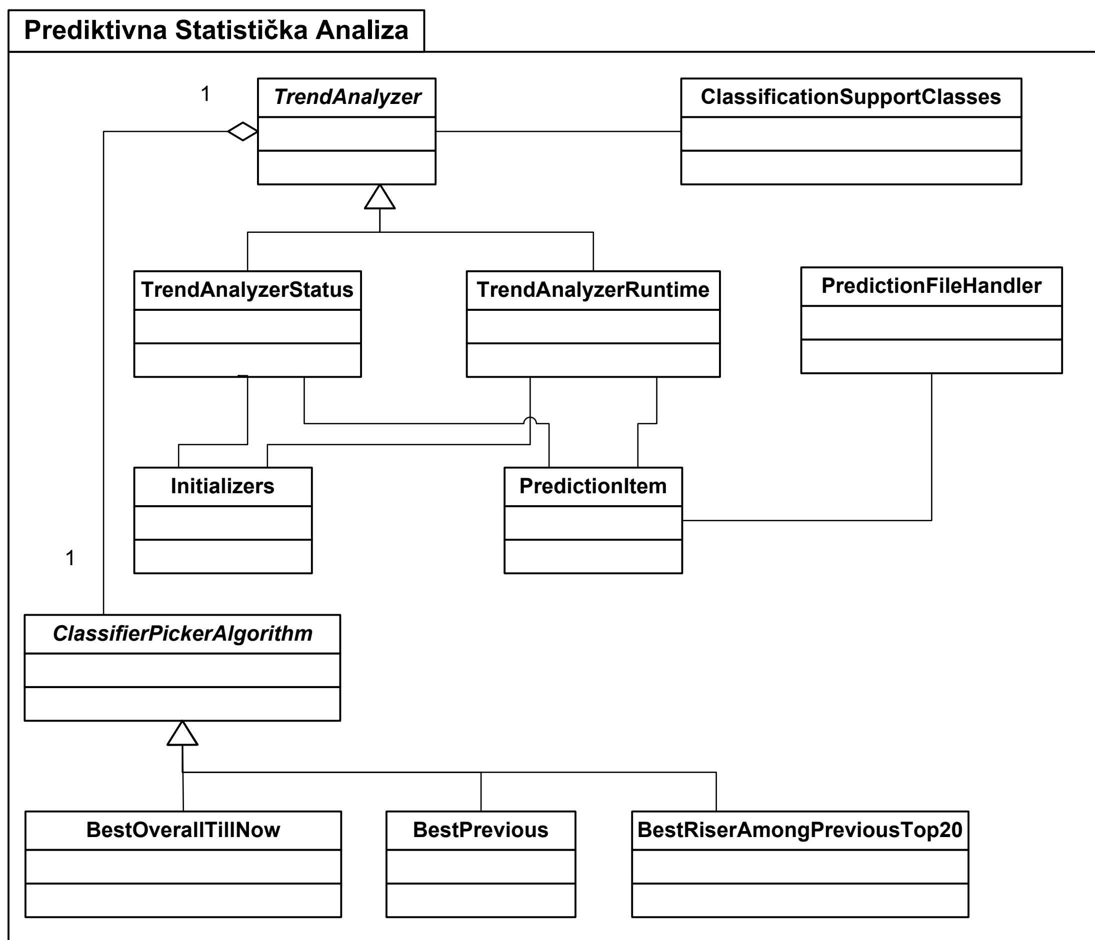
Na slici 4.9 prikazano je grafičko sučelje upravitelja simulacija. U prozoru sa oznakom *SimulationManagementForm* označene su četiri različite simulacije grozda računala. Rezultati tih simulacija uspoređeni su u prozorima sa oznakama *SimulationAnalyzer* koji su implementirani istoimenim razredom. Na temelju usporedbe parametra srednjeg usporenja poslova vidljive su značajne razlike u učinkovitosti uspoređenih postupaka raspoređivanja poslova.

4.5. Komponente vezane uz prediktivnu statističku analizu

U SARG sustavu nalaze se komponente *Prediktivna Statistička Analiza* i *Upravitelj Prediktivne Statističke Analize*. Komponenta *Upravitelj Prediktivne Statističke Analize* ostvarena je kao komandno linijski primjenski program koji poziva različite funkcionalnosti

komponente *Prediktivna Statistička Analiza*. Komponenta *Prediktivna Statistička Analiza* sastoji se od nekoliko razreda u kojima su grupirane statičke metode namijenjene pretvorbi ulaznih podataka, pokretanju odgovarajuće dubinske statističke analize podataka ostvarene u Weka sustavu, interpretaciji rezultata te klasifikaciji podataka o novonastalim poslovima.

Dijagram razreda komponente *Prediktivna Statistička Analiza* prikazan je na slici 4.10. Razred *ClassificationSupportClasses* sadrži postupke nužne za predviđanje novih podataka u vremenskoj seriji koji su zasnovani na metodi pomičnog prozora (*engl. sliding window*), pri čemu se koristi i funkcionalnost Weka sustava. Traženje optimalne veličine pomičnog prozora na temelju testiranja različitih veličina u prošlosti ostvareno je u razredu *TrendAnalyzer,* te u njemu izvedenim razredima *TrendAnalyzerStatus* i *TrendAnalyzerRuntime* ovisno o tome da li se vrši predviđanje statusa ili trajanja poslova. Različite metode odabira optimalne veličine prozora ostvarene su različitim implementacijama razreda *ClassPickerAlgorithm*, koje uključuju razrede *BestOverallTillNow*, *BestPrevious* i *BestRiserAmongPreviousTop20*.



Slika 4.10 Dijagram razreda komponente *Prediktivna Statistička Analiza*

Predikcija statusa i trajanja poslova pohranjena je u razredu *PredictionItem*, pri čemu je, uz njegovo izravno korištenje u simulacijama grozdova računala, moguć i zapis svih predviđanja u datoteku putem razreda *PredictionFileHandler*.

5. Metode za dubinsku analizu podataka o opterećenju

Raspoređivač poslova na grozdovima računala donosi odluke na temelju raspoloživih resursa i dostupnih poslova. Podaci potrebni za raspoređivanje posla uključuju količinu potrebnih resursa i trajanje posla. Na računalnim grozdovima informacija o trajanju posla zasnovana je na korisničkoj procjeni. Dodatno se zbog pogreške u primjenskom programu može desiti prijevremeni prekid posla. Da bi se popravila korisnička procjena trajanja poslova te izolirali slučajevi s visokom vjerojatnošću pogreške koriste se metode za dubinsku analizu podataka. U izradi ovog rada korištena su tri postupka analize koji su opisani u ovom poglavlju. U odjeljku 5.1 prikazan je postupak klasifikacije temeljen na naivnim Bayesovim mrežama. Klasifikacija stablima odlučivanja dana je u odjeljku 5.2, a postupak klasifikacije slučajnim šumama opisan je u odjeljku 5.3.

5.1. Klasifikacija naivnim Bayesovim mrežama

Klasifikacija naivnim Bayesovim mrežama [59] zasnovana je na pretpostavci nezavisnosti među atributima koji opisuju skup podataka. Bayesov klasifikator gradi se na temelju skupa za učenje D koji čine instance za učenje. Instance za učenje sastoje se od jednodimenzionalnog vektora atributa $X=(x_1,x_2...,x_n)$ i njima dodijeljenog ciljnog atributa. Elementi vektora X označavaju poznate ili mjerene vrijednosti atributa A_1, A_2, ... , A_n. Ciljni atribut A_c može poprimiti jednu od vrijednosti iz skupa $\{C_1, C_2, \ldots, C_m\}$ koje se nazivaju klasama.

Klasifikacija naivnim Bayesovim mrežama zasniva se na traženju klase C_i koja ima najveću vjerojatnost pojavljivanja s obzirom na poznate podatke iz vektora X. Vjerojatnost pojavljivanja klase C_i uz poznate vrijednosti vektora X određena je Bayesovim teoremom:

$$P(C_i \mid X) = \frac{P(X \mid C_i) \cdot P(C_i)}{P(X)}$$

Kod pronalaska najveće vrijednosti izraza $P(C_i \mid X)$, potrebno je maksimizirati samo produkt $P(X \mid C_i) \cdot P(C_i)$ budući da je vjerojatnost $P(X)$ konstanta neovisna o klasi C_i. U slučaju da su sve klase C_i jednako vjerojatne problem klasifikacije svodi se na maksimizaciju izraza $P(X \mid C_i)$. Ukoliko nisu poznate vjerojatnosti pripadanja klasama $P(C_i)$ one se mogu procijeniti izrazom:

$$P(C_i) = \frac{|C_{i,D}|}{|D|}$$

gdje $|C_{i,D}|$ označava broj instanci klase C_i u skupu za učenje D.

Smanjenje složenosti izračuna izraza $P(X \mid C_i)$ omogućeno je pretpostavkom o nezavisnosti atributa $A_1..A_n$. Uz pretpostavljenu nezavisnost atributa vrijednost $P(X \mid C_i)$ određena je sljedećim izrazom:

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

Pojedinačne vrijednosti $P(x_k \mid C_i)$ izračunavaju se s obzirom na vrstu atributa A_k. Za kategorijski atribut A_k izraz $P(x_k \mid C_i)$ jednak je broju pojavljivanja vrijednosti x_k kod instance klase C_i u skupu D podijeljenom sa $|C_{i,D}|$. U slučaju da A_k sadrži kontinuirane vrijednosti potrebno je na temelju distribucije tih vrijednosti za klasu C_i, procijeniti vjerojatnost pojavljivanja x_k.

5.2. Klasifikacija stablima odlučivanja

Postupci klasifikacije stablima odlučivanja zasnovani su na izgradnji hijerarhijskog dijagrama odlučivanja. Svaki čvor u stablu na temelju vrijednosti atributa instance određuje klasu instance ili određuje podstablo unutar kojeg će se donijeti odluka o klasi instance.

Izgradnja stabla odlučivanja temelji se na rekurzivnom postupku [59] prikazanom na slici 5.1. Ulazni podaci postupka su skup instanci za učenje D i popis svih atributa. Na

```
GenerirajStablo(D,lista_atributa){
  Stvori čvor N
  ako(sve instance skupa D pripadaju istoj klasi C)
    Vrati N kao list stabla označen klasom C
  ako(|lista_atributa|=0)
    Vrati N kao list stabla označen klasom koja je najzastupljenija u D
  Odaberi atribut A s najvećom informacijskom vrijednosti i odredi kriterij_podjele
  Označi čvor N sa kriterij_podjele
  ako(A je kategorijski ∧ dozvoljena je višestruka podjela)
    lista_atributa=lista_atributa-A
  za(svaki ishod j od kriterij_podjele)radi{
    Dj=skup instanci od D koji zadovoljavaju j
    ako(|Dj|=0)
      Dodaj list čvoru N označen s klasom koja je većinski zastupljena u D
    inače
      GenerirajStablo(Dj)
  }
}
```

Slika 5.1 Postupak izgradnje stabla odlučivanja

početku postupka se stvara novi čvor *N*, koji se u slučaju da sve instance skupa *D* pripadaju istoj klasi *C* označava tom klasom te postupak završava. Ukoliko sve instance skupa *D* ne pripadaju istoj klasi odabire se atribut *A* koji nosi najveću informacijsku vrijednost i na temelju kojeg se definira kriterij podjele instanci koji se sastoji od više mogućih ishoda. U slučaju da je *A* kategorijski atribut i da svaki čvor stabla može imati više djece, kriterijem podjele se definiraju svi mogući ishodi tog atributa te ga se uklanja iz *lista_atributa*.

Algoritam završava dijeljenjem skupa *D* s obzirom na moguće ishode kriterija podjele te se za svaki podskup koji nastane dijeljenjem skupa *D* iznova rekurzivno poziva isti postupak izgradnje stabla. U slučaju da je neki od tako nastalih podskupova skupa *D* prazan skup, stvara se novi list povezan s tekućim čvorom *N* koji se označava klasom većinskom u *D*. Složenost postupka izgradnje stabla odlučivanja jednaka je $O(n \cdot |D| \cdot \log(|D|))$ pri čemu je *n* broj atributa. Ova složenost dobivena je gornjom ogradom $\log(|D|)$ na broj čvorova u stablu i složenosti $n \cdot |D|$ nužnoj za izračun informacijskog dobitka svih atributa na temelju instanci u skupu *D*.

Nakon što se izgradi stablo odlučivanja neke od grana će odražavati anomalije koje se nalaze u ulaznom skupu podataka. Anomalije mogu nastati zbog šuma (*engl. noise*) ili stršećih vrijednosti (*engl. outliers*) u ulaznim podacima. Ukoliko su anomalije nastale zbog stršećih vrijednosti dolazi do prenaučenosti (*engl. overfitting*) stabla odlučivanja. Problem prenaučenosti se rješava kraćenjem (*engl. pruning*) stabla koje se može ostvariti zaustavljanjem postupka u određenim fazama izgradnje stabla, ili se već izgrađeno stablo može naknadno skraćivati.

Različiti postupci izgradnje stabala poput CART [60], C4.5 [61] i ID3 [62] uglavnom se razlikuju u metodama odabira atributa i definiranju kriterija podjele te načina kraćenja stabla. U ovom radu korišten je C4.5 postupak izgradnje stabla.

5.3. Klasifikacija slučajnim šumama

Grupne metode klasifikacije (*engl. ensemble methods*) [63] su algoritmi za učenje koji se sastoje od više individualnih klasifikatora čijom kombinacijom se određuje klasa nove instance. Slučajne šume (*engl. random forests*) [64] su grupna metoda klasifikacije koja se zasniva na glasanju više stabala odlučivanja, pri čemu se kao klasa testne instance odabire ona koja je dobila najviše glasova. Kod izgradnje pojedinačnih stabala odlučivanja, skup za učenje svakog pojedinog stabla dobiven je uzorkovanjem sa zamjenom (*engl. sampling with replacement*) originalnog skupa za učenje pri čemu su veličina uzorkovanog i originalnog skupa jednake.

Izgradnja pojedinačnih stabala odlučivanja neznatno se razlikuje od postupka u prethodnom odjeljku. Kod stvaranja svakog čvora najbolji atribut se ne traži na temelju cijelog skupa atributa, već na temelju slučajno odabranog podskupa svih atributa. Proces skraćivanja stabla se ne provodi što znači da se kod pojedinačnih stabala javlja problem prenaučenosti.

Temeljna dva parametra slučajnih šuma koja se mogu podešavati su broj stabala i veličina podskupa atributa koji se koriste kod izgradnje pojedinačnih stabala. Za veličinu podskupa atributa tipično se koristi drugi korijen ukupnog broja atributa, pri čemu je taj parametar određen iskustveno. Budući da se kod izgradnje svakog pojedinačnog stabla ne koristi cijeli skup atributa te nema skraćivanja stabla, brzine izgradnje slučajnih šuma i stabala odlučivanja su usporedive, iako se kod slučajnih šuma gradi više stabala odlučivanja.

6. Postupci raspoređivanja s predviđanjem statusa poslova

Postoji nekoliko načina (statusa) na koji poslovi u računalnom grozdu mogu završiti. Statusi uključuju uspješno završene poslove, poslove koji završe s greškom i poslove koji su prekinuti prije uspješnog završetka.

Raspoređivač poslova dobiva informaciju o statusu posla tek po njegovom završetku tako da ne može utjecati na redoslijed izvođenja poslova ovisno o vrsti statusa. Kad bi raspoređivač imao naznaku o statusu posla prije njegovog izvođenja bilo bi moguće favoriziranje poslova ovisno o načinu završetka u svrhu poboljšanja razine usluge korisnicima grozda.

U ovom poglavlju razmatra se mogućnost predviđanja statusa poslova na temelju prošlosti sustava, kao i iskoristivost te informacije u postupku raspoređivanja poslova. U odjeljku 6.1 definirani su svi statusi poslova i način na koji se oni izračunavaju na temelju dostupnih podataka. Potencijalna korisnost i načini iskorištenja predviđanja statusa poslova opisani su u odjeljku 6.2. U odjeljku 6.3 opisan je postupak predviđanja statusa poslova metodama za dubinsku analizu podataka. Algoritam raspoređivanja poslova koji povećava prioritet poslovima koji potencijalno završavaju s pogreškom definiran je i analiziran u odjeljku 6.4.

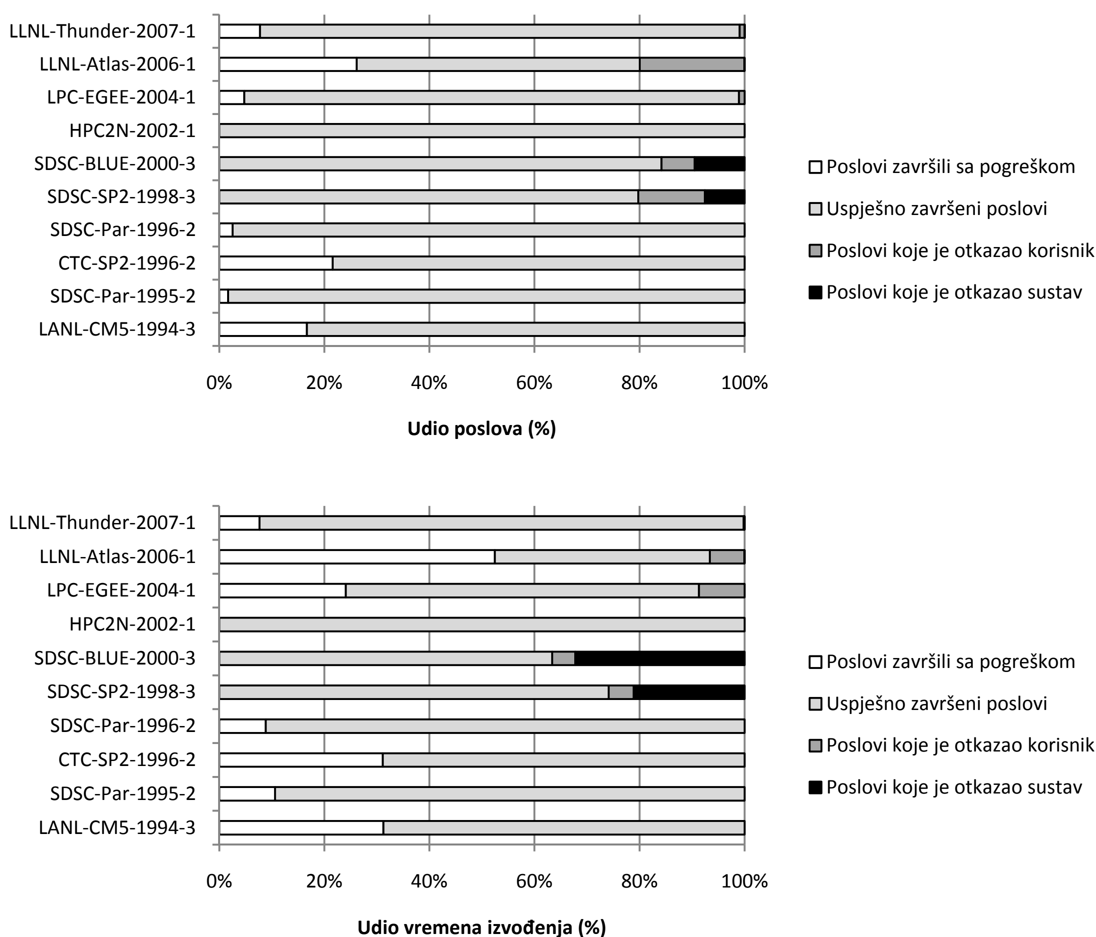
6.1. Definicija statusa poslova

U arhivi paralelnog opterećenja grozdova [77] definirani su slijedeći mogući statusi poslova:

- Uspješno završeni posao
- Posao koji je završio s pogreškom (pogreška nastaje zbog neispravnog funkcioniranja posla, pri čemu program vrati pogrešku ili je prekinut od strane operacijskog sustava zbog nelegalnog korištenja memorijskog prostora)
- Otkazani posao

Budući da poslovi mogu biti otkazani od strane korisnika i od strane sustava, te dvije vrste statusa su razdvojene. Razdvajanje je izvršeno na temelju odnosa vremena izvođenja poslova i vremena za izvođenje koje je zatraženo od strane korisnika.

Korisnički otkazani poslovi su poslovi koji su prekinuti prije početka izvođenja i poslovi koji su otkazani za vrijeme izvođenja, ali prije nego što je vrijeme izvođenja premašilo vrijeme zatraženo za izvođenje tog posla.



Slika 6.1 Udio u broju poslova i vremenu izvođenja za poslove sa različitim statusima

Otkazani poslovi koji su premašili zatraženo vrijeme izvođenja spadaju u grupu poslova prekinutih od strane sustava, odnosno od raspoređivača poslova. Udjeli poslova s različitim statusima na raznim grozdovima računala prikazani su na slici 6.1.

Na gornjem grafikonu prikazani su udjeli broja poslova u ukupnom broju poslova za svaki od statusa, dok su na donjem grafikonu prikazani udjeli vremena izvođenja poslova s određenim statusom u ukupnom vremenu izvođenja svih poslova. Kod tri grozda računala se ne pojavljuju poslovi koji završavaju s pogreškom, dok kod ostalih udio broja poslova koji završavaju pogreškom iznosi između 2% i 24%. Udio istih poslova u vremenu izvođenja je jednak ili do dvostruko veći od udjela broja poslova i iznosi od 5% do 50%, što dovodi do zaključka da je srednje vrijeme izvođenja poslova koji završe s pogreškom veće od srednjeg vremena izvođenja ostalih poslova. Ta činjenica se protivi intuitivnom razmišljanju da se do pogreške dolazi vrlo rano u toku izvođenja posla.

Udio broja poslova koji su uspješno završili izvođenje je očekivano najveći i kreće se od 50% do 99.5%. Udio vremena izvođenja uspješno završenih poslova je također većinski osim u slučaju groza *LLNL-Atlas-2006-1* i iznosi od 54% do 91%.

Broj poslova koje su otkazali korisnici zabilježen je na pet grozdova računala i čini od 1% do 18% ukupnog broja poslova i sudjeluje u vremenu izvođenja s 0.1% do 8.7%. Uobičajeno je da otkazani poslovi imaju kraće vrijeme izvođenja pa sudjeluju i s manjim udjelom u ukupnom vremenu izvođenja poslova.

Poslovi koje je otkazao raspoređivač poslova zabilježeni su na četiri grozda računala. Takvi poslovi sudjeluju u ukupnom broju poslova s 0,01% do 9,1%, a u ukupnom vremenu izvođenja s 0,01% do 32%. Ti poslovi otkazani su zbog prekoračenja korisničke procjene vremena izvođenja te je očekivano da takvi poslovi traju dulje od prosječnih poslova u sustavu.

6.2. Potencijalna korisnost predviđanja statusa poslova

Analizom prošlosti ponašanja sustava u bilo kojem trenutku rada računalnog grozda može se pokušati predvidjeti status poslova koji su pripravni za izvođenje. Takvo predviđanje je nemoguće napraviti bez unošenja određene pogreške pri čemu treba paziti da se prilikom korištenja rezultata predviđanja u raspoređivanju poslova ne pokvari ispravno ponašanje raspoređivača poslova te da se značajnije ne naruše temeljni pokazatelji učinkovitosti grozda računala.

Postoji nekoliko potencijalnih poboljšanja kvalitete usluge prema korisniku koja se mogu ostvariti predviđanjem statusa posla prije njegovog izvođenja. Ukoliko je s određenom pouzdanošću utvrđeno da će neki posao završiti s pogreškom, moguće je povećati prioritet takvih poslova. Povećanjem prioriteta poslova koji će potencijalno završiti s pogreškom, poboljšava se odziv sustava na takve poslove i korisnici će ranije doći do informacije o pogrešci. Ukoliko korisnici dođu dovoljno rano do te informacije, moguće je da će ranije popraviti pogrešku te eventualno otkazati sve poslove koji se trenutno izvode ili čekaju na izvođenje.

Jedan od temeljnih nedostataka povećavanja prioriteta poslova za koje se vjeruje da će završiti s pogreškom je mogućnost korisnika da lažno prikaže status primjenskog programa i da time poveća vjerojatnost predviđanja da će slijedeći njegovi programi završiti s istim statusom. Na taj način je moguće da korisnici koji poznaju rad sustava povećaju prioritet svojih poslova na štetu ostalih korisnika.

Rezultat predviđanja poslova koje je otkazao korisnik može se također iskoristiti u raspoređivanju poslova. Budući da korisnici otkazuju neke poslove za vrijeme izvođenja, a

neke prije izvođenja, moguće je poslovima za koje se smatra da će biti otkazani smanjiti prioritet izvođenja. Time bi se povećala šansa za da ti poslovi budu otkazani i prije nego su započeli izvođenje te bi se tako rasteretilo računalni grozd. Iako postoje potencijalno dobri učinci takve strategije raspoređivanja, stvarnu korisnost takvog pristupa nemoguće je mjeriti s dostupnim podacima budući da korisničko ponašanje u slučaju odgode posla nije moguće predvidjeti. Trenutno ne postoje podaci koji bi omogućili modeliranje otkazivanja poslova.

Predviđanje poslova koje je otkazao sustav zbog prekoračenja vremena izvođenja može se iskoristiti tako da se za takve poslove rezervira više vremena od zatraženog ili da se takvi poslovi izvode u trenucima kad je računalni grozd podopterećen. Podopterećenost grozda omogućila bi da produljivanje vremena izvođenja tih poslova ne narušava raspored za ostale poslove. Temeljni nedostatak ovakvog pristupa leži u činjenici da nema informacija o stvarnom trajanju poslova, pa je nemoguće procijeniti koliko poslova bi uspješno završilo ukoliko im se dozvoli prekoračenje vremena izvođenja . Takvu bi strategiju prvo trebalo isprobati u praksi i na temelju toga izvući podatke o korisnosti raspoređivanja.

Budući da predviđanje poslova koji su završili s pogreškom uz prikladnu promjenu strategije raspoređivanja jedino može dati mjerljive podatke o korisnosti takve strategije, u slijedećim odjeljcima detaljno je analizirano predviđanje statusa poslova s naglaskom na takve poslove.

6.3. Predviđanje statusa statističkim metodama za dubinsku analizu podataka

Za predviđanje statusa korisničkih poslova na računalnom grozdu evaluirano je nekoliko metoda za dubinsku analizu u svrhu odabira najpreciznijeg klasifikatora. Za evaluaciju su odabrane Bayesove mreže, stabla odlučivanja i slučajne šume koje su opisane u poglavlju 5.

Tradicionalno se u dubinskoj analizi podataka klasifikatori treniraju i evaluiraju na jednom skupu podataka, a nakon toga se primjenjuju na nove neklasificirane podatke. Kod predviđanja statusa poslova na računalnom grozdu, za svaki novi posao potrebno je predvidjeti status, s time da je nakon završetka posla poznat i pravi status posla pri čemu je vidljiva i uspješnost predviđanja. Budući da sa završetkom svakog posla nova informacija o statusu postaje dostupna, moguće je ponovno izgraditi klasifikator poslova koji uzima u obzir sve dostupne informacije.

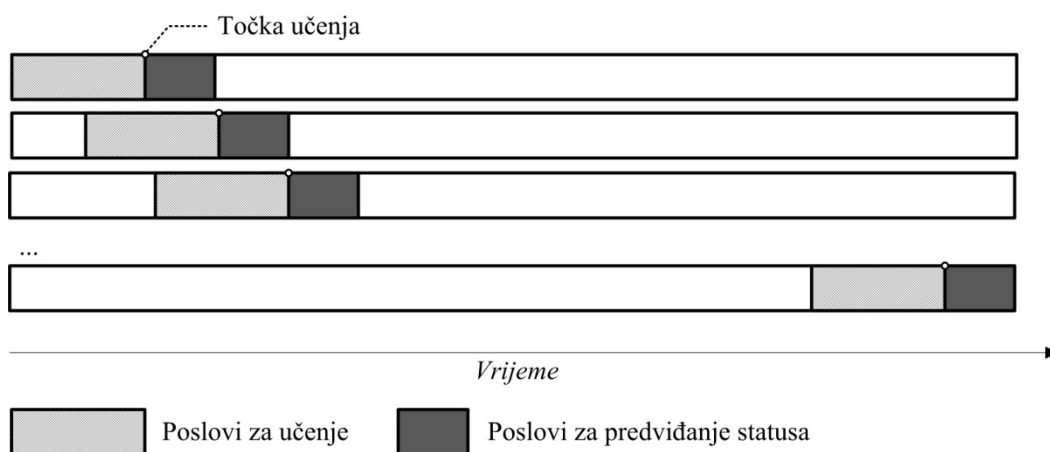
Prilikom predviđanja statusa poslova potrebno je odabrati niz vremenskih trenutaka u kojima se vrši učenje klasifikatora i nakon toga primjenjivati klasifikator u određenom broju predviđanja. Određivanje optimalne količine podataka i najboljeg klasifikatora opisano je u odjeljku 6.3.1. Budući da je bez poznavanja cijele prošlosti sustava nemoguće unaprijed

odrediti optimalnu veličinu podataka za učenje, definirani su i evaluirani dinamički postupci određivanja te količine podataka. Definicija i rezultati evaluacije različitih dinamičkih postupaka prikazani su u odjeljcima 6.3.2 i 6.3.3. Temeljem evaluacije dinamičkih postupaka ustanovljeno je da i promatranje vrlo kratke povijesti dovodi do dobrih rezultata te je oblikovana jednostavna heuristika predviđanja statusa posla koja je opisana u odjeljku 6.3.4. Kombinacija jednostavne heuristike i dinamičkih postupaka predviđanja statusa rezultira hibridnom metodom predviđanja koja je analizirana u odjeljku 6.3.5.

6.3.1. Odabir klasifikatora i optimalne veličine skupa podataka za učenje

Da bi se odredila optimalna količina podataka potrebnih za učenje te odabrao klasifikator koji će se koristiti za predviđanje statusa poslova, uspoređene su učinkovitosti predviđanja tri vrste klasifikatora s različitim veličinama skupa za predviđanje podataka.

Postupak testiranja uspješnosti klasifikatora za jednu količinu podataka iz prošlosti prikazan je na slici 6.2. Veliki pravokutnici ispunjeni bijelom bojom označavaju sve poslove u sustavu vremenski poredane prema dolasku u sustav. Takva aproksimacija vremenskog tijeka u sustavu ne odgovara idealno stvarnom stanju kod raspoređivanja poslova zato jer se poslovi ne izvršavaju nužno redom prispijeća, a i različitog su trajanja. Budući da bi na rezultate predviđanja statusa poslova kod potpuno vjerne simulacije koja uključuje raspoređivanje poslova utjecalo više faktora počevši od strategije raspoređivanja, analiza u ovom poglavlju se radi prema navedenoj aproksimaciji koja zapravo modelira strategiju raspoređivanja poslova prema redoslijedu prispijeća uz ograničenje da se svaki posao smije izvesti tek kada je prethodni završio.



Slika 6.2 Prikaz metodologije učenja klasifikatora i predviđanja statusa za jednu količinu podataka iz prošlosti

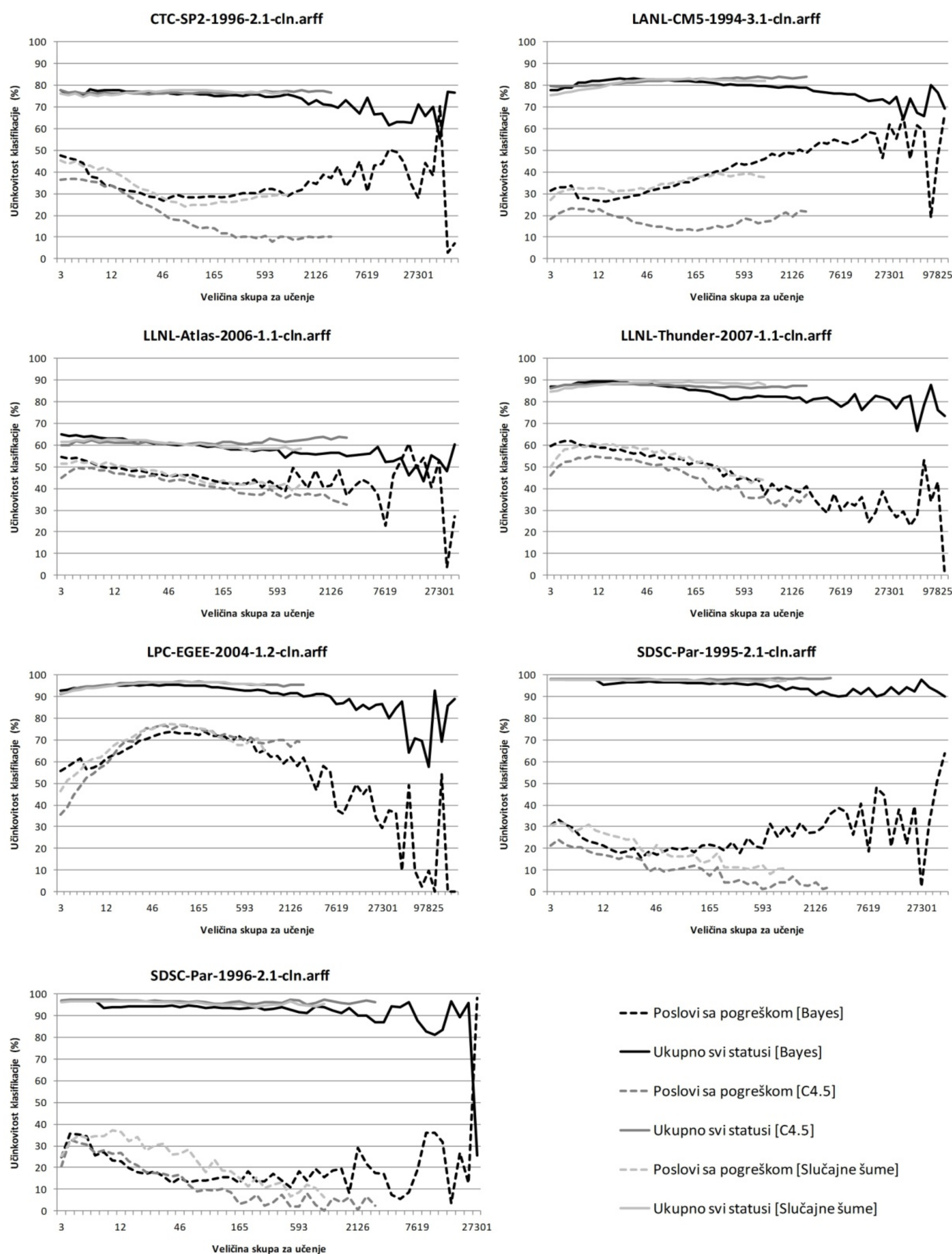
Svaki bijeli pravokutnik na slici 6.2 označava jedan korak testiranja, pri čemu prvi korak označava pravokutnik na vrhu slike. Prva aktivnost u sustavu se provodi na prvoj točki učenja kako je naznačeno na slici. U točki učenja se pomoću određene količine prošlih poslova označenih svjetlo sivom bojom gradi klasifikator čija se uspješnost evaluira na poslovima koji dolaze nakon točke učenja i označeni su tamnijom nijansom sive boje. Omjer broja poslova koji se koristi za učenje i evaluaciju iznosi 2:1. Testirani su i drugi omjeri veličine ta dva skupa do 9:1, ali nisu dobiveni značajno različiti rezultati klasifikacije, a odabir manjeg omjera potencijalno označava rjeđu izgradnju klasifikatora što je povoljno za opterećenje računala na kojem se izvodi raspoređivanje poslova.

Nakon evaluacije jedne grupe poslova sustav za mjerenje se premješta u slijedeću točku evaluacije i gradi se novi klasifikator na temelju prethodnih poslova. Korak pomaka algoritma jednak je veličini skupa za evaluaciju, odnosno polovini veličine skupa poslova za učenje.

Navedeni postupak je izveden za različite veličine skupa za učenje i dobiveni su rezultati za nekoliko klasifikatora koji uključuju Bayesove mreže, stabla odlučivanja i slučajne šume. Na slici 6.3 prikazani su rezultati klasifikacije poslova koji završavaju pogreškom za različite veličine skupa za učenje. Dodatno je prikazana i ukupna točnost predviđanja svih statusa. Ukupna točnost predviđanja je značajno veća od predviđanja poslova koji završavaju s pogreškom jer poslovi koji završavaju pogreškom čine manjinu ukupnog broja poslova. Na apscisi grafikona prikazan je broj poslova na kojem se uči klasifikator, a na ordinati udio istinito pozitivnih (*engl. true positives*) poslova unutar vlastite kategorije. Potrebno je napomenuti da je apscisa prikazana u logaritamskom mjerilu s bazom 1.2 da bi se smanjio broj uzoraka i povećala brzina mjerenja.

Na temelju prikazane analize može se zaključiti da nije moguće odrediti jedinstvenu količinu poslova na temelju koje bi trebalo izvršiti učenje klasifikatora koja bi se univerzalno primjenjivala. Na većini prikazanih računalnih grozdova uspješnost klasifikacije je visoka na jako malom broju poslova za učenje i kontinuirano opada s porastom uzorka. Iznimno kod grozda *LANL-CM5-1994* uspješnost klasifikacije kontinuirano raste s povećanjem uzorka učenja, a kod grozda *LPC-EGEE-2004* raste do uzorka učenja od 66 poslova i dalje kontinuirano opada.

Kod usporedbe triju različitih klasifikatora nisu uspoređene sve veličine učenja klasifikatora zbog ograničenja računalnih resursa. Budući da su Bayesove mreže najbrži od testiranih postupaka učenja, klasifikacija njima je uspješno provedena za sve moguće veličine učenja, dok su kod stabla odlučivanja i slučajnih šuma veličine skupa za učenje ograničene na dvije tisuće i tisuću poslova.



Slika 6.3 Rezultati klasifikacije poslova koji završavaju pogreškom za različite veličine skupa za učenje

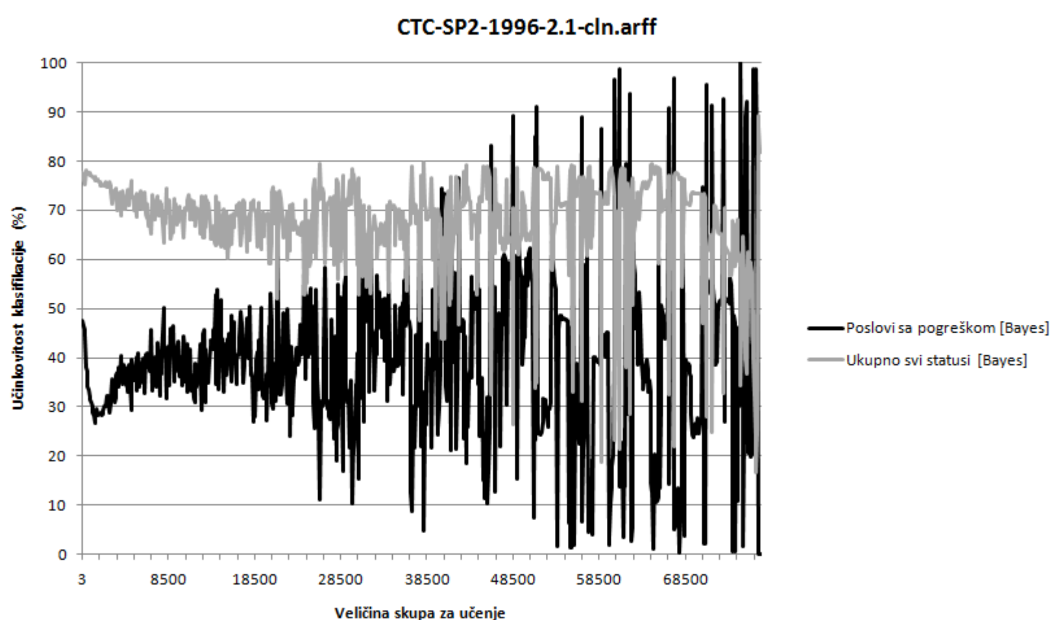
Na svim grozdovima je vidljivo da klasifikacija stablima odlučivanja daje najmanju točnost kategorizacije statusa poslova. Bayesove mreže i slučajne šume daju podjednake rezultate klasifikacije za one veličine uzorka za koje je izvršena klasifikacija slučajnim šumama. Na računalnom grozdu *SDSC-Par-1996* iznimno su slučajne šume ostvarile 10% bolju klasifikaciju statusa poslova do veličine uzorka za učenje od 170 poslova. Detaljnija

razrada i automatizacija klasifikacije poslova izvršena je korištenjem Bayesovih mreža zbog veće brzine rada i mogućnosti učenja na većim uzorcima podataka.

Na slici 6.3 je također vidljivo da kod većih uzoraka dolazi do nestabilnosti u klasifikaciji te da za bliske veličine uzorka za učenje rezultat klasifikacije poslova jako varira. Na primjeru klasifikacije poslova koji završavaju pogreškom na grozdu CTC-SP2-1996 razlika između uspješnosti klasifikacije za veličine uzorka učenja 6349 poslova i 7619 poslova iznosi 15%. Treba napomenuti da nestabilnost u klasifikaciji izgleda puno blaže na logaritamskoj skali nego na linearnoj skali. Na slici 6.4 prikazan je grafikon uspješnosti klasifikacije statusa poslova s kombiniranom skalom pri čemu je do uzorka za učenje od 2000 poslova primijenjena logaritamska skala u bazi 1.2, a na većim uzorcima linearna skala s korakom od 100 poslova pri čemu se vidi velika nestabilnost klasifikacije na veličinama uzoraka većima od 20 000 poslova gdje varijacija u klasifikaciji poslova s pogreškom između susjednih veličina uzoraka dostiže 20%.

Jedan od razloga za takvu varijaciju leži u činjenici da se kod velikih uzoraka za učenje testiranje ne provodi na prvom skupu podataka za učenje te su oni izostavljeni iz mjere učinkovitosti klasifikacije. Izrazita vremenska lokalnost ponašanja korisnika i način mjerenja učinkovitosti opisan na početku odjeljka važniji su razlog za nestabilnost klasifikatora kod većih uzoraka za učenje.

Poslovi koji završavaju pogreškama tipično dolaze u serijama i pojavljuju se sve dok se uzroci pogrešaka ne poprave. Kod velikih uzoraka poslova za učenje automatski se



Slika 6.4 Rezultati klasifikacije poslova za različite veličine skupa za učenje (kombinirana skala)

povećavaju i veličine uzoraka za testiranje pa se može desiti da se cijele serije sličnih poslova s pogreškama ne nađu u skupu za učenje klasifikatora, što uzrokuje smanjenu točnost klasifikacije. Kad se taj zaključak primijeni na dvije susjedne veličine uzorka za učenje, postoji vjerojatnost da će jedan uzorak za učenje pogoditi bar dio serije pogrešnih poslova te značajno unaprijediti sposobnost klasifikacije za tu seriju poslova, dok će drugi, susjedni uzorak promašiti cijelu seriju netočnih poslova.

6.3.2. Dinamički postupak odabira najbolje veličine skupa za učenje metodom stabilne točke

Rezultati klasifikacije statusa poslova iz prethodnog odjeljka ukazuju na različitu količinu podataka iz prošlosti koje je potrebno koristiti za izgradnju klasifikatora. Kako takva analiza i određivanje najbolje veličine uzorka za učenje nije moguće napraviti prije nego računalni grozd završi s radom, potrebno je definirati dinamički postupak odabira duljine prošlosti potrebne za učenje klasifikatora s obzirom na trenutno poznatu povijest sustava.

Promatranjem različitih veličina učenja ustanovljeno je da je zbog lokalnosti pojave poslova koji završavaju pogreškom teško odabrati veličinu učenja klasifikatora koja će se pokazati dobrim klasifikatorom u slijedećem periodu. Da bi se dinamički odredilo veličinu uzorka za učenje klasifikatora primijenjen je algoritam sa slike 6.5. Kroz vrijeme je odabrano 20 točaka kalibracije u kojima se vrši evaluacija klasifikatora za različite veličine uzorka za učenje. Razmak između dvije točke kalibracije iznosi 18 dana na grozdu računala budući da je većina skupova podataka s grozdova računala praćena kroz godinu dana.

U svakoj točki kalibracije se provodi traženje optimalnog broja poslova za učenje pomoću funkcije *OptimalnaVeličinaUčenja*. Na početku funkcije definiraju se svi poslovi koji prethode točki kalibracije kao i različite veličine učenja za koje je potrebno naći optimum klasifikacije. Različite veličine učenja za koje je potrebno testirati postupak klasifikacije poprimaju vrijednosti od 1000 do polovine broja promatranih poslova s korakom od 100 poslova i nalaze se u skupu *VUU*. Funkcija kao optimalnu veličinu za učenje vraća klasifikacijski najuspješniju stabilnu veličinu uzorka poslova za učenje. Stabilna veličina uzorka poslova za učenje definirana je kao veličina uzorka za učenje kod čijih susjednih veličina uzoraka za učenje nema oscilacije u uspješnosti klasifikacije za više od 20%. Susjednim veličinama uzroka za učenje smatraju se one koje su brojem instanci udaljene za maksimalno 25% od promatrane veličine uzorka. Stabilnost veličine uzorka poslova za učenje provjerava se pozivom funkcije *Stabilna*.

$P - skup\ svih\ poslova$
$R: P \rightarrow N, injektivna\ funkcija, preslikava\ svaki\ posao\ u\ prirodni\ broj\ ovisno\ o\ redoslijedu\ kojim\ su\ poslovi\ došli\ u\ sustav$
$korak = \left\lfloor \frac{|P|}{20} \right\rfloor$
$TočkeKalibracije = \{tk: \exists n(n \in N) \wedge (t = korak \times n) \wedge (t < |P|)\}$

$\boldsymbol{OptimalnaVeličinaUčenja}(tKalibracije)\{$
$PosloviPrijeTK = \{p: (p \in P) \wedge (R(p) < tKalibracije)\}$
$VUU = \left\{v: \exists n(n \in N) \wedge (v = 100 \times n) \wedge \left(v \in \left[1000, \frac{|PosloviPrijeTK|}{2}\right]\right)\right\}$
$vrati\ (x: (x \in VUU) \wedge Stabilna(x, VUU, PosloviPrijeTK) \wedge \forall y((y \in VUU \wedge Stabilna(y, VUU, PosloviPrijeTK)) \rightarrow (UspješnostKlasifikacije(x, PosloviPrijeTK) \geq Uspješnostklasifikacije(y, PosloviPrijeTK)))$
$\}$

$\boldsymbol{Stabilna}(veličinaUzorka, VUU, Poslovi)$
$$= \begin{cases} \boldsymbol{true}, & \forall x(x \in VUU \wedge \left|\frac{veličinaUzorka - x}{veličinaUzorka}\right| < 25\% \rightarrow \left|\begin{matrix} UspješnostKlasifikacije(veličinaUzorka, Poslovi) - \\ UspješnostKlasifikacije(x, Poslovi \end{matrix}\right| < 20\%) \\ \boldsymbol{false}, & inače \end{cases}$$

$\boldsymbol{UspješnostKlasifikacije}(veličinaUzorka, Poslovi)=$
$$= \frac{|\{p: p \in Poslovi \wedge R(p) > veličinaUzorka\ \wedge\ Klasificiraj(veličinaUzorka, Poslovi, p) = Status(p)\}|}{|Poslovi| - veličinaUzorka}$$

$\boldsymbol{Klasificiraj}(veličinaUzorka, Poslovi, posao)\{$
$veličinaUzorkaZaTestiranje = \left\lfloor \frac{veličinaUzorka}{2} \right\rfloor$
$kIndeks = \left(\begin{matrix} x: \exists\ n\ (n \in N_0\ \wedge x = veličinaUzorkaZaTestiranje \times n\ \wedge \\ x < R(posao) < x + veličinaUzorkaZaTestiranje \end{matrix}\right)$
$posloviZaUčenje = \left\{\begin{matrix} p: p \in Poslovi\ \wedge \\ (kIndeks - veličinaUzorka < R(posao) < kIndeks) \end{matrix}\right\}$
$vrati\ Bayes(posloviZaUčenje, posao)$
$\}$

Slika 6.5 Dinamički postupak odabira najbolje veličine skupa za učenje metodom stabilne točke

Funkcije *UspješnostKlasifikacije* i *Klasificiraj* su pomoćne funkcije koje se koriste prilikom određivanja optimalne veličine skupa za učenje. Funkcija *UspješnostKlasifikacije* kao parametre prima veličinu uzorka za učenje i skup poslova nad kojima treba procijeniti uspješnost klasifikacije. Uspješnost klasifikacije se definira kao omjer uspješno klasificiranih statusa poslova i ukupno testiranih statusa poslova. Ovako definirana funkcija vraća

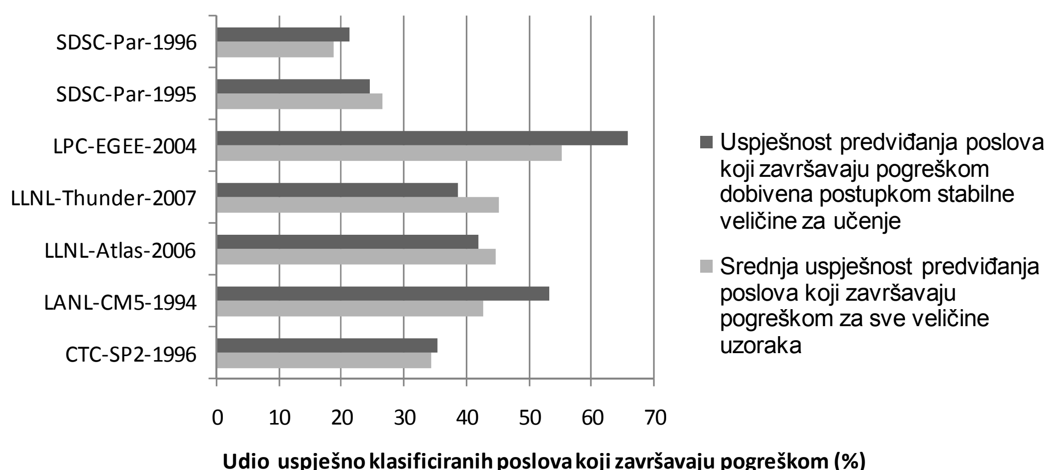
uspješnost klasifikacije za sve vrste završetka poslova. Malom modifikacijom funkcije *UspješnostKlasifikacije* moguće je fokusirati traženje optimalne veličine uzorka za učenje na samo jednu vrstu statusa poslova.

Funkcija *Klasificiraj* služi klasifikaciji posla na temelju skupa poslova i definirane veličine uzorka za učenje Bayesovog klasifikatora. Na početku funkcije određuje se skup poslova za učenje pri čemu *kIndeks* označava indeks zadnjeg posla koji će se koristiti za učenje. Ovakav način izračuna skupa poslova za učenje odgovara postupku iz prethodnog odjeljka prikazanom na slici 6.2. te se koristi da bi se smanjio broj izgrađenih klasifikatora.

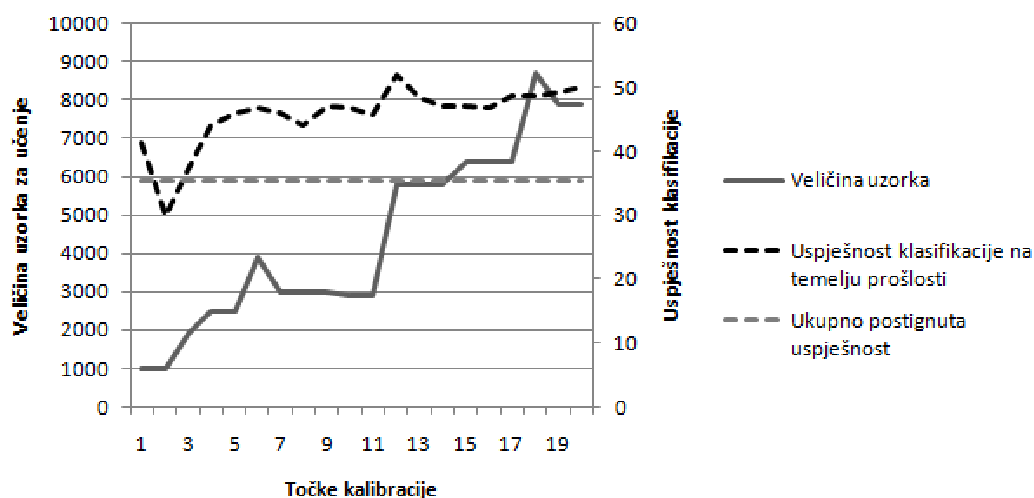
Nakon što se odredi optimalna veličina uzorka za učenje *vu* u točki kalibracije t_k, svi poslovi *p* koji se nalaze između točaka kalibracije t_k i t_{k+1} se klasificiraju funkcijskim pozivom *Klasificiraj(vu,P,p)*.

Na slici 6.6 prikazana je uspješnost klasifikacije poslova koji završavaju s pogreškom pomoću opisanog postupka za računalne grozdove kod kojih se pojavljuju takvi poslovi. Usporedno s tom uspješnošću prikazana je srednja uspješnost klasifikacije izračunata na temelju svih veličina uzoraka za učenje koji su prikazani u prethodnom odjeljku.

Kod svih računalnih grozdova na kojima je izvršeno mjerenje se može primijetiti da klasifikacija metodom stabilne točke daje rezultate vrlo slične srednjoj vrijednosti dobivenoj na temelju klasifikacije statusa poslova sa svim različitim veličinama klasifikatora. Odudaranje od srednjih vrijednosti je vrlo malo i iznosi od -6,54% kod računalnog grozda *LLNL-Thunder-2007* do +10,58% kod računalnog grozda *LANL-CM5-1994*. Na temelju tih rezultata vidljivo je da ova metoda predviđanja odabire veličinu učenja na stabilan način proučavajući uspješnost klasifikatora za različite veličine uzorka za učenje.



Slika 6.6 Uspješnost klasifikacije poslova koji završavaju pogreškom metodom stabilne veličine za učenje



Slika 6.7 Konvergencija stabilne veličine za učenje na grozdu CTC-SP2-1996

Stabilnost navedenog postupka moguće je provjeriti analizom odabira stabilnih veličina uzoraka za učenje i procijenjene uspješnosti predviđanja u prošlosti na temelju kojih je izvršen odabir. Na slici 6.7 prikazana je konvergencija postupka ka globalno najboljoj stabilnoj veličini za učenje prema definiranom dinamičkom postupku na računalnom grozdu *CTC-SP2-1996*.

Usporedba uspješnosti klasifikacije za određenu veličinu uzorka za učenje na temelju prošlih poslova i ukupno postignute uspješnosti pokazuje da je uspješnost odabrane veličine uzorka za učenje veća za ukupno promatranu prošlost na kojoj je evaluirana nego u budućem intervalu u kojem je ta veličina iskorištena za klasifikaciju novih poslova. Mogući razlog takvom ponašanju algoritma leži u činjenici da buduće ponašanje poslova u sustavu ne ovisi nužno o cijeloj prošlosti, nego je vjerojatnije lokalnog karaktera. Završavanje nekog posla s pogreškom vrlo vjerojatno nema prevelike uzročno posljedične veze s istim primjenskim programom drugog korisnika koji se na računalnom grozdu izvodio prije pet mjeseci.

6.3.3. Dinamički postupak odabira najbolje veličine skupa za učenje metodom najboljeg prethodnog klasifikatora

Da bi se poboljšala uspješnost klasifikacije i zanemario učinak jako starih poslova postupak evaluacije različitih veličina uzoraka za učenje na cijeloj prošlosti i odabir optimalnog stabilnog rješenja za buduće predviđanje zamijenjen je postupkom odabira najbolje moguće veličine uzorka za učenje iz novije prošlosti sustava.

Postupak na slici 6.8 prikazuje izmijenjenu metodu određivanja veličine uzorka, pri čemu se također koriste točke kalibracije koje su definirane na jednaki način kao i u

$P - skup\ svih\ poslova$

$R: P \to N, injektivna\ funkcija, preslikava\ svaki\ posao\ u\ prirodni\ broj\ ovisno\ o\ redoslijedu\ kojim\ su\ poslovi\ došli\ u\ sustav$

$korak = \left\lfloor \frac{|P|}{20} \right\rfloor$

$TočkeKalibracije = \{tk: \exists n(n \in N) \wedge (t = korak \times n) \wedge (t < |P|)\}$

$\boldsymbol{OptimalnaVeličinaUčenja}(tKalibracije)\{$

$PosloviPrijeTK = \{p: (p \in P) \wedge (tKalibracije - korak < R(p) < tKalibracije)\}$

$VUU = \{v: \exists n(n \in N) \wedge (v = 100 \times n) \wedge (v \in [1000, tKalibracije])\}$

$vrati\ (x: (x \in VUU) \wedge \forall y(y \in VUU \to \big(UspješnostKlasifikacije(x, PosloviPrijeTK) \geq Uspješnostklasifikacije(y, PosloviPrijeTK)\big))$

$\}$

$\boldsymbol{UspješnostKlasifikacije}(veličinaUzorka, Poslovi) =$

$$= \frac{|\{p: p \in Poslovi\ \wedge\ Klasificiraj(veličinaUzorka, P, p) = Status(p)\}|}{|Poslovi| - veličinaUzorka}$$

$\boldsymbol{Klasificiraj}(veličinaUzorka, Poslovi, posao)\{$

$veličinaUzorkaZaTestiranje = \left\lfloor \frac{veličinaUzorka}{2} \right\rfloor$

$kIndeks = \begin{pmatrix} x: \exists\ n\ (n \in N_0\ \wedge x = veličinaUzorkaZaTestiranje \times n\ \wedge \\ x < R(posao) < x + veličinaUzorkaZaTestiranje \end{pmatrix}$

$posloviZaUčenje = \begin{Bmatrix} p: p \in Poslovi\ \wedge \\ (kIndeks - veličinaUzorka < R(posao) < kIndeks) \end{Bmatrix}$

$vrati\ Bayes(posloviZaUčenje, posao)$

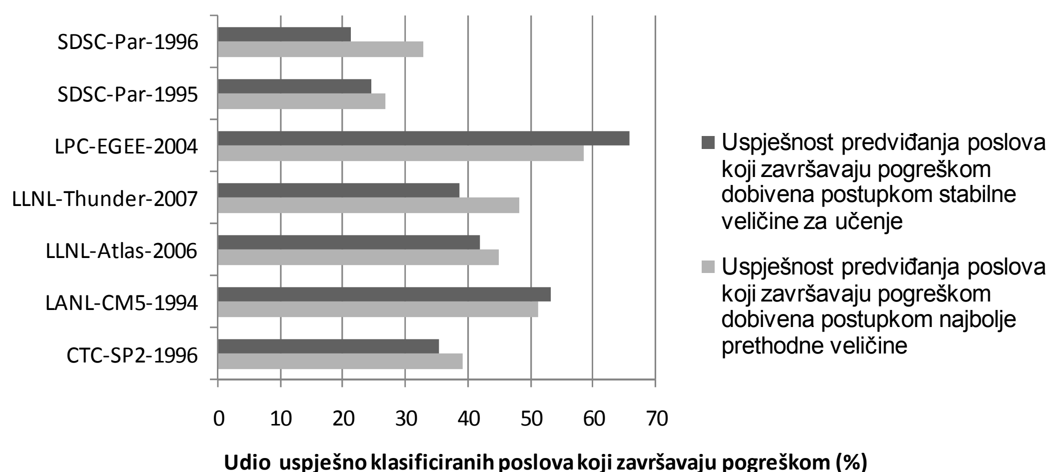
$\}$

Slika 6.8 Dinamički postupak odabira najbolje veličine skupa za učenje metodom najboljeg prethodnog klasifikatora

prethodnom odjeljku. Funkcija *Klasificiraj* također je u cijelosti preuzeta iz algoritma iz prethodnog odjeljka. Kod funkcije *UspješnostKlasifikacije* unesena je izmjena pri čemu se kod klasifikacije testiraju svi poslovi iz zadanog skupa *Poslovi* zato jer se podaci za učenje uzimaju iz poslova koji prethode tom skupu.

Temeljna izmjena postupka provedena je u funkciji *OptimalnaVeličinaUčenja* kojoj je promijenjena semantika. Skup *PosloviPrijeTK* je reduciran sa skupa svih prošlih poslova na skup poslova koji su se pojavili u sustavu između trenutne i prošle točke kalibracije. Skup veličina za učenje *VUU* poprima vrijednosti od 1000 do broja trenutno poznatih poslova koji je jednak iznosu *tKalibracije*.

Odabir najbolje veličine za učenje provodi se tako da se odabere ona veličina uzorka koja ima najbolju uspješnost klasifikacije u periodu između dvije točke kalibracije, i ta veličina za predviđanje se koristi za klasifikaciju poslova u slijedećem periodu sve do nove točke



Slika 6.9 Usporedba postupaka klasifikacije poslova koji završavaju pogreškom

kalibracije. Funkcijski poziv *Klasificiraj(najbolja veličina iz prethodnog perioda, P, posao)* provodi klasifikaciju novog posla *posao.*

Usporedba algoritma koji promatra cijelu prošlost sustava za odabir veličine uzorka za klasifikaciju i algoritma koji odabire najbolju veličinu klasifikacije u prethodnom periodu dan je na slici 6.9. Algoritam koji promatra samo prethodni period za odabir veličine klasifikacije se pokazao uspješnijim kod određivanja poslova koji završavaju pogreškom na svim računalnim grozdovima osim na *LANL-CM5-1994 i LPC-EGEE-2004* gdje je izdvojio 1,82% i 7,06% manje takvih poslova.

U svim ostalim grozdovima računala porast uspješnosti klasifikacije korištenjem algoritma iznosi od 3% za *LLNL-Atlas-2006* do 9,41% za *LLNL-Thunder-2007* čime je opravdana uporaba algoritma koji traži najbolju veličinu uzorka za učenje na temelju prošlosti ograničene na period između dvije točke kalibracije.

6.3.4. Jednostavna heuristika na temeljena na prošlom istovjetnom poslu

Analizom podataka iz odjeljka 6.3.1 vidljivo je da je za većinu računalnih grozdova dovoljno samo nekoliko susjednih podataka za učenje da bi se ostvarila visoka uspješnost klasifikacije statusa poslova. Detaljnijim uvidom u podatke ustanovljeno je da se poslovi koji završavaju pogreškom uglavnom grupiraju u serije, odnosno dolaze jedan za drugim.

Razlog tome leži u tipičnom ponašanju korisnika koji obično pošalju na računalni grozd veću skupinu istovrsnih poslova i onda tek nakon nekog vremena pokušavaju prikupiti rezultate izračuna. Istovrsnim poslovima se smatraju isti primjenski programi s različitim ulaznim podacima. U slučaju da postoji pogreška u primjenskom programu velika je

vjerojatnost da će se ona, za vrijeme korisnikove odsutnosti, manifestirati za sve poslove koje je korisnik poslao u sustav, a započeli su izvođenje odlukom raspoređivača poslova.

Korisnici vrlo često nisu u stanju testirati ispravnost primjenskih programa na platformi na kojoj će se izvoditi budući da moraju čekati na njihovo izvođenje, a vrijeme pokretanja primjenskog programa uglavnom nije jasno određeno zbog svojstava samog raspoređivača poslova i ostalih nedeterminističkih komponenti u sustavu. Jedina mjera koja korisnika kažnjava za trošenje resursa na neispravne poslove je mjerenje njegovog zauzeća grozda računala i podešavanje prioriteta poslova strategijom pravednog korištenja računalnog grozda.

Na temelju ovakvog ponašanja definirana je slijedeća jednostavna heuristika predviđanja statusa poslova temeljena na istovjetnom prošlom poslu:

$$\begin{aligned}
Status(p_{novi}) = Status(p_{stari} \mid & IstovjetniPoslovi(p_{novi}, p_{stari}) \ \wedge \\
& VDolaska(p_{stari}) < VDolaska(p_{novi}) \ \wedge \\
& \neg\exists p_x (IstovjetniPoslovi(p_x, p_{novi}) \wedge \\
& \quad VDolaska(p_{stari}) < VDolaska(p_x) < VDolaska(p_{novi}))
\end{aligned}$$

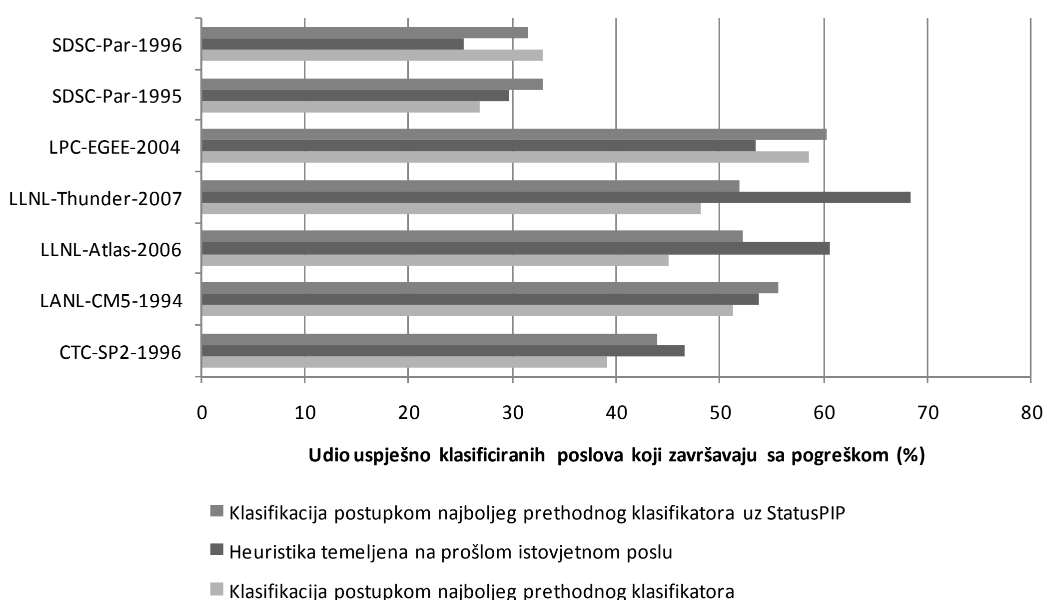
$$\begin{aligned}
IstovjetniPoslovi(p_1, p_2) := & [Korisnik(p_{novi}) = Korisnik(p_{stari})] \wedge \\
& [PrimjenskiProgram(p_{novi}) = PrimjenskiProgram(p_{stari})]
\end{aligned}$$

koja predviđa da će status novog posla biti jednak statusu prethodnog posla u slučaju da je vlasnik oba posla isti korisnik i da se radi o istovrsnom primjenskom programu, pri čemu se za prethodni posao odabire vremenski najbliži prethodnik.

Rezultati klasifikacije poslova takvom jednostavnom heuristikom uspoređeni su s metodom najboljeg prethodnog klasifikatora i prikazani na slici 6.10. Metoda najboljeg prethodnog klasifikatora primijenjena je i na prošireni skup podataka u koji je dodan atribut *StatusPIP* koji označava status prethodnog istovjetnog posla, tako da se i ta informacija može iskoristiti u klasifikaciji statusa poslova.

Kod usporedbe učinka dodanog atributa na klasifikaciju postupkom najboljeg prethodnog klasifikatora, vidljivo je da dodani atribut povećava uspješnost klasifikacije za 3.7% do 6% za sve grozdove osim za *SDSC-Par-1996* gdje neznatno opada uspješnost klasifikacije.

Ukoliko se taj postupak s dodanim atributom usporedi s heuristikom temeljenom na prošlom istovjetnom poslu može se vidjeti da je heuristika temeljena na prošlom istovjetnom poslu uspješnija u klasifikaciji za 2,64% do 16,57% na tri računalna grozda, dok je lošija za 1,18% do 6,93% na ostalim računalnim grozdovima.



Slika 6.10 Usporedba heuristike prošlog istovjetnog posla i klasifikacije postupkom najboljeg prethodnog klasifikatora

Klasifikacija Bayesovim mrežama uzima u obzir značajno dulju prošlost i zavisnost u podacima od heuristike temeljene na prošlom istovjetnom poslu, ali se pokazuje da jako bliska prošlost jako utječe na ponašanje poslova.

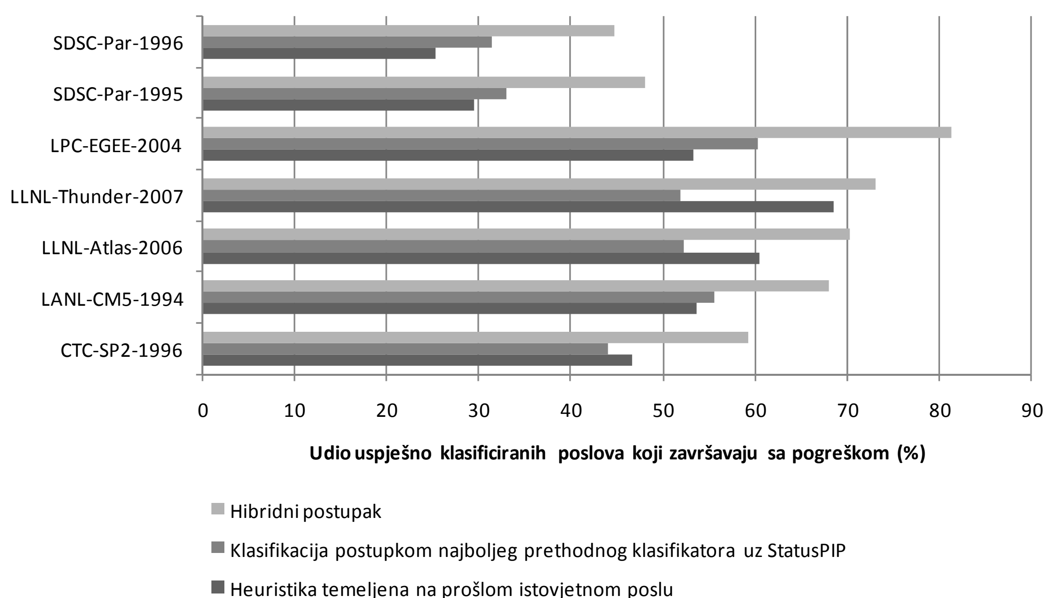
6.3.5. Kombinacija metode najboljeg prethodnog klasifikatora i heuristike temeljene na prošlom istovjetnom poslu

Klasifikacija poslova s pogreškom pomoću Bayesovih mreža analizira dulju prošlost sustava, dok heuristika temeljena na prošlom istovjetnom poslu gleda najbliži prošli posao za predviđanje statusa posla.

Promatranje dulje prošlosti kod određivanja poslova može uhvatiti zavisnosti poput:

- korisnika koji općenito ima problema s pokretanjem poslova na računalnom grozdu
- grupe korisnika koji imaju neispravno podešen primjenski program,
- primjenskih programa koji se pokreću prije vikenda jer traju dulje i zbog povećane potrebe za resursima ostaju bez memorije u sustavu

Takve mogućnosti navode na zaključak da bi hibridna kombinacija Bayesovih mreža i heuristike prošlog istovjetnog posla mogla biti uspješnija u detekciji statusa poslova nego svaka od tih metoda zasebno.



Slika 6.11 Usporedba hibridne metode klasifikacije poslova sa postupcima koji sudjeluju u hibridnoj metodi

Hibridna metoda klasifikacije koja se temelji na postupku najboljeg prethodnog Bayesovog klasifikatora uz *StatusPIP* i metode temeljene na prošlom istovjetnom poslu definirana je slijedećim izrazima:

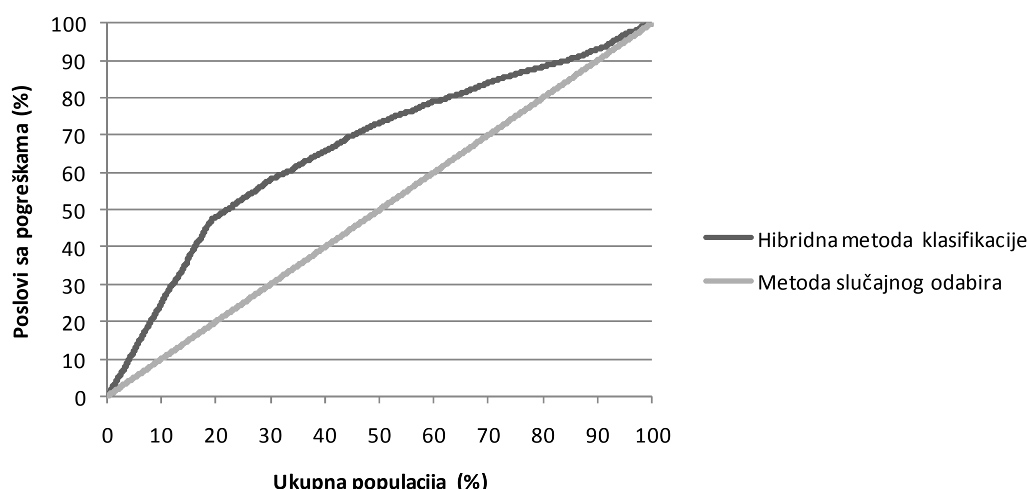
$$\boldsymbol{Status}(p) = \begin{cases} Pogreška & ako\ \boldsymbol{Bayes}(p) = Pogreška \vee \boldsymbol{HeuristikaP}(p) = Pogreška \\ Bayes(p) & inače \end{cases}$$

$$\boldsymbol{Vjerojatnost}(p) = \begin{cases} 1 & ako\ \boldsymbol{HeuristikaP}(p) = Pogreška \\ \boldsymbol{VjerojatnostBayes}(p) & inače \end{cases}$$

Gdje funkcije *Bayes(p)* i *HeuristikaP(p)* vraćaju predviđeni status posla odgovarajućom metodom predviđanja, a funkcije *Status* i *Vjerojatnost* označavaju status posla i mjeru sigurnosti da posao završava s određenim statusom. Mjera sigurnosti je korisna kod raspoređivanja poslova jer omogućava da se razlikuju poslovi unutar istog predviđenog statusa.

U slučaju da bilo koja od dvaju metoda proglasi da će posao završiti s pogreškom, posao se klasificira na taj način, a za klasifikaciju ostalih statusa se koriste samo Bayesove mreže. Mjera sigurnosti predviđanja za poslove koji su klasificirani kao pogrešni zahvaljujući postupku prošlog istovjetnog posla iznosi jedan, a za sve ostale slučajeve jednaka je rezultatu Bayesovog klasifikatora.

Usporedba ovako definirane hibridne metode klasifikacije sa svakom od metoda koja sudjeluje u hibridnoj klasifikaciji prikazana je na slici 6.11. Vidljivo je kako je hibridni



Slika 6.12 Lift krivulja za hibridnu metodu klasifikacije poslova koji završavaju pogreškom na računalnom grozdu CTC-SP2-1996

postupak klasifikacije uspješniji u detekciji poslova koji završavaju pogreškom na svim računalnim grozdovima i bez obzira ne metodu s kojom je uspoređen.

Razlika u broju uspješno klasificiranih poslova između hibridne i najbolje od dvaju prikazanih metoda iznosi od 4,69% do 21,05% što pokazuje razinu preklapanja između klasifikatora koji čine hibridnu metodu. Jasno je da veličina skupa preklapanja tih klasifikatora nije jednaka za sve grozdove, što znači da kod nekih grozdova, poput *LLNL-Thunder-2007* gdje je preklapanje izrazito veliko, najveću ulogu u klasifikaciji imaju susjedni poslovi istog korisnika. Kod nekih računalnih grozdova ukupna uspješnost klasifikacije je vrlo visoka i iznosi do maksimalnih 81,36% kod računalnog grozda *LPC-EGEE-2004*.

Postotak uspješno klasificiranih poslova koji završavaju s pogreškom samo je jedna od mjera za kvalitetu nastalog klasifikatora. Lift krivulja (*engl. Lift chart*) grafički prikazuje koliko je klasifikacija provedena određenim postupkom uspješnija od slučajnog uzorkovanja.

Na slici 6.12 prikazana je lift krivulja za hibridnu metodu klasifikacije poslova zajedno s pravcem koji označava metodu slučajnog odabira. Na apscisi je prikazan udio ukupno promatrane populacije, a na ordinati udio ciljane populacije u uzorku ukupno promatrane populacije. Kod slučajnog odabira udio ciljane populacije u određenom uzorku jednak je udjelu tog uzorka u cijeloj populaciji, tako da je slučajan odabir na lift krivulji prikazan pravcem $y=x$.

Kvaliteta klasifikatora na krivulji uočljiva je iz njegove udaljenosti od pravca slučajnog odabira. Iz slike 6.12 vidljivo je da je klasifikator hibridnom metodom bolji od slučajnog odabira. Ukoliko se poslovi poredaju prema vjerojatnosti klasifikacije poslova koji

završavaju s pogreškom, tada se u 20% poslova iz sustava koje je hibridna metoda označila kao najvjerojatnije poslove koji će završiti pogreškom nalazi 49% poslova s pogreškom, dok je ostalih 51% pogrešno klasificirano.

6.4. Raspoređivanje poslova na temelju predviđanja statusa poslova

Da bi se iskoristila informacija o poslovima koji će potencijalno završiti s pogreškom potrebno je izmijeniti postojeće algoritme raspoređivanja poslova. Algoritam nad kojim je izvršena izmjena je postupak unazadnog popunjavanja praznina s konfigurabilnim brojem rezervacija UPPX.

Kod algoritma unazadnog popunjavanja praznina redoslijed stvaranja rezervacija i popunjavanja praznina ovisi o redoslijedu poslova u prioritetnom redu čekanja. Modifikacija algoritma unazadnog popunjavanja praznina odnosi se na transformaciju prioritetnog reda čekanja, odnosno transformaciju uređaja među poslovima koja se provodi na sljedeći način:

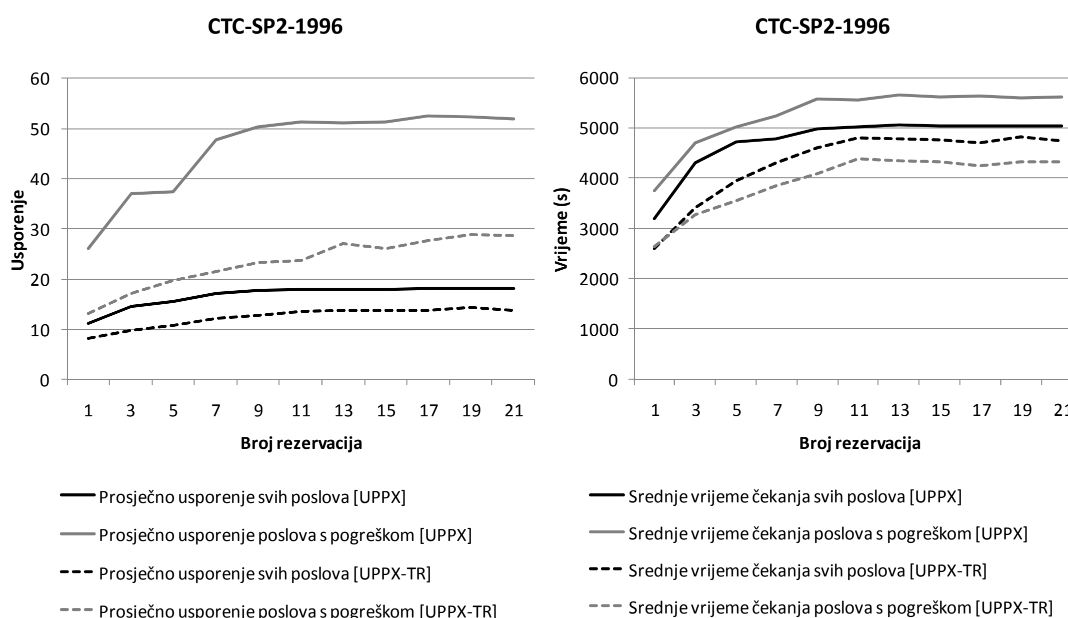
$$<_{Novi}(posao1, posao2) := [\forall p <_{Stari}(posao1, p)] \vee [\exists p <_{Stari}(p, posao2) \wedge vPogreške(posao1) > vPogreške(posao2)]$$

Navedena definicija pokazuje da se *posao1* nalazi ispred *posla2* u novom prioritetnom redu u slučaju da je *posao1* bio prvi u starom prioritetnom redu ili u slučaju da je vjerojatnost pogreške *posla1* veća od vjerojatnosti pogreške *posla2* i da *posao2* istovremeno nije prvi u starom prioritetnom redu.

Efektivno ova transformacija preslikava prvi element starog prioritetnog reda u prvi element novog prioritetnog reda, a ostale elemente starog prioritetnog reda sortira padajući prema vjerojatnosti pogreške i tako sortirane ih pridružuje novom prioritetnom redu. Ovo preslikavanje obavlja se kod svakog dolaska novog posla i izlaska posla iz reda.

Očuvanje pozicije prvog elementa u redu je nužno zbog prevencije izgladnjivanja poslova s vrlo malom vjerojatnošću pogreške. Izgladnjivanje je spriječeno na globalnoj razini iako se čuva pozicija samo za prvi posao iz starog reda, zato jer će nakon izlaska prvog posla iz novog prioritetnog reda na prvo mjesto u redu doći element koji je u originalnom redu bio na drugom mjestu. Takvo ponašanje osigurano je ponovnim stvaranjem novog prioritetnog reda nakon svakog odlaska posla.

Definirani postupak preslikavanja prioritetnih redova jamči ranije izvršavanje poslova s većom vjerojatnošću pogreške uz prevenciju izgladnjivanja. Učinkovitost sustava korištenjem promijenjenog redoslijeda poslova se ne može precizno mjeriti, zato jer je nemoguće predvidjeti reakciju korisnika kojemu je informacija o statusu pogreške u poslu dostupna ranije. Očekivano je da će korisnik koji dođe do informacije o pogreški u



Slika 6.13 Usporedba postupaka UPPX i UPPX-TR za računalni grozd CTC-SP2-1996

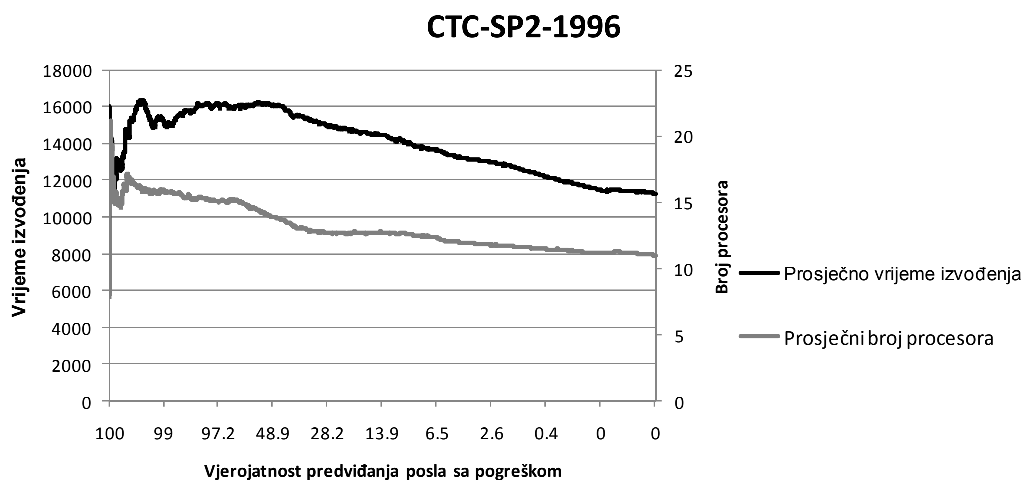
primjenskom programu otkazati ostale istovrsne poslove dok se pogreška ne otkloni, ali takvo ponašanje je nemoguće modelirati na temelju dostupnih podataka.

Modeliranje ponašanja korisnika u takvim slučajevima moglo bi se iskoristiti i za modifikaciju algoritma raspoređivanja, tako da se u slučaju pogreške u primjenskom programu odgode istovrsni poslovi istog korisnika s očekivanjem da će ih korisnik sam otkazati.

S obzirom da ovakvo ponašanje korisnika nije moguće modelirati, provedena je usporedba postupka unazadnog popunjavanja praznina sa standardnim prioritetnim redom uređenim na temelju redoslijeda dolaska poslova u sustav (UPPX) i istog postupka s transformiranim prioritetnim redom (UPPX-TR). Rezultati usporedbe standardnih mjera učinkovitosti raspoređivača poslova za različite brojeve rezervacija na grozdu *CTC-SP2-1996* prikazani su na slici 6.13.

Na slici je vidljivo da podizanje prioriteta poslova za koje se smatra da će završiti pogreškom smanjuje prosječno usporenje tih poslova od 44% do 53% ovisno o broju rezervacija, pri čemu se bolji rezultati postižu za male brojeve rezervacija koji omogućavaju više unazadnog popunjavanja praznina. Dodatno se smanjuje i ukupno prosječno usporenje svih poslova u sustavu za 20% do 33%, s time da su u tu skupinu uključeni i poslovi koji završavaju pogreškom.

Kod analize srednjih vremena čekanja, vidljivo je da je srednje vrijeme čekanja poslova s pogreškom kod korištenja algoritma UPPX-TR smanjeno za 21% do 30% u odnosu na



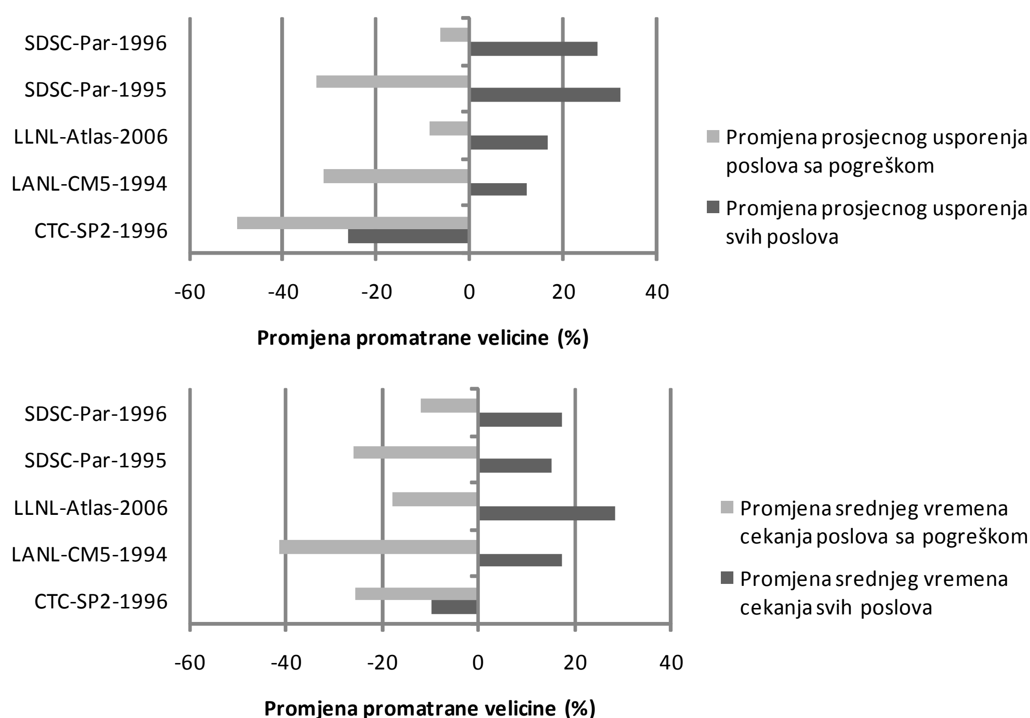
Slika 6.14 Odnos prosječnog trajanja poslova i prosječnog broja zauzetih procesora ovisno o predviđenoj vjerojatnosti da posao završi sa pogreškom

vremena čekanja algoritma UPPX-TR, ili apsolutnim iznosom za 18 do 24 minute. Ukoliko se usporede srednja vremena čekanja za sve poslove, također je vidljivo smanjenje čekanja od 4% do 20%, odnosno 3 do 15 minuta.

Očekivano je da će mjere uspješnosti za poslove koji završavaju pogreškom biti bolje za algoritam UPPX-TR koji prioritetni red poslova preslaguje prema šansi da će određeni posao završiti pogreškom, no nije očekivano ukupno poboljšanje odziva za sve poslove u sustavu. Povećanje prioriteta poslovima koji će potencijalno završiti s pogreškom dovodi do odgađanja ostalih poslova, što može rezultirati različitim ukupnim ponašanjem sustava koje zavisi o prirodi zadanih poslova.

Općenito poboljšanje prosječnog vremena čekanja i prosječnog usporenja u sustavima moguće je postići na dva načina. Jednostavan način za postizanje unapređenja postupka raspoređivanja odnosi se na prioritiziranje kratkih poslova s manjim brojem procesora koji jednakim težinskim faktorom sudjeluju u ukupnoj statistici, a troše značajno manje računalnih resursa. Složeniji način unapređenja postupka raspoređivanja odnosi se na smanjenje fragmentacije u rasporedu izvođenja, odnosno bolje pakiranje poslova.

Da bi se ustanovio razlog zbog kojeg je došlo do poboljšanja usporenja i prosječnog vremena čekanja za sve poslove u sustavu, izmjerena je promjena srednje vrijednosti trajanja poslova i srednje vrijednosti zatraženog broja procesora s predviđenom vjerojatnošću da će posao završiti s pogreškom. Rezultati mjerenja prikazani su na slici 6.14, pri čemu su poslovi na apscisi poslagani prema padajućoj vjerojatnosti da će posao završiti s pogreškom. Točka *(x,y)* za bilo koju od dvije prikazane funkcije na grafikonu označava da je za poslove s vjerojatnošću pogreške većom od *x* srednja vrijednost ciljane funkcije jednaka *y*.



Slika 6.15 Promjena srednjeg vremena čekanja i prosječnog usporenja nastala korištenjem UPPX-TR u odnosu na postupak UPPX

Sa slike je vidljivo da je srednja vrijednost trajanja poslova s vjerojatnoću statusa pogreške većim od 50% jednaka 16000 sekundi, dok je ukupna srednja vrijednost trajanja svih poslova manja od 12000 sekundi. Isto tako je vidljivo da ista grupa poslova zauzima prosječno više od 14 procesora, dok srednji broj zauzetih procesora za sve poslove u sustavu iznosi 11 procesora po poslu.

Na temelju tih podataka moguće je zaključiti kako kod postupka UPPX-TR poboljšanje usporenja i srednje vrijednosti čekanja nije rezultat povećanja prioriteta poslovima koji traju kraće ili zauzimaju manje procesora, nego poboljšavanja pakiranja poslova u rasporedu. Jedan od razloga za bolje pakiranje poslova može biti u činjenici da vremenski susjedni poslovi koji potencijalno završavaju pogreškom imaju sličnost u broju zauzetih procesora i vremenu trajanja što smanjuje fragmentaciju u sustavu. Utjecaj smanjenja fragmentacije kod grozda *CTC-SP-1996* više nego poništava činjenicu da se povećava prioritet poslovima čije je srednje vrijeme trajanja veće od srednjeg vremena trajanja svih poslova u sustavu.

Ovakvo pozitivno poboljšanje prosječnih vremena čekanja i prosječnog usporenja na razini svih poslova u sustavu nije ostvareno na svim računalnim grozdovima. Na slici 6.15 prikazana je srednja promjena prosječnog usporenja i prosječnog vremena čekanja nastala korištenjem postupka UPPX-TR na pet računalnih grozdova. Prosječna promjena izračunata je kao srednja vrijednost promjena mjerenih veličina za različite vrijednosti broja

rezervacija. U usporedbu nije uključen grozd *LPC-EGEE-2004* zbog višestrukih promjena u konfiguraciji i veličini samog sustava u raznim vremenskim trenucima koje potencijalno vode do nedosljednosti u rezultatima i povećavaju složenost postupka simulacije.

Iz rezultata je vidljivo da je jedino na računalnom grozdu *CTC-SP-1996* ostvareno poboljšanje na razini svih poslova. Kod ostalih računalnih grozdova ostvareno je poboljšanje od 6% do 31% za prosječno usporenje i 11% do 41% za prosječno vrijeme čekanja poslova koji završavaju pogreškom. Kod ukupne populacije poslova na ostalim grozdovima računala izmjerena su pogoršanja koja se kreću od 12% do 32% za prosječno usporenje, odnosno od 15% do 28% za srednje vrijeme čekanja.

Razlog pogoršanju učinkovitosti raspoređivača poslova UPPX-TR za većinu računalnih grozdova leži u činjenici da je srednje vrijeme trajanja poslova koji završavaju s pogreškom veće od srednjeg vremena trajanja svih poslova, što automatski udaljuje ovu strategiju poslova od optimalne. Na računalnom grozdu *CTC-SP2-1996* taj efekt je uspješno eliminiran smanjenjem fragmentacije u sustavu zbog sličnosti poslova koji završavaju pogreškom, dok kod ostalih grozdova smanjenje fragmentacije nije utjecalo u dovoljnoj mjeri na poboljšanje učinkovitosti sustava.

7. Postupci raspoređivanja s predviđanjem trajanja poslova

Raspoređivanje poslova u sustavu je otežano zbog neizvjesnosti vezane uz trajanje poslova koji su poslani na izvođenje. Većina postupaka raspoređivanja poslova ovisi o nekoj vrsti informacije o trajanju poslova, pri čemu se vrlo često korisnička procjena trajanja posla smatra kao čvrsti rok završetka posla. Ukoliko posao ne završi u roku kojeg je korisnik procijenio tada raspoređivač poslova prekida posao.

Budući da je raspoređivaču poslova poznata cijela povijest sustava moguće je da na temelju prošlosti izvrši procjenu trajanja novog posla koji je došao u sustav. Takav pristup omogućen je postojanjem sličnosti između poslova koji dolaze na računalni grozd [65].

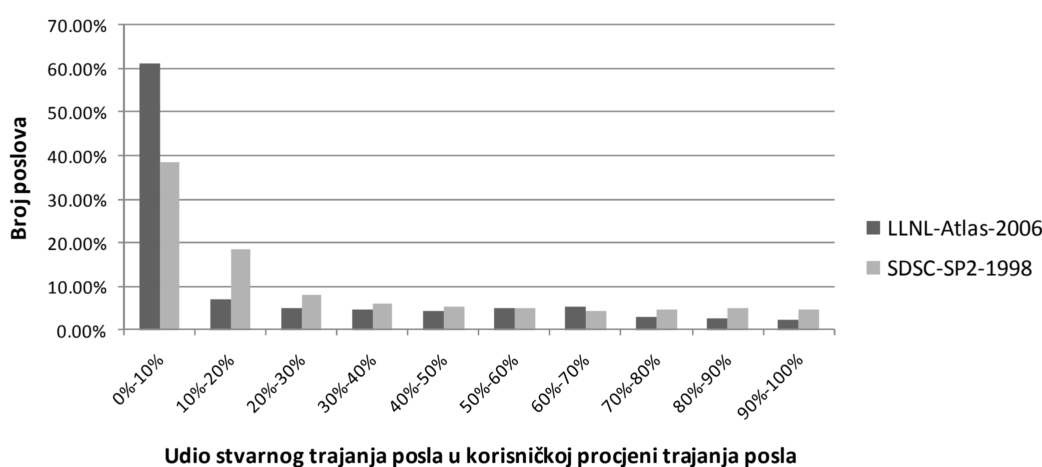
U ovom poglavlju analizirane su korisničke procjene trajanja poslova, metode predviđanja trajanja poslova i utjecaj točnosti predviđanja trajanja poslova na algoritme za raspoređivanje poslova. U odjeljku 7.1 analizirane su korisničke procjene trajanja poslova. Postojeći postupci predviđanja trajanja poslova opisani su u odjeljku 7.2. Da bi se procijenila potencijalna korisnost preciznog predviđanja trajanja poslova na postojeće postupke raspoređivanja poslova utjecaj upotrebe pravih vremena trajanja poslova analiziran je u odjeljku 7.3. U odjeljku 7.4 definiran je postupak procjene trajanja poslova korištenjem metoda za dubinsku analizu podataka te je ocjenjena njegova točnost predikcije. Definicija i procjena učinkovitosti modificiranog postupka raspoređivanja poslova zasnovanog na procjeni trajanja poslova metodama za dubinsku analizu podataka dana je u odjeljku 7.5.

Uz korisničke i automatski određene procjene trajanja poslova, u same poslove je moguće ugraditi prijavu stupnja dovršenosti na temelju koje je moguće poboljšati učinkovitost postupka raspoređivanja poslova. Procjena korisnosti informacije o stupnju dovršenosti posla na postupke raspoređivanja poslova na grozdu računala prikazana je u odjeljku 7.6.

7.1. Korisničke procjene trajanja poslova

Temeljni problemi koji se javljaju kod raspoređivanja poslova odnose se na veličinu pogreške korisničke procjene, razloge krive procjene i mogućnosti poboljšanja procjene korisnika. Na slici 7.1 prikazan je odnos korisničke procjene vremena trajanja poslova i stvarnog trajanja poslova izmjeren na računalnim grozdovima *LLNL-Atlas-2006* i *SDSC-SP2-1998*.

Sa slike se može vidjeti da korisnici daju značajno veće procjene od stvarnog trajanja poslova u velikoj većini slučajeva. Udio korisničke procjene je manji od 10% stvarnog trajanja poslova za preko 60% poslova na računalnom grozdu *LLNL-Atlas-2006*, što znači da



Slika 7.1 Korisnička pogreška procjene trajanja poslova

većina poslova dolazi raspoređivaču s procjenama koje su za red veličine veće od stvarnih vrijednosti. Vrlo je mali udio poslova u sustavu za koje je korisnička procjena unutar 10% stvarne vrijednosti trajanja i iznosi manje od 5%.

Postoji nekoliko razloga za davanje precijenjenih procjena trajanja poslova. Jedan od osnovnih razloga je što većina raspoređivača poslova prekida poslove koji su premašili korisničku procjenu trajanja. Drugi razlog prevelikim procjenama leži u standardnim vrijednostima koje se dodjeljuju poslovima u slučaju da korisnik ne pruži nikakvu informaciju o trajanju poslova. Jedina pozitivna motivacija korisnika da što preciznije predvidi trajanje posla leži u činjenici da poslovi koji kraće traju imaju veću šansu da počnu ranije s izvođenjem kod nekih algoritama raspoređivanja poput algoritma unazadnog popunjavanja praznina. Pitanje je koliko korisnika je upoznato s činjenicom da preciznija procjena potencijalno znači ubrzano izvođenje posla i koliko je ta mogućnost privlačna u odnosu na moguće prekidanje posla zbog preniske procjene trajanja.

Utjecaj korisnikove motivacije na točnost procjene je ispitan [66] tako da je ponuđena nagrada za korisnike koji pruže najtočniju procjenu trajanja posla pri čemu je konfiguracija raspoređivača poslova modificirana na način da se spriječi prekidanje poslova koji prekorače procijenjeno vrijeme trajanja. Ustanovljeno je da je srednja nepreciznost predviđanja trajanja poslova u takvom eksperimentu smanjena s 61% na 57%. Isto tako je pokazano da korisnici koji nisu promijenili svoju procjenu imaju srednju nepreciznost predviđanja 55% što je niže od ukupnog prosjeka svih korisnika.

Uz procjenu vremena trajanja poslova od korisnika je zatražena i ocjena sigurnosti procjene i ustanovljeno je da korisnici koji imaju visoku sigurnost u svoju procjenu i nisu

promijenili odluku o trajanju posla uz nove uvjete imaju najnižu srednju nepreciznost predviđanja koja iznosi 38%.

Nesigurni korisnici koji su mijenjali procjenu trajanja nisu bili u stanju dati ni ocjenu svoje sigurnosti u tu procjenu tako da je srednja nepreciznost takvih korisnika jednaka za sve razine sigurnosti korisnika.

7.2. Postojeći postupci predikcije trajanja poslova

Nekoliko različitih postupaka je razvijeno za predikciju trajanja poslova, s ciljem optimizacije raspoređivanja poslova. Metode predikcije trajanja poslova su redom primijenjene na različite grozdove računala, a na različite načine su prikazane i točnosti predikcije. Rezultati nekih postupaka predikcije iskorišteni su kod različitih algoritama raspoređivanja poslova te su prikazana poboljšanja nekih od mjera učinkovitosti raspoređivanja.

Metoda heurističkog određivanja profila poslova (HOPP) [67] mjeri statističke značajke vremena trajanja za poslove grupirane po korisniku, broju procesora, korisničkoj procjeni vremena izvođenja i količini tražene memorije. Kada dođe novi posao u sustav, na temelju sličnih poslova u prošlosti određuje se srednja vrijednost njegovog trajanja, kao i interval u kojem će se ostvariti 95%-tna pouzdanost predviđanja. Rezultati predviđanja poslova nisu analitički prikazani, ali testirane su različite inačice algoritama posluživanja redom prispijeća, unazadnog popunjavanja praznina i posluživanja s podizanjem prioriteta najkraćih poslova pri čemu su ostvarena smanjenja u srednjem vremenu odziva sustava do 10%.

Napredniji oblik predviđanja trajanja poslova [68] temelji se na traženju skupa dobrih predložaka za određivanje sličnih poslova. Jedan predložak označava način grupiranja poslova, odnosno grupe atributa koji određuju istovrsne kategorije poslova, zajedno s načinom na koji iz određene kategorije poslova odrediti trajanje novog posla. Tipični načini za određivanje trajanja novog posla na temelju njegove kategorije uključuju srednju vrijednost i linearnu regresiju. Kod linearne regresije nužno je da jedan od atributa u kategoriji bude korisnička procjena trajanja posla.

Ideja iza traženja optimalnog predloška leži u činjenici da na različitim računalnim grozdovima različiti skupovi atributa poslova mogu određivati sličnost poslova te je nužno prilagoditi način izračuna trajanja poslova specifičnom sustavu. Odabir optimalnog predloška vrši se pohlepnim pretraživanjem i genetskim algoritmima uz ograničenje promatrane prošlosti. Pokazano je da traženje optimalnog skupa predložaka dovodi do nešto boljih rezultata predviđanja trajanja poslova. Srednja pogreška ovakvog predviđanja trajanja poslova iznosi od 40% do 58% ovisno o računalnom grozdu na kojem je testiran postupak.

Računalni grozdovi na kojima je provedeno mjerenje nalaze se u standardnoj arhivi opterećenja [77]. Nije izmjeren utjecaj ovakvog predviđanja trajanja poslova na raspoređivanje poslova.

Slična metoda temeljena na jedinstvenom predlošku [69] definira skup temeljnih atributa i skup potencijalnih atributa svakog posla. Jedinstveni predložak se izgrađuje reduciranjem potencijalnog skupa atributa koji ne nose informacijsku vrijednost uz dodavanje skupa temeljnih atributa pomoću postupka grubih skupova (*engl. rough sets*) [70]. Predikcija trajanja posla se dobiva grupiranjem poslova prema jedinstvenom predlošku i odabirom srednje vrijednosti trajanja poslova koji se nalaze u istoj grupi kao ciljani posao. Evaluacija postupka je provedena za računalne grozdove *SDSC-Par-1995* i *SDSC-Par-1996* te su dobivene srednje pogreška predviđanja trajanja poslova od 30% i 126%. Rezultati nisu iskorišteni za raspoređivanje poslova.

Jednostavniji skup heurističkih metoda koje uključuju srednju vrijednost najbližih susjeda, težinsku srednju vrijednost najbližih susjeda i lokalno težinsku polinomnu regresiju testiran je na računalnom grozdu sveučilišta Purdue [71]. Pri testiranju je ustanovljeno da korištenje srednje vrijednosti najbližih nekoliko susjeda nadmašuje ostale dvije metode te dovodi do predviđanja s 100% srednjom pogreškom.

Jedna od najjednostavnijih heuristika za određivanje trajanja poslova je srednja vrijednost prošla dva posla istog korisnika [72]. Pokazano je da je s tako jednostavnom metodom predikcije moguće smanjiti srednje usporenje poslova za najviše 28%.

7.3. Savršeno predviđanje trajanja poslova

Jedan od najjednostavnijih načina za procijeniti korisnost predviđanja trajanja poslova je korištenje stvarnih vrijednosti umjesto korisničkih procjena trajanja. Bilo koji heuristički postupak predviđanja trajanja poslova trebao bi točnošću nadmašiti korisničku procjenu, ali nikada neće moći postići apsolutnu točnost.

Da bi se ustanovio prostor unutar kojeg je moguće ostvariti poboljšanje postupka raspoređivanja pomoću heurističkog određivanja trajanja poslova, uspoređena je promjena učinkovitosti dva postupka raspoređivanja poslova koja je ostvarena zamjenom korisničke procjene trajanja posla sa stvarnim izmjerenim vrijednostima. Prvi postupak kod kojeg je provedeno ispitivanje je algoritam prioritiziranja najkraćeg posla (PNP) budući da taj algoritam daje najbolje rezultate za najvažnije mjere učinkovitosti raspoređivanja poslova, dok je drugi postupak algoritam unazadnog popunjavanja praznina (UPP1) koji ima visoku učinkovitost raspoređivanja poslova uz jamstvo da neće doći do izgladnjivanja poslova. Utjecaj primjene stvarnih vrijednosti na algoritme PNP i UPP1 prikazan je u odjeljcima 7.3.1

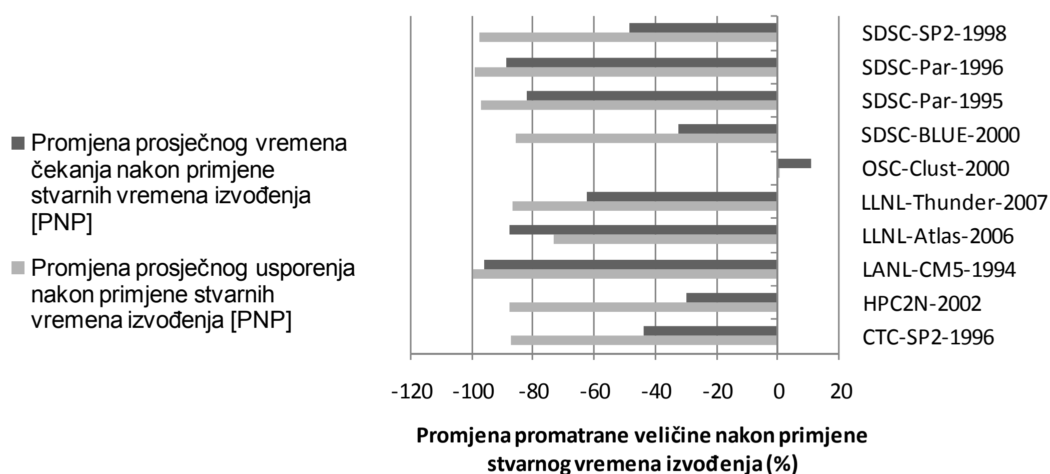
i 7.3.2. U odjeljku 7.3.3 prikazan je utjecaj umjetnog povećanja stvarnih vremena trajanja poslova na učinkovitost algoritma UPP1.

7.3.1. Utjecaj korištenja stvarnih vremena trajanja umjesto korisničkih procjena kod algoritma PNP

Algoritam PNP daje najbolje rezultate za najčešće korištene mjere usporedbe raspoređivača poslova i služi za usporedbu s ostalim postupcima raspoređivanja poslova. Zahvaljujući jednostavnoj heuristici algoritma PNP kod koje je prioritet posla obrnuto proporcionalan njegovom trajanju, očekivano je da će preciznost procjene trajanja posla značajno utjecati na učinkovitost raspoređivanja. Utjecaj korištenja stvarnih vremena trajanja na algoritam PNP izmjeren je na 10 računalnih grozdova i prikazan na slici 7.2.

Sa slike je vidljivo da je na većini računalnih grozdova usporenje smanjeno za preko 70% korištenjem pravih vremena izvođenja umjesto nepreciznih korisničkih procjena trajanja poslova. Jedino na grozdu *OSC-Clust-2000* je primijećeno povećanje usporenja od 0,08%. Detaljnijom analizom podataka ustanovljeno je da je taj grozd jako slabo iskorišten s faktorom iskorištenja manjim od 13%, što označava da se svi poslovi na tom grozdu uglavnom izvršavaju odmah po prispijeću. Takvo ponašanje računalnog grozda dovodi do usporenja jako bliskog vrijednosti jedan bez obzira na strategiju raspoređivanja i predviđanje trajanja poslova. Računalni grozd *OSC-Clust-2000* je zbog slabe iskoristivosti isključen iz ostalih mjerenja u ovom poglavlju.

Na računalnim grozdovima *SDSC-Par-1995*, *SDSC-Par-1996*, *SDSC-SP2-1998* i *LANL-CM5-1994* vidljivo je značajno smanjenje usporenja poslova koje je veće od 97% u svim



Slika 7.2 Utjecaj korištenja stvarnih vremena trajanja umjesto korisničkih predviđanja kod raspoređivanja poslova algoritmom PNP

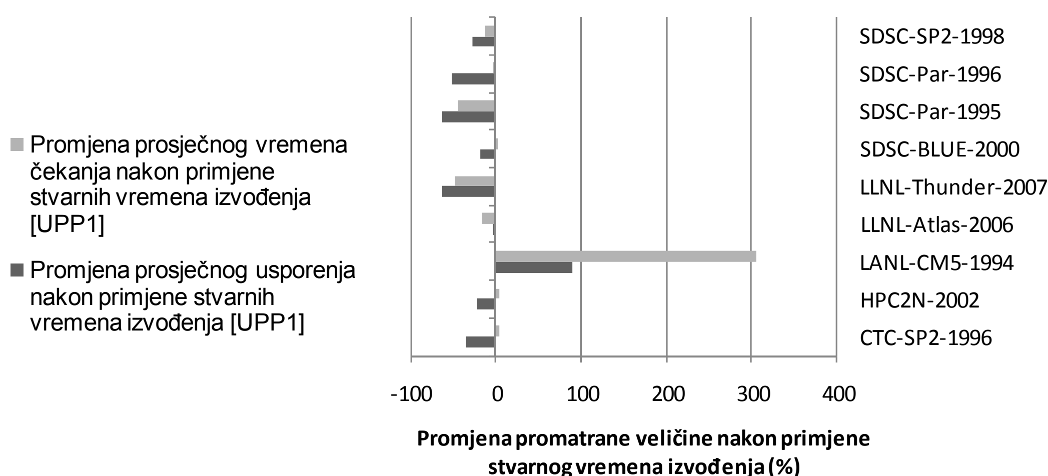
navedenim slučajevima. Razlog tako drastičnoj razlici u rezultatima djelomično leži u nepreciznoj korisničkoj procjeni trajanja posla, dok je jednim djelom zaslužan nedostatak korisničke procjene. U slučajevima gdje korisnička procjena nije navedena, raspoređivači poslova tipično odrede standardiziranu vrijednost trajanja posla ovisno o konfiguraciji particije u koju je posao poslan. Budući da za testirane računalne grozdove nisu navedene standardne vrijednosti trajanja poslova, standardna vrijednost korištena u mjerenjima iznosi 200 000 sekundi (55 sati).

Korištenjem takvih standardiziranih vrijednosti dodatno se pogoršava neprecizna korisnička procjena trajanja posla što jako utječe na prosječno usporenje sustava. Kratki poslovi kod kojih nije navedeno vrijeme izvođenja također su procijenjeni na vrlo visoku vrijednost od 55 sati, što znači da će njihovo izvođenje biti odgođeno sve dok poslovi s kraćom procjenom ne završe s izvođenjem. Usporenje takvih kratkih poslova je jako veliko jer ovisi o omjeru boravka posla u sustavu i njegovog trajanja što dovodi do velikih razlika u prosječnoj srednjoj vrijednosti usporenja.

Prosječno vrijeme čekanja poslova je mjera koja je otpornija na nepreciznost korisničke procjene zato jer u njenu definiciju nije uključeno trajanje posla, već je njena promjena uvjetovana posredno zbog odluka raspoređivača poslova. Izmjereni utjecaj korištenja pravih vremena izvođenja poslova na srednje vrijeme čekanja prikazan na slici 7.2 pokazuje da su prosječna vremena čekanja smanjena za prosječno 63% na svim grozdovima računala. Maksimalno smanjenje od prosječnog vremena čekanja od 95.98% ostvareno je na računalnom grozdu *LANL-CM5-1994* pri čemu je temeljni razlog poboljšanja manjak korisničkih procjena trajanja poslova budući da je procjena dostupna za samo 9,24 % poslova.

7.3.2. Utjecaj korištenja stvarnih vremena trajanja umjesto korisničkih procjena kod algoritma UPP1

Algoritam UPP1 prilikom raspoređivanja poslova koristi korisničke procjene trajanja poslova da bi se odredila duljina rezervacije za posao s najvećim prioritetom, a i da bi se rasporedili ostali poslovi koji mogu popuniti praznine nastale tom rezervacijom. Kod stvaranja rezervacija za poslove trajanje poslova određuje duljinu rezervacije. Nakon što je napravljena rezervacija za posao s najvećim prioritetom popunjavaju se eventualne praznine koje su nastale zbog nove rezervacije posla i poslova koji se trenutno izvode. Da bi se povećala efikasnost popunjavanja praznina važna je precizna procjena trajanja poslova koji bi je potencijalno mogli popuniti. Budući da je najbolja procjena trajanja posla njegovo stvarno trajanje, korištenje stvarnih trajanja poslova prilikom raspoređivanja UPP1



Slika 7.3 Utjecaj korištenja stvarnih vremena trajanja umjesto korisničkih predviđanja kod raspoređivanja poslova algoritmom UPP1

postupkom bi trebalo povećati uspješnost popunjavanja praznina te popraviti ukupnu učinkovitost računalnog grozda.

Usporedba učinkovitosti algoritma UPP1 s korištenjem korisničkih procjena trajanja poslova i stvarnim vremenima trajanja prikazan je na slici 7.3. Na većini računalnih grozdova na kojima je provedeno mjerenje vidljivo je poboljšanje srednjeg usporenja poslova i srednjeg vremena čekanja. Najveće poboljšanje usporenja od 63% ostvareno je na računalnom grozdu *LLNL-Thunder-2007*, dok je najmanje poboljšanje usporenja od 3.5% izmjereno na računalnom grozdu *LLNL-Atlas-2006*. Vrijednosti prosječnih vremena čekanja prate srednje vrijednosti usporenja i iznose od 2% do 48%.

Značajno pogoršanje učinkovitosti raspoređivanja nastalo korištenjem stvarnih vremena trajanja poslova izmjereno je na računalnom grozdu *LANL-CM5-1994* gdje je prosječno usporenje poraslo za 89%, a srednje vrijeme čekanja za 306%. Naknadnom analizom utvrđeno je da kod računalnog grozda LANL-CM5-1994 poslovi nisu prekidani u slučaju prekoračenja korisnički predviđenog vremena trajanja što znači da su vremena trajanja poslova u nekim slučajevima dulja od korisničke procjene. Budući da algoritam UPP1 kod prekoračenja korisničkih procjena automatski prekida takav posao, u scenariju korištenja stvarnih vremena trajanja u zamjenu za korisničke procjene došlo je do povećanja ukupnog obima posla u sustavu što je rezultiralo značajno lošijom učinkovitosti raspoređivanja na računalnom grozdu LANL-CM5-1994.

7.3.3. Utjecaj umjetnog povećanja trajanja poslova na učinkovitost raspoređivanja algoritmom UPP1

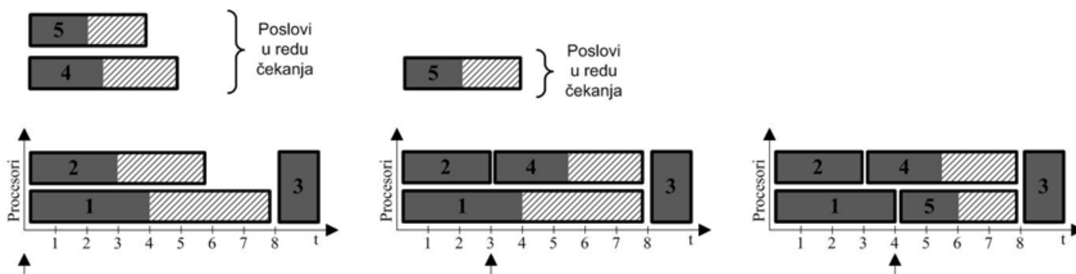
U prethodnim odjeljcima procijenjen je utjecaj zamjene korisničke procjene stvarnim vremenima trajanja poslova za algoritme PNP i UPP1. Kod algoritma UPP1 primiječeno je da umjetno povećanje procjene ili stvarnog trajanja poslova za konstantni faktor [72] [24] smanjuje prosječno usporenje i srednje vrijeme čekanja u sustavu zbog čega korisnost preciznog predviđanja vremena izvođenja poslova kod UPP1 postupka postaje upitna. Kod PNP postupka raspoređivanja poslova nema promjene učinkovitosti s umjetnim povećanjem trajanja poslova ukoliko se ono provede na uniforman način za sve poslove.

Poboljšanje učinkovitosti raspoređivanja poslova kod povećanja procjene trajanja poslova za konstantni faktor događa se zbog implicitnog povećanja prioriteta kratkim poslovima. Primjer umjetnog povećanja prioriteta kratkih poslova zbog korištenja umjetno povećanih trajanja poslova kod algoritma UPP1 prikazan je na slici 7.4.

U početnom trenutku *t=0* u sustavu se nalaze poslovi s prioritetima *1-5*, pri čemu je najviši prioritet označen brojem *1*. Stvarna vremena trajanja poslova označena su sivom bojom, a umjetno dodano povećanje trajanja označeno je teksturom. Raspoređivač poslova UPP1 donosi odluku o trenutnom početku izvršavanja za poslove s prioritetima *1* i *2*, dok se za posao *3* stvara rezervacija za vrijeme *t=8*.

Slijedeći ciklus raspoređivanja izvršava se u trenutku *t=3* koji je obilježen završetkom posla s prioritetom *2*. U tom trenutku rezervacija posla *3* ostaje nepromijenjena, ali zbog dostupnog procesora dolazi do unazadnog popunjavanja praznine i pokreće se posao s prioritetom *4*. U posljednjem prikazanom trenutku raspoređivanja *t=4* dolazi do završetka posla s prioritetom *1* i postupkom unazadnog popunjavanja praznina na istom procesoru se pokreće posao s prioritetom *5*.

Na temelju objašnjenog postupka vidljivo je kako su poslovi s prioritetima *4* i *5* izvršeni prije posla s prioritetom *3*. Takav scenarij je moguć i u slučajevima poznatog trajanja



Slika 7.4 Primjer implicitnog povećanja prioriteta kratkih poslova umjetnim povećanjem trajanja poslova

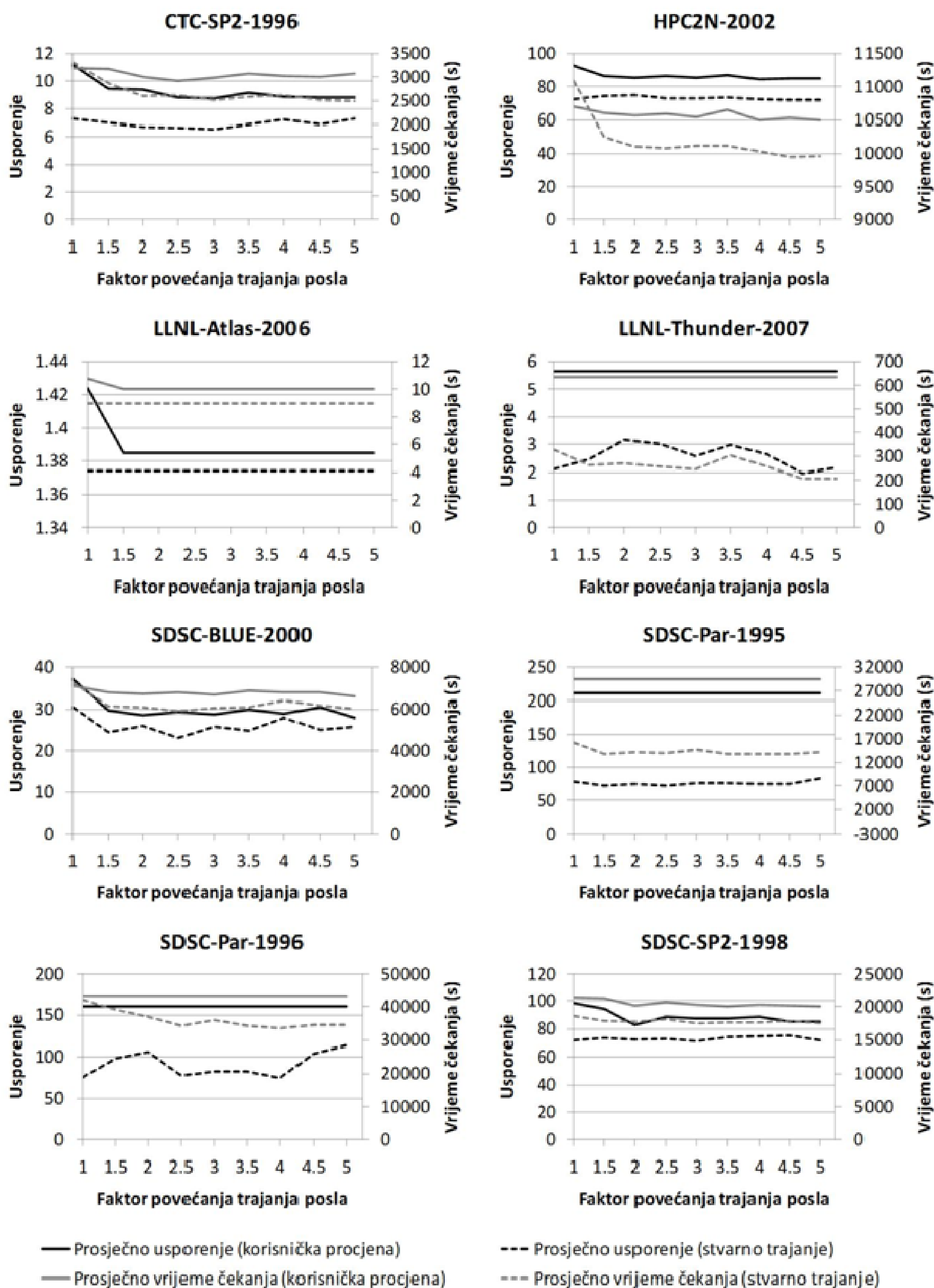
poslova, ali da se isti postupak raspoređivanja primijenio na poslove sa zadanim stvarnim vremenima trajanja ne bi došlo do takve inverzije prioriteta. Budući da je veća šansa da do inverzije prioriteta dođe kod povećanih vremena trajanja poslova i to u korist kraćih poslova koji zahtijevaju manje računalnih resursa, na nekim računalnim grozdovima se umjetnim povećanjem trajanja ili procjene trajanja poslova može postići veća učinkovitost raspoređivanja.

Isto tako, bezgranično umjetno povećanje trajanja može dovesti do negativnog efekta kod kojeg kraći poslovi koji zahtijevaju više resursa budu odgađani u korist dugačkih poslova koji zahtijevaju manje resursa. Na primjeru sa slike 7.4 trajanje posla s prioritetom *3* nije razmatrano, a moguće je da je značajno kraće od svih navedenih poslova. U većini slučajeva poslovi za koje se radi rezervacija nisu uspjeli ranije popuniti praznine te se očekuje da traže mnogo resursa ili vremena.

Analiza utjecaja linearnog povećanja stvarnog i predviđenog trajanja posla za raspon faktora od jednostrukog do peterostrukog povećanja na raspoređivanje poslova za osam računalnih grozdova prikazana je na slici 7.5. Sa slike je vidljivo da umjetno linearno povećanje ne dovodi nužno do poboljšanja prosječnog usporenja i prosječnog vremena čekanja.

Najveće povećanje učinkovitosti sustava s obzirom na usporenje i vrijeme čekanja koji su postignuti povećanjem originalnih korisničkih procjena trajanja poslova ostvareno je na računalnim grozdovima *CTC-SP2-1996* i *SDSC-BLUE-2000*, pri čemu je prosječno usporenje smanjeno za 21% i 23%, a srednje vrijeme čekanja za 8% i 7%. Takvi rezultati ostvareni su za trostruko i dvostruko uvećanje korisničke procjene. Na računalnim grozdovima *LLNL-Thunder-2007*, *SDSC-Par-1995* i *SDSC-Par-1996* nije zabilježena razlika u učinkovitosti raspoređivanja kad su korisničke procjene trajanja poslova multiplicirane s konstantnim faktorom. Takvo ponašanje sustava rezultat je nepostojećih korisničkih procjena na računalnim grozdovima *SDSC-Par-1995* i *SDSC-Par-1996* te vrlo malog broja definiranih korisničkih procjena trajanja poslova na računalnom grozdu *LLNL-Thunder-2007*. U slučaju nepostojećih korisničkih procjena trajanja raspoređivač poslova preuzima standardnu vrijednost procjene koja iznosi 55h koja u većini slučajeva predstavlja značajno uvećanu stvarnu vrijednost trajanja poslova. Daljnje povećanje ionako prevelike standardne vrijednosti konstantnim faktorom ne mijenja ponašanje raspoređivača poslova.

Promjena učinkovitosti raspoređivanja primjenom konstantnog množitelja na stvarna trajanja poslova pokazuje da na većini analiziranih računalnih grozdova dolazi do poboljšanja učinkovitosti postupka raspoređivanja poslova. Na računalnom grozdu *LLNL-Atlas-2006* ne dolazi do promjene usporenja i vremena čekanja zato jer je računalni grozd



Slika 7.5 Utjecaj umjetnog povećanja trajanja poslova na učinkovitost algoritma UPP1

razmjerno neopterećen sa srednjom iskoristivošću resursa od 56% i srednjim vremenom čekanja na izvođenje od samo 9 sekundi.

Razmjerno pravilna promjena učinkovitosti sustava za umjetno povećana stvarna trajanja poslova zabilježena je na računalnim grozdovima *CTC-SP2-1996* i *HPC2N-2002*. Na

računalnim grozdovima LLNL-Thunder-2007, SDSC-BLUE-2000 i SDSC-Par-1996 promjena učinkovitosti raspoređivanja nije kontinuirana s promjenom veličine faktora umjetnog uvećanja stvarnog trajanja poslova. Usporedbom grozdova s kontinuiranom i nepravilnom promjenom učinkovitosti ustanovljeno je da grozdovi s nepravilnom promjenom usporenja i vremena čekanja imaju jako visoku iskorištenost sustava koja iznosi od 76% do 87%. Time je pokazano da su sustavi s visokom iskoristivosti jako osjetljivi na promjenu parametara te da je teško postići značajno bolje rezultate raspoređivanja na takvim sustavima.

Da bi se uklonilo heurističko određivanje faktora multiplikacije vremena trajanja predložena je modifikacija UPP1 algoritma [72]. Modifikacija ostvaruje efekt povećanja prioriteta kraćih poslova, pri čemu su izbjegnuta izgladnjivanja poslova tako da se napravi rezervacija za posao s najvećim prioritetom, a da se praznine nastale tom rezervacijom popunjavaju ostalim poslovima uz podizanje prioriteta kratkim poslovima.

Unatoč različitom ponašanju raspoređivača poslova za razne multiplikativne faktore na različitim računalnim grozdovima, primjena stvarnih trajanja poslova umjesto korisničkih procjena uvijek dovodi do stvaranja učinkovitijeg rasporeda.

7.4. Predviđanje trajanja poslova statističkim metodama za dubinsku analizu podataka

Statističke metode za dubinsku analizu podataka dijele se na klasifikacijske i regresijske postupke. Kod klasifikacijskih postupaka cilj je određivanje klase nepoznatih instanci, dok je kod regresijskih postupaka cilj predviđanje numeričke vrijednosti nepoznatih instanci. Većina postupaka predviđanja trajanja poslova navedenih u odjeljku 7.2 fokusirana je na regresijsku analizu podataka, pri čemu se kao rezultat analize dobiva predviđena numerička vrijednost trajanja posla. Na temelju dobivene vrijednosti i poznate pogreške prediktora može se odrediti interval u kojem se nalazi određeno trajanje posla sa željenom pouzdanošću.

Budući da je nemoguće napraviti predikciju trajanja poslova s apsolutnom točnošću, bitno je odrediti kakva vrsta informacije je najkorisnija kod postupaka raspoređivanja te kako dobiti što kvalitetniju informaciju te vrste. Kod većine postupaka raspoređivanja treba odrediti da li je neki posao odgovarajućeg trajanja da bi ga se rasporedilo u određenom vremenskom intervalu. Specifično kod metoda raspoređivanja koje se zasnivaju na popunjavanju praznina informacija o vjerojatnosti da posao upadne u neiskorišteni prostor za raspoređivanje ključna je za učinkovitost postupka raspoređivanja.

Praznine koje nastaju kao rezultat fragmentacije u sustavu stvaraju se na temelju različitih veličina poslova, što znači da raspon veličina dostupnih praznina treba donekle odgovarati rasponu veličina poslova. S obzirom da je za postupke raspoređivanja bitna procjena mogućnosti poslova da popune praznine određenih veličina definiran je postupak diskretizacije vremena trajanja poslova tako da potencijalno odgovaraju kapacitetima praznina i klasifikacija poslova u diskretne intervale. Diskretizacija trajanja poslova te procjena sigurnosti pripadanja različitim intervalima trajanja poslova nose sa sobom određenu normalizaciju predviđanja trajanja poslova koja može prividno omogućiti bolje pakiranje poslova u vremenu.

Klasifikaciju trajanja poslova na računalnim grozdovima moguće je napraviti nakon što se odredi postupak diskretizacije trajanja poslova i način na koji će se odrediti veličina skupa poslova na kojem će se vršiti učenje klasifikatora. Da bi se odredio broj poslova za učenje klasifikatora provedena je analiza različitih veličina skupova za učenje koja je prikazana u odjeljku 7.4.1. Postupak dinamičke klasifikacije poslova s promjenjivom veličinom skupa za učenje i načina diskretizacije trajanja poslova prikazan je u odjeljku 7.4.2.

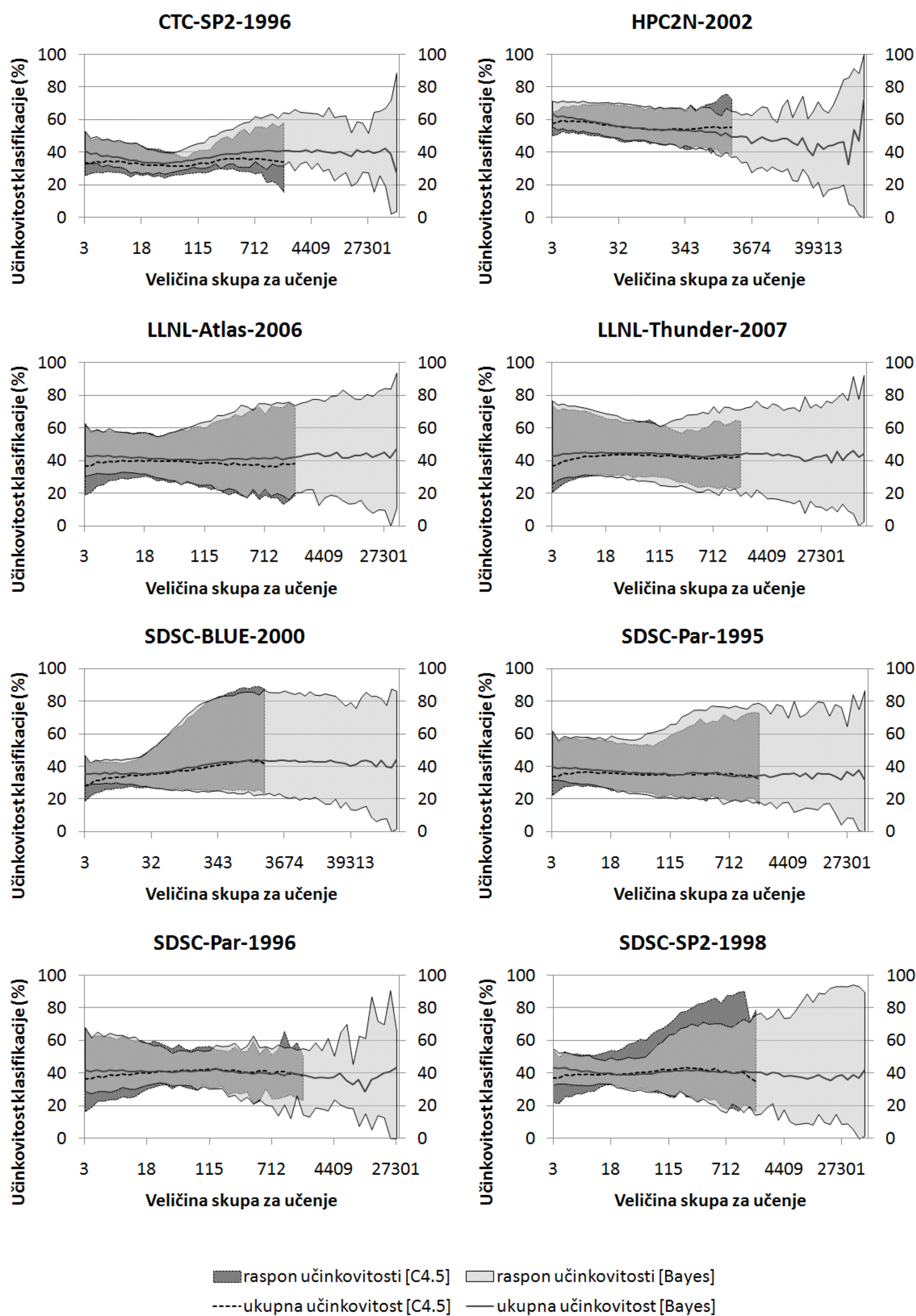
7.4.1. Analiza utjecaja veličine skupa za učenje na klasifikaciju trajanja poslova

Analiza različitih veličina skupova za učenje provedena je pomoću postupka pomičnog prozora (*engl. windowing*) objašnjenim u odjeljku 6.3.1.

Diskretizacija koja je provedena za potrebe analize utjecaja veličine skupa za učenje na klasifikaciju trajanja poslova izvedena je dijeljenjem svih poslova u sustavu u deset intervala trajanja poslova, pri čemu je održana jednakost broja poslova u svim razredima. Ova vrsta diskretizacije trajanja poslova u daljnjim postupcima nije iskorištena budući da u sustavu nije dostupna informacija o trajanju svih poslova sve dok svi poslovi ne budu raspoređeni i završe s izvođenjem.

Na slici 7.6 prikazana je uspješnost klasifikacije trajanja poslova diskretiziranih u deset intervala korištenjem Bayesovih mreža i stabla odlučivanja za osam računalnih grozdova. Uz Bayesove mreže i stabla odlučivanja korištena je i klasifikacija trajanja poslova metodom slučajnih šuma, ali su rezultati bili slični klasifikaciji stablima odlučivanja pa nisu naznačeni zbog pojednostavljenja prikaza.

Na prikazanim grafikonima dane su dvije informacije za svaku od klasifikacijskih metoda. Punom i isprekidanom linijom označena je ukupna uspješnost klasifikacije, odnosno udio uspješno klasificiranih trajanja poslova u svim poslovima za obje klasifikacijske metode. Teksturama su označeni rasponi u kojima se kreću uspješnosti klasifikacije za svaki od deset diskretnih razreda, pri čemu je uspješnost klasifikacije određenog razreda jednaka



Slika 7.6 Rezultati klasifikacije trajanja poslova za različite veličine skupa za učenje

omjeru uspješno klasificiranih poslova tog razreda i ukupnog broja poslova u tom diskretnom razredu. Kod izravne usporedbe Bayesovih mreža i stabla odlučivanja vidljivo je

da su Bayesove mreže neznatno učinkovitije u ukupnoj točnosti klasifikacije trajanja poslova na svim promatranim grozdovima računala. Kod stabla odlučivanja usporedba nije izvršena za sve veličine skupa za učenje zbog ograničenja računalnih resursa. Da bi se s boljom razlučivošću prikazala točnost klasifikacije za manje veličine skupova za učenje os apscisa prikazana je u logaritamskoj skali.

Ukupna točnost klasifikacije ne mijenja se značajno s veličinom skupa za učenje. Veličina skupa za učenje primjetno utječe na širinu raspona točnosti klasifikacije različitih diskretnih razreda, pri čemu je kod većine grozdova računala raspon točnosti najmanji za vrlo male veličine skupa za učenje.

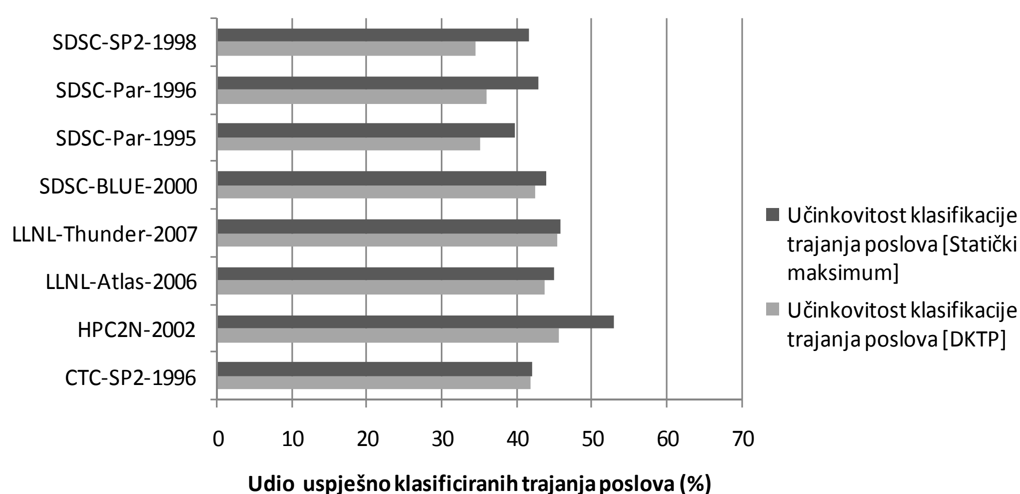
Budući da je redoslijed poslova nad kojim je vršen postupak pomičnog prozora jednak redoslijedu dolazaka poslova u sustav, te se takav redoslijed ne može jamčiti kod raspoređivanja poslova u sustavu, kod odabira veličine skupa za učenje treba zanemariti skupove s manje od nekoliko stotina poslova. U slučaju da se uzme mali broj zadnje završenih poslova za učenje klasifikatora, može se desiti da zbog strategije raspoređivanja poslova upravo završeni poslovi imaju slabu sličnost s poslovima koji čekaju na raspoređivanje pa potencijalno može doći do slabih klasifikacijskih rezultata.

7.4.2. Dinamički postupak klasifikacije trajanja poslova temeljen na Bayesovim mrežama

Dinamički postupak klasifikacije trajanja poslova (DKTP) zasnovan je na Bayesovim mrežama koje su se pokazale neznatno učinkovitijim u statičkoj analizi različitih veličina skupova za učenje, a imaju značajno manju vremensku složenost.

Odabir veličine skupa za učenje provodi se metodom najboljeg prethodnog klasifikatora koja je opisana u odjeljku 6.3.3. U svakoj od točaka kalibracije promatra se prethodni interval i testiraju učinkovitosti klasifikatora izgrađenih raznim veličinama skupova za učenje. Na kraju postupka odabire se klasifikator s najvećom točnosti klasifikacije. Jedina razlika u odnosu na navedeni postupak je uvođenje diskretizacije trajanja poslova. Diskretizacija trajanja poslova se ne može provesti na globalnoj razini jer podaci o trajanju određenog posla postaju dostupni tek u trenutku završetka pojedinog posla. Zbog toga se diskretizacija trajanja poslova provodi u svakom trenutku učenja klasifikatora, pri čemu se na temelju poslova u skupu za učenje provodi određivanje intervala trajanja u koje će se poslovi grupirati.

Ovakav pristup diskretizaciji je povoljan za postupak učenja klasifikatora jer su grupe za učenje stratificirane, odnosno sve su klase trajanja poslova zastupljene u skupu za učenje. Dodatna prednost ovog pristupa je inherentna lokalnost poslova koji se u sustavu nalaze



Slika 7.7 Uspješnost klasifikacije trajanja poslova

unutar istog vremenskog perioda. Može se očekivati da poslovi koji se unutar istog intervala izvode na računalnom grozdu imaju sličnije statističke značajke trajanja poslova od poslova koji su se izvršavali prije nekoliko mjeseci pa će diskretizacija u slične intervale trajanja omogućiti bolje pakiranje vremenski susjednih poslova u praznine koje nastaju u sustavu.

Rezultati klasifikacije trajanja poslova DKTP postupkom prikazani su na slici 7.7. Ti rezultati uspoređeni su s učinkovitosti statičke klasifikacije pri korištenju najbolje moguće veličine skupa za učenje koja je određena isprobavanjem svih mogućih veličina za učenje kako je prikazano u prethodnom odjeljku. Na slici je vidljivo kako DKTP postupak s dinamičkim određivanjem veličine skupa za učenje i dinamičkom diskretizacijom trajanja poslova na većini računalnih grozdova nije značajno lošiji od najboljeg mogućeg statički određenog postupka klasifikacije trajanja poslova. Minimalna razlika u učinkovitosti klasifikacije od 0,78% ostvarena je na računalnom grozdu *CTC-SP2-1996*, dok je maksimalna razlika od 20,58% ostvarena na računalnom grozdu *SDSC-SP2-1998*. Ukupna učinkovitost klasifikacije trajanja poslova postupkom DKTP iznosi od 34,48% do 45,58% na računalnim grozdovima *SDSC-SP2-1998* i *HPC2N-2002*.

Mogućnost da se dinamički postupak odabira veličine skupa za učenje jako približi statički određenoj najboljoj veličini skupa za učenje leži u izrazito lokalnim ponašanjima računalnog grozda kod kojeg odabir optimalne veličine za učenje samo na temelju prošlog segmenta ima gotovo jednaku prediktivnu vrijednost kao i globalno odabrana najbolja veličina skupa za učenje.

Postupak klasifikacije trajanja poslova DKTP nije uspoređen s korisničkim procjenama trajanja poslova zbog različitog karaktera procijenjenih vrijednosti. Korisnička procjena

trajanja posla je numerička vrijednost, dok je rezultat DKTP postupka niz intervala s procijenjenom šansom pripadanja za svaki od intervala pa izravna usporedba nije moguća, a usporedbe koje bi se mogle napraviti pretvorbom između tih dvaju vrsta vrijednosti nužno bi preferirale jednu od metoda klasifikacije. Usporedba točnosti dviju različitih vrsta informacija nije toliko značajna koliko uporabna vrijednost samog podatka u postupku raspoređivanja poslova.

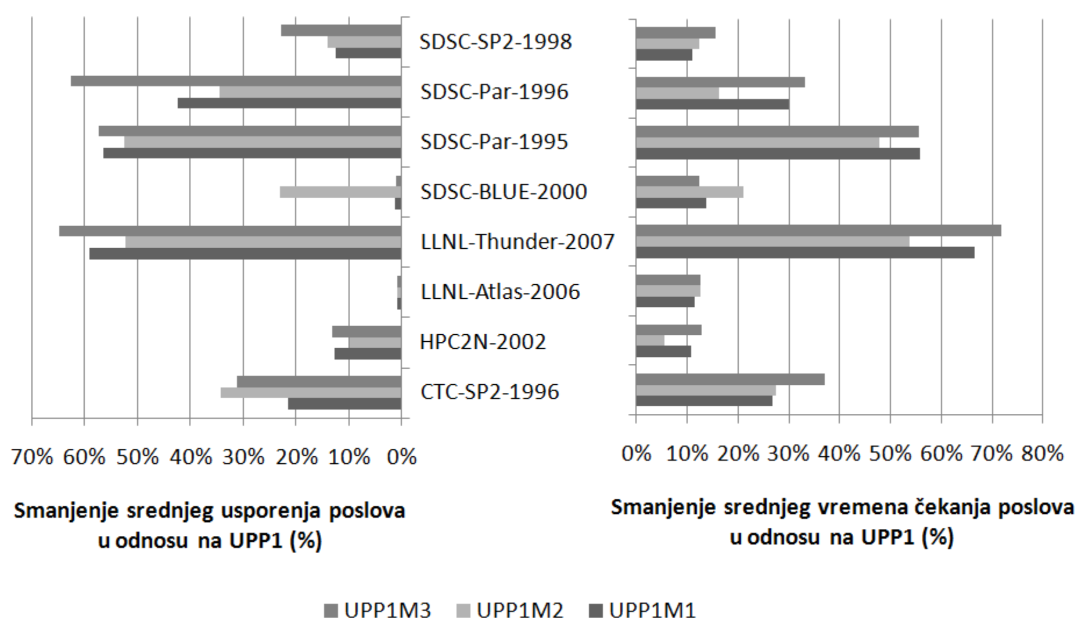
7.5. Utjecaj korištenja predviđenih trajanja poslova kod raspoređivanja modificiranim UPP1 postupkom

U standardnom UPP1 postupku raspoređivanja korisnici daju svoju procjena trajanja posla koje se koristi za odluku da li je u neki slobodni vremenski odsječak određene dimenzije moguće smjestiti taj posao. U slučaju da je korisnička procjena veća od trajanja posla, posao se raspoređuje u slobodni vremenski odsječak i izvodi do isteka trajanja. Ponekad korisnici podcijene trajanje posla što uzrokuje ranije prekidanje posla od strane raspoređivača poslova.

Ovakvu semantiku raspoređivanja poslova nije moguće zadržati uz izravnu zamjenu korisničkih procjena trajanja poslova vrijednostima koje su generirane postupkom DKTP. Temeljni problem korištenja sistemski generiranih procjena trajanja posla je što one mogu podcijeniti trajanje posla u situacijama u kojima korisnici nisu podcijenili trajanje posla. U tom slučaju se ne može prekinuti posao koji je premašio sistemsku procjenu zato jer korisnik na tu procjenu nema utjecaja. Taj problem se rješava jednostavnom modifikacijom postupka raspoređivanja UPP1 [72], kod koje se posao čija je sistemska procjena trajanja manja od korisničke raspoređuje u prikladan vremenski interval na temelju sistemske procjene trajanja. U slučaju da stvarno vrijeme izvođenja premaši sistemsku procjenu trajanja posla, u trenutku u kojem dolazi do isteka sistemske procjene produljuje se rezervacija resursa za taj posao. Duljina produljenja rezervacije jednaka je razlici korisničke i sistemski generirane procjene trajanja posla.

Postupak DKTP ne generira precizne vrijednosti trajanja posla nego intervale s pridruženom pouzdanosti da će vrijeme izvođenja posla pripadati određenom intervalu. Kod sistemski generirane procjene trajanja posla iskorištena je donja granica intervala s najvećom pouzdanošću pripadanja, u namjeri da se ostvari bolje popunjavanje praznina makar i na štetu ostalih poslova.

U slučajevima u kojima je sistemska procjena trajanja posla veća od korisničke procjene koristi se korisnička procjena trajanja posla koja funkcionira i kao vrijeme završetka posla u slučaju da trajanje posla premaši tu procjenu. Učinkovitost raspoređivanja ovako



Slika 7.8 Usporedba postupaka raspoređivanja UPP1M1, UPP1M2 i UPP1M3 sa postupkom UPP1

modificiranog UPP1 postupka (UPP1M1) uspoređena je s UPP1 postupkom koji koristi korisničke procjene i s modificiranim postupkom koji kao procjenu trajanja poslova koristi srednju vrijednost zadnja dva posla istog korisnika (UPP1M2). Rezultat usporedbe prikazan je na slici 7.8.

Postupci UPP1M1 i UPP1MP2 su ocijenjeni relativno prema UPP1 postupku s obzirom na srednje usporenje i srednje vrijeme čekanja poslova i kod oba je smanjeno usporenje za 1% do 59%, dok su srednja vremena čekanja smanjena od 6% do 67% ovisno o promatranom grozdu računala. Postupak UPPM1 je učinkovitiji od jednostavnog heurističkog postupka UPPM2 na pet grozdova računala, dok se na tri grozda računala upotreba jednostavne heuristike pokazala uspješnijom.

Budući da se heuristika s prosječnom vrijednosti prošla dva posla pokazala učinkovitijom za neke grozdove računala konstruirana je kombinirana metoda predviđanja trajanja poslova s pripadajućim postupkom raspoređivanja poslova (UPP1M3). Kod UPP1M3 raspoređivača poslova procjena trajanja poslova jednaka je minimumu predviđanja dobivenih jednostavnom heuristikom i postupkom DKTP. Minimum je odabran zato jer kvaliteta postupka ovisi o uspješnom popunjavanju praznina kraćim poslovima, pa se svi poslovi za koje postoji šansa da su kratki na temelju bilo koje od metoda proglašavaju takvima.

Hibridni postupak UPP1M3 je najuspješniji raspoređivač poslova za sve grozdove računala osim za *SDSC-BLUE-2000*. Smanjenje usporenja u odnosu na postupak UPP1

iznosi od 1% do 65%, uz srednju vrijednost smanjenja usporenja za sve grozdove od 32%. Srednja vremena čekanja su također smanjena od 12% do 72%, uz srednje smanjenje od 31%.

Hibridni postupak UPP1M3 globalno je najučinkovitija metoda predviđanja i raspoređivanja poslova na grozdu računala. Bitna značajka tog postupka je da je potpuno automatiziran i neovisan o karakteristikama grozda računala na kojemu se primjenjuje. Postoji nekoliko parametara u UPP1M3 postupku čijom daljnjom optimizacijom bi se mogli postići još bolji rezultati predikcija trajanja i raspoređivanja poslova.

7.6. Raspoređivanje poslova sa prijavom stupnja dovršenosti

Predviđanje trajanja poslova na temelju prošlosti sustava jedan je od načina smanjenja neodređenosti sustava kod postupaka raspoređivanja na grozdu računala. Alternativa predviđanju trajanja poslova su poslovi sa prijavom stupnja dovršenosti, odnosno poslovi koji u toku izvođenja mogu dati procjenu svog završetka na temelju zadataka odrađenih od trenutka pokretanja posla.

Programeri primjenskih programa koji se izvode na grozdu računala mogu ostvariti prijavu stupnja dovršenosti kao povremenu eksplicitnu dojavu stupnja dovršenosti raspoređivaču poslova ili kao sučelje u programu koje na svaki upit raspoređivača poslova pruža informaciju o stupnju dovršenosti posla.

Raspoređivači poslova kod kojih je omogućeno prekidanje i nastavljanje poslova optimalno bi trebali započeti svaki posao u svrhu što ranije procjene trajanja posla, te bi temeljem prijave stupnja dovršenosti poslova vrlo rano raspolagali razmjerno točnom procjenom trajanja posla. Nakon dobivanja prve procjene trajanja pokrenuti poslovi se mogu zaustaviti te nastaviti kasnije na istom ili različitim skupovima računala.

Kod raspoređivača poslova u sustavima gdje prekidanje i nastavljanje poslova nije podržano dodatna informacija o statusu dovršetka posla korisna je za poslove koji će premašiti dozvoljeno vrijeme trajanja budući da je takve poslove moguće unaprijed zaustaviti. Za poslove kod kojih se na temelju prijave stupnja dovršenosti posla ustanovi da neće premašiti korisničku procjenu trajanja posla, moguće je na temelju dobivene dodatne informacije dodatno optimirati raspored budućih poslova.

Iako se prijava stupnja dovršenosti posla može iskoristiti za optimizaciju izrade rasporeda poslova na različitim vrstama grozdova računala, temeljni problemi korištenja takvog pristupa su ostvarivost prijave unutar primjenskih programa te dobivena preciznost prijavljenog stupnja dovršenosti posla. Ostvarivost prijave stupnja dovršenosti posla unutar samih primjenskih programa mora biti omogućena od strane programera koji su

implementirali program, te dodatno mora biti moguća s obzirom na domenu, odnosno vrstu izračuna koja se tim programom obavlja. Kod nekih primjenskih programa izvođenje se odvija u više stupnjeva pri čemu trajanje nekog od stupnjeva može ovisiti o rezultatima prethodnih stupnjeva. U takvim slučajevima je nemoguće kroz cijeli tijek programa pružiti preciznu informaciju o stupnju dovršenosti posla.

Dodatni problem ovakvom pristupu raspoređivanju odnosi se na motivaciju korisnika da pruže podršku prijavi stupnja dovršenosti posla u primjenskim programima. Procjenu korisnosti ovakvog pristupa simulacijom nije moguće precizno odredit budući da ne postoji vjerni model programa sa prijavom stupnja dovršenosti, a nije poznat ni trud koji bi korisnici bili voljni uložiti da bi se unaprijedila učinkovitost postupka raspoređivanja poslova.

8. Zaključak

Paralelni računalni sustavi su zahtjevna okolina za ostvarenje postupka raspoređivanja zbog nužnosti brzog donošenja odluka i neizvjesnosti vezane uz vrijeme dolaska i karakteristike već pristiglih poslova.

U ovoj disertaciji razmatrana su četiri problema koja uključuju simulaciju grozda računala, raspoređivanje poslova na grozdu računala te predviđanje statusa i trajanja poslova i korištenje tih predikcija pri raspoređivanju poslova.

Simulacija grozda računala ostvarena je raspodijeljenim SARG sustavom, koji omogućava simulaciju različitih postupaka raspoređivanja uz njihovu pojednostavljenu nadogradnju. Temeljna razlika između SARG sustava i ostalih simulatora paralelnih sustava odnosi se na pristup simulaciji entiteta posla. Kod ostalih simulacijskih sustava posao se smatra pasivnom prebrojivom komponentom koja nije u mogućnosti generirati događaje te se ne prati individualno kroz cijelu simulaciju već kao dio grupe poslova. U SARG sustavu je, zahvaljujući korištenju trostupanjske simulacijske jezgre, omogućeno da svi poslovi generiraju zasebne događaje poput prijave stupnja završetka posla i prijave trenutka isteka sistemski predviđenog vremena trajanja. Dodatna vrijednost SARG sustava je posebno oblikovani upravitelj simulacija koji omogućava definiciju simulacija, njihovo raspodijeljeno izvođenje i praćenje u tijeku izvođenja te analizu rezultata završenih simulacija. U toku istraživanja postupaka raspoređivanja SARG sustavom su uspješno izvedene tisuće simulacija različitih grozdova računala uz varijacije parametara raspoređivanja.

U disertaciji je dan prikaz različitih postupaka raspoređivanja prekidivih i neprekidivih poslova. Detaljnija analiza provedena je za neprekidive poslove koji se tipično izvode na grozdovima računala. Usporedbom postupaka ustanovljeno je da se najveća učinkovitost raspoređivanja poslova, uz prevenciju izgladnjivanja, postiže postupkom agresivnog popunjavanja praznina (AUPP). Na temeljima AUPP postupka raspoređivanja izgrađen je postupak unazadnog popunjavanja praznina uz optimizaciju dinamičkim programiranjem (UPPDP) koji ima nekoliko varijanti. Najbolja varijanta UPPDP postupka superiorna je AUPP postupku za sve testirane grozdove računala, pri čemu je ostvareno smanjenje srednjeg vremena čekanja od 0,3% do 40% ovisno o vrsti i iskoristivosti mjerenog sustava.

Analizom računalnih grozdova ustanovljeno je da ponekad veliki dio resursa u sustavu zauzimaju poslovi koji završavaju pogreškom. Uz pretpostavku da korisnici svjesno ne pokreću poslove koji ne mogu valjano završiti, ranije izvođenje poslova za koje se pretpostavlja da završavaju pogreškom može dovesti do ranijeg ispravljanja pogreške i potencijalnog otkazivanja poslova za koje korisnici smatraju da će također završiti

pogreškom. Da bi se omogućilo ranije izvođenje potencijalno neispravnih poslova oblikovan je postupak predviđanja tih poslova pri čemu se udio uspješno klasificiranih neispravnih poslova kreće od 45% do 78% ovisno o grozdu računala. Ovakvo predviđanje omogućava povećanje prioriteta tih poslova u svrhu ranog informiranja korisnika. Potencijalno loši efekti povećanja prioriteta, odnosno posredno ranijeg izvođenja neispravnih poslova, također su izmjereni na modificiranom AUPP postupku. Kod većine grozdova računala srednje vrijeme čekanja svih poslova povećano je za 17% do 27%, dok je na jednom grozdu računala zabilježeno smanjenje od 10%. Ovo smanjenje ukupne učinkovitosti poslova napravljeno je zbog povećanja razine usluge neispravnim poslovima kod kojih je ostvareno smanjenje srednjeg vremena čekanja od 10% do 40%.

Svi postupci raspoređivanja poslova na grozdu računala koriste korisničke procjene trajanja kao čvrsti krajnji rok završetka posla. U disertaciji je ostvarena izvorna metoda sistemske procjene trajanja poslova zasnovana na klasifikaciji Bayesovim mrežama. Korištenjem novih sistemskih procjena trajanja poslova, uz korisnički definirane čvrste krajnje rokove završetka poslova, modificiranim AUPP postupkom raspoređivanja ostvareno je smanjeno srednje vrijeme čekanja od 12% do 72% ovisno o grozdu računala. Uz predviđanje trajanja poslova komentirana je mogućnost ostvarenja poslova sa prijavom stupnja dovršenosti, te potencijalna korist te informacije u postupcima raspoređivanja poslova na grozdu računala.

Potencijalne smjernice budućeg istraživanja zasnovane na ostvarenim rezultatima uključuju modeliranje reakcije korisnika na ranu dojavu neispravnih poslova te predviđanje dolazaka poslova u budućnosti. Detaljni razvoj postupaka raspoređivanja za poslove sa samostalnom prijavom stupnja dovršenosti također je u planu nastavka istraživanja.

DODATAK A – Uporaba SARG sustava

SARG sustav ostvaren je kao programska biblioteka koja čini temelj za izgradnju različitih diskretnih simulacija uz proširenja namijenjena olakšanoj izradi simulacija grozda računala. U ovom dodatku je prikazana izgradnja raspoređivača poslova kod kojeg redoslijed izvođenja poslova odgovara redoslijedu dolazaka tih poslova u sustav, koja čini varijantu PDP postupka raspoređivanja poslova. Uz dogradnju sustava novim raspoređivačem poslova prikazan je način upravljanja izvođenjem simulacija te analiza rezultata putem komponente upravitelj simulacijama.

U odjeljku A.1 prikazan je na koji su način raspoređene komponente unutar programskog ostvarenja SARG sustava. Primjer ugradnje novog postupka raspoređivanja poslova u SARG sustav opisan je u odjeljku A.2. U odjeljku A.3 dan je detaljni prikaz upravljanja simulacijama u SARG sustavu.

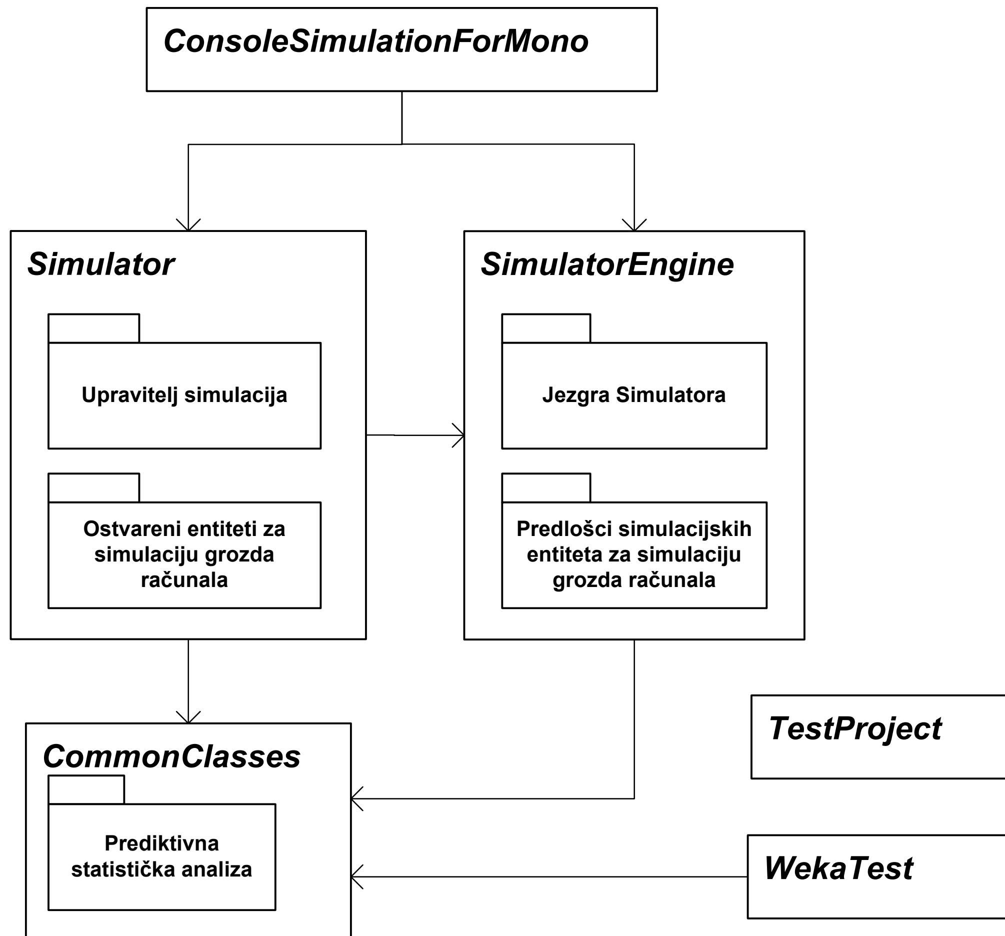
A.1. Programsko ostvarenje SARG sustava

Programsko ostvarenje SARG sustava izgrađeno je na temelju arhitekture prikazane u poglavlju 4 u razvojnoj okolini Visual Studio 2008 [87]. Komponente SARG sustava ostvarene su kao zasebni projekti ili kao dijelovi projekata unutar razvojne okoline. Na slici A.1 prikazana je međusobna zavisnost projekata, te raspored komponenti unutar projekata čije poznavanje je nužno kod nadogradnje SARG sustava.

Projekt *CommonClasses* ostvaruje sve funkcionalnosti komponente *Prediktivna Statistička Analiza*, uz dodatke raznih pomoćnih funkcija koje se koriste u više projekata. U projektu *WekaTest* ostvaren je komandno linijski primjenski program namijenjen prediktivnoj analizi podataka o opterećenjima grozdova računala. Navedeni program se izvodi odvojeno od simulacije grozda računala u svrhu određivanja optimalnih parametara dubinske analize koji će se koristiti kod simulacije grozda računala.

Projekt *SimulatorEngine* ostvaruje *Jezgru Simulatora* koja je nužna za ostvarivanje generičkih diskretnih simulacija i komponentu *Predlošci simulacijskih entiteta za simulaciju grozda računala* kojom su ostvareni osnovni razredi nužni za simulaciju grozda računala. Komponente *Upravitelj simulacija* i komponenta *Ostvareni entiteti za simulaciju grozda računala* dio su projekta *Simulator* kojim je omogućeno definiranje, upravljanje i usporedba rezultata simulacija temeljenih na dostupnim simulacijskim entitetima.

Da bi se omogućilo izvođenje diskretnih simulacija na Linux operacijskom sustavu izgrađen je zaseban komandno linijski primjenski program koji je sadržan u projektu *ConsoleSimulationForMono*.



Slika A.1 Klasifikacija MIMD paralelnih sustava

Različiti testovi funkcionalnosti različitih razreda svih projekata nalaze se u projektu *TestProject*. Na temelju različitih testova unutar projekta *TestProject* vidljiv je princip oblikovanja jednostavnih simulacija te načini na koje je moguće koristiti dijelove funkcionalnosti cijelog sustava.

A.2. Implementacija PDP postupka raspoređivanja poslova

Implementaciju novog postupka raspoređivanja sustava moguće je ostvariti kao zasebnu komponentu ili ugraditi kao jedan od postupaka unutar SARG sustava.

Kod proširenja SARG sustava novim raspoređivačem poslova, novi raspoređivač poslova potrebno je pohraniti u projekt *Simulator* koji ostvaruje i sve ostale podržane strategije raspoređivanja poslova. U slučaju da se novi postupak raspoređivanja razvija kao izdvojena komponenta potrebno je stvoriti novi projekt koji referencira projekt *Simulator*. Primjer implementacije PDP postupka s prioritetima definiranima prema redoslijedu prispijeća prikazan je na slici A.2.

```
public class PDP_Scheduler : SchedulerEntity{
  List<JobEntity> queuedJobs = new List<JobEntity>();
  List<ResourceEntity> freeResources = new List<ResourceEntity>();
  Dictionary<JobEntity, List<ResourceEntity>> allocatedRes =
            new Dictionary<JobEntity, List<ResourceEntity>>();

  public PDP_Scheduler(SimulationExecutive exc): base(exc){}

  public override void queueJobHandling(JobEntity jobE){
    queuedJobs.Add(jobE);
  }

  protected override void handleActivatedResurce(ResourceEntity resE){
    freeResources.Add(resE);
  }

  protected override long scheduleMakeDecisions(){
    while (queuedJobs.Count != 0){
      SimpleJobEntity topJob = (SimpleJobEntity)queuedJobs[0];
      if (topJob.ResCount > freeResources.Count) break;
      List<ResourceEntity> allocatedResources =new List<ResourceEntity>();
      allocatedResources=freeResources.GetRange(0, topJob.ResCount);
      freeResources.RemoveRange(0, topJob.ResCount);
      allocatedRes[topJob] = allocatedResources;
      StartJob(topJob, allocatedResources);
      queuedJobs.Remove(topJob);
    }
    return 0;
  }

  protected override void handleJobProgressReport(JobEntity jobE, double progres){
    if (progres == 100){
      List<ResourceEntity> allocatedResources;
      allocatedResources = allocatedRes[jobE];
      freeResources.AddRange(allocatedRes[jobE]);
      foreach (ResourceEntity resE in allocatedRes[jobE])
        resE.setStateToIdle();
      allocatedRes.Remove(allocatedRes[jobE]);
      jobE.Remove();
    }
  }
}
```

Slika A.2 Implementacija PDP postupka raspoređivanja poslova (prema redoslijedu prispijeća)

Svaki postupak raspoređivanja ostvaruje se kao jedna od podvrsta osnovnog razreda *SchedulerEntity* pri čemu je nužno implementirati funkcije koje upravljaju dolaskom novog posla, aktivacijom resursa u sustavu, završetkom posla, te donošenjem odluka o raspoređivanju poslova. Lokalni podaci koji su ostvareni u PDP raspoređivaču poslova sa

slike uključuju popis poslova na čekanju (*queuedJobs*), popis raspoloživih resursa (*freeResources*) i rječnik koji preslikava poslove u resurse koji su im dodijeljeni (*allocatedRes*).

Upravljanje dolaskom novog posla ostvareno je funkcijom *queueJobHandling* koja na kraj popisa poslova na čekanju (*queuedJobs*) dodaje pridošli posao. Aktivacijom resursa se upravlja na sličan način, odnosno dodavanjem aktiviranog resursa u popis raspoloživih resursa (*freeResources*).

Donošenje odluke u raspoređivanju poslova najvažniji je dio svakog raspoređivača poslova koji se ostvaruje funkcijom *scheduleMakeDecisions*. Ta se funkcija poziva nakon svake interne promjene u raspoređivaču poslova, odnosno izravno nakon što se u određenom vremenskom trenutku potreban broj puta izvedu funkcije *queueJobHandling*, *handleActivatedResource* i *handleJobProgressReport*. Na taj način je spriječeno da se odluka o raspoređivanju poslova u nekom vremenskom trenutku donosi prije nego što se prikupi informacija o svim promjenama u poslovima i resursima u sustavu koje se dešavaju u istom vremenskom trenutku.

U funkciji *scheduleMakeDecisions* u primjeru sa slike A.2 se za vršni posao (*topJob*) u redu čekanja pokušavaju alocirati resursi. Ukoliko alokacija resursa ne uspije, raspoređivanje poslova završava. U slučaju da su slobodni resursi dostatni za izvršavanje vršnog posla, taj posao se pokreće na traženom broju resursa (*StartJob*) te se uklanja iz popisa poslova na čekanju (*queuedJobs*). Uspjelo pokretanje posla nastavljeno je pokušajem raspoređivanja slijedećeg vršnog posla. U slučaju da nema poslova na čekanju postupak raspoređivanja poslova ne provodi proces odlučivanja. Povratna vrijednost funkcije *scheduleMakeDecisions* označava vrijeme koje je potrebno pričekati prije provođenja odluke o raspoređivanju poslova. Kod brzih strategija raspoređivanja poslova, koje uključuju i PDP postupak raspoređivanja, nije potrebno posebno modelirati postupak donošenja i provođenja odluka s vremenskim pomakom pa je u ovom slučaju povratna vrijednost funkcije *scheduleMakeDecisions* jednaka nuli.

Upravljanje prijavom stupnja dovršenosti posla ostvareno je funkcijom *handleJobProgressReport*. Budući da PDP postupak kod donošenja odluka o raspoređivanju ne koristi informacije o stupnju dovršenosti posla, u funkciji *handleJobProgressReport* se zanemaruju sve prijave dovršenosti ukoliko posao nije završen. U slučaju završetka posla svi resursi dodijeljeni tom poslu se dodaju na popis slobodnih resursa (*freeResources*) i mijenja im se stanje u *Slobodan* pomoću funkcije *setStateToIdle*. Nakon oslobađanja resursa završeni posao se uklanja iz simulacije.

```
SimulationExecutive exc = new SimulationExecutive();
PDP_Scheduler schE = new PDP_Scheduler(exc);
SimpleResourceGenerator resGen = new SimpleResourceGenerator(exc, schE, 2);
resGen.GetResources();
RandomJobGenerator jobGen = new RandomJobGenerator(exc, schE,5,2,100, -0.1,0.1);
jobGen.start();
while (exc.step() != 0) ;
```

Slika A.3 Programsko ostvarenje simulacije temeljene na PDP postupku

Na primjeru sa slike A.2 zbog pojednostavljenja prikaza nije objašnjeno upravljanje pogreškama kod korisničkih ili sistemskih predikcija trajanja posla

U slučaju da se novi postupak raspoređivanja poslova ugrađuje u projekt *Simulator* definiranje cijele simulacije i upravljanje simulacijom omogućeno je iz grafičkog sučelja. Ukoliko se novi postupak raspoređivanja razvija u zasebnom projektu, moguće je iz njega pokrenuti grafičko sučelje *Upravitelja simulacije*, ali se može ostvariti i novi grafički ili komandno linijski primjenski program koji izvodi cijelu simulaciju nezavisno od *Upravitelja simulacija.* Primjer programskog isječka koji ostvaruje potpunu funkcionalnost diskretne simulacije bez oslanjanja na *Upravitelja simulacijama* prikazan je na slici A.3.

Postupak provođenja diskretne simulacije uključuje definiciju simulacije i izvršavanje simulacije. Kod definicije simulacije inicijalizira se razred *SimulationExecutive* koji ostvaruje jezgru simulatora. Nakon inicijalizacije jezgre simulatora u simulaciju se dodaju raspoređivač poslova (*PDP_Scheduler*), generator resursa (*SimpleResourceGenerator*) i generator poslova (*RandomJobGenerator*) te se inicijaliziraju generatori poslova i resursa. Generatori poslova i resursa korišteni u ovom primjeru dio su kolekcije dostupnih generatora, pri čemu je korisniku dozvoljeno stvaranje vlastitih generatora.

Izvođenje simulacije provodi se uzastopnim izvođenjem koraka trostupanjske jezgre simulacije pomoću funkcije *step*. Izvođenje simulacije završeno je u trenutku kad funkcija *step* vrati vrijednost nula, odnosno kada se unutar koraka simulacije ne izvede niti jedna aktivnost.

A.3. Upravljanje simulacijama

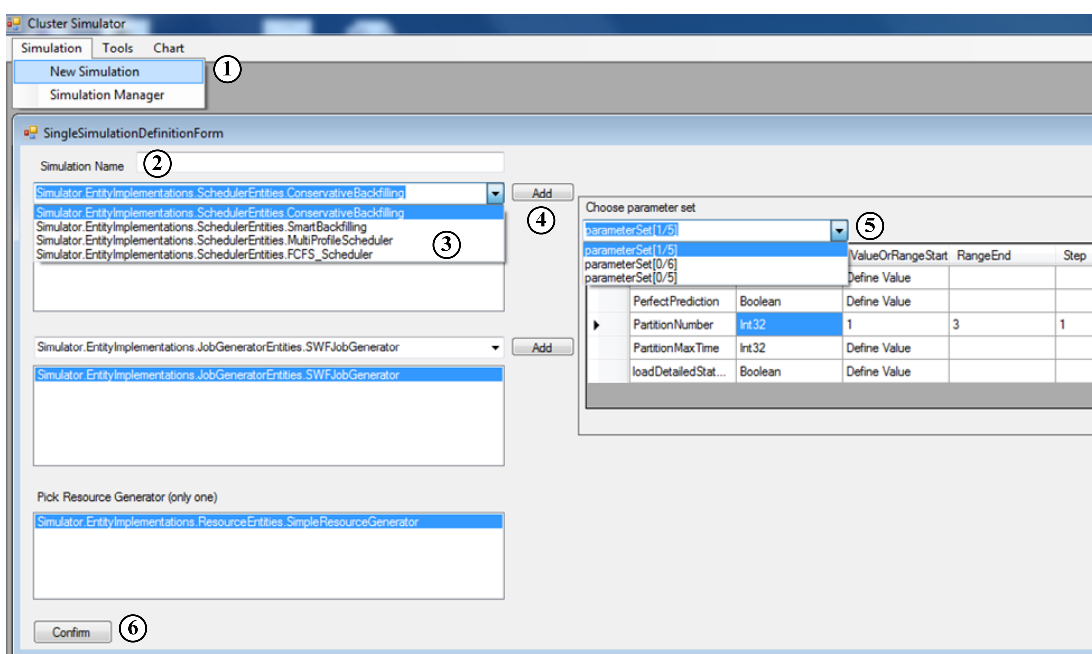
Upravljanje simulacijama, koje uključuje definiranje novih simulacija, ostvareno je komponentom *Upravitelj simulacija* koja čini dio projekta *Simulator*, te se pokreće odabirom primjenskog programa *Simulator.exe*. U ovom odjeljku dane su upute za korištenje većine funkcionalnosti *Upravitelja simulacijama*. Definiranje novih simulacija opisano je u odjeljku A.3.1, a izvođenje simulacija u odjeljku A.3.2. Postavke nužne za izvođenje simulacija na

udaljenim računalima prikazane su u odjeljku A.3.3. Analiziranje rezultata simulacija opisano je u odjeljku A.3.4.

A.3.1. Definiranje simulacija

Upravitelj simulacija omogućava olakšano definiranje jedne ili više simulacija u nekoliko jednostavnih koraka korištenjem grafičkog korisničkog sučelja. Da bi se otvorio prozor za definiranje simulacija potrebno je unutar primjenskog programa *Simulator.exe* u glavnom izborniku odabrati *Simulation→New Simulation* kako je prikazano na slici A.4 (1). Odabirom te opcije otvara se novi prozor naziva *SingleSimulationDefinitionForm* u kojem je potrebno odabrati naziv simulacije, raspoređivače poslova, te generatore poslova i resursa koji će se koristiti u simulaciji.

Na početku definicije skupa simulacija potrebno je navesti naziv skupa u polju označenom "*Simulation Name*" (2) čemu slijedi odabir raspoređivača poslova. Raspoređivač poslova odabire se iz padajuće liste (3) u kojoj su ponuđena imena svih razreda koji su izvedeni iz razreda *SchedulerEntity*. Odabir raspoređivača poslova potvrđuje se pritiskom na odgovarajuće dugme s oznakom "*Add*". Nakon odabira raspoređivača poslova potrebno je inicijalizirati njegove parametre. Parametri raspoređivača poslova postavljaju se kroz konstruktor razreda koji implementira odabrani raspoređivač poslova. Budući da neki raspoređivači poslova imaju više konstruktora, kod odabira parametara potrebno je prvo



Slika A.4 Definiranje nove simulacije pomoću *Upravitelja Simulacija*

odabrati konstruktor, a nakon toga definirati sve parametre (5). Odabir konstruktora omogućen je u padajućoj listi s oznakom "*Choose parameter set*". Nakon odabira konstruktora potrebno je definirati vrijednosti svih parametara u tablici koja se nalazi ispod padajuće liste.

Kod odabira vrijednosti parametara uz ime i tip svakog parametra potrebno je navesti njegovu vrijednost ili raspon vrijednosti. Raspon vrijednosti definiran je početkom i krajem intervala, te korakom uvećanja. Odabir raspona vrijednosti omogućen je samo za numeričke tipove podataka.

Odabir generatora poslova i generatora resursa, te postavljanje njihovih parametara identično je postupku opisanom kod raspoređivača poslova. Nakon odabira svih komponenti simulacije potrebno je pritisnuti dugme "*Confirm*" (6) pri čemu se generira skup od jedne ili više simulacija. U slučaju da je u postupku definiranja simulacije odabrano više različitih raspoređivača poslova ili generatora poslova, ili da je kod postavljanja parametara nekog od simulacijskih entiteta odabran raspon vrijednosti generira se veći broj simulacija.

Kod generiranja većeg broja simulacija svaki od definiranih raspoređivača poslova se kombinira sa svim generatorima poslova i generatorima resursa pri čemu svaka kombinacija čini jednu simulaciju. Sve generirane simulacije pohranjuju se mapu s imenom simulacije u obliku XML datoteka. Primjer generirane XML datoteke koja opisuje jednu simulaciju prikazan je na slici A.5.

Na početku definicije simulacije nalazi se naziv skupa simulacija *"jednostavneHeuristike"* i jedinstveni identifikator simulacije (*guid*). Naziv XML datoteke jednak je jedinstvenom identifikatoru simulacije. XML element *SimulationStatusType* određuje stanje simulacije. Simulacija može biti inicijalizirana (*idle*), pokrenuta na lokalnom računalu (*startedLocally*), pripravna u redu čekanja udaljenog računala (*queuedRemote*), pokrenuta na udaljenom računalu (*startedRemote*) i završena (*finished*).

Nakon osnovnih podataka o simulaciji slijedi XML Element *EntityConstructorInformation* u kojem se nalaze opisnici konstruktora i parametara svih entiteta koji se koriste u simulaciji. U primjeru na slici A.5 odabran je *FCFS_Scheduler* raspoređivač poslova s parametrom *prioritizeType* postavljenim na vrijednost 1. Parametar naziva *exc* označava pridruženu jezgru simulatora i samo se zbog potpunosti nalazi u XML datoteci budući da se kod izvođenja simulacije stvara nova jezgra simulatora.

XML datoteka završava rezultatima simulacije, odnosno vrijednostima svih metrika u različitim vremenskim trenucima koji se dopisuju nakon završetka cijele simulacije. XML element *SchedulerSingleMetric* određuje područje u kojem se nalaze svi podaci vezani za praćenje jedne metrike. Na slici A.5. prikazano je praćenje metrike *Utilization*, koja se nalazi

```
<?xml version="1.0" encoding="utf-8"?>
<SimulationConstructor>
  <string>jednostavneHeuristike</string>
  <guid>0bf942c5-44a3-4df0-9da5-16504f67406b</guid>
  <SimulationStatusType>finished</SimulationStatusType>
  <EntityConstructorInformation>
    <string>Simulator.EntityImplementations.SchedulerEntities.FCFS_Scheduler,
         Simulator,Version=1.0.0.0,Culture=neutral,PublicKeyToken=null</string>
    <ArrayOfEntityConstructorParameter
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <EntityConstructorParameter>
        <string>exc</string>
        <string>SimulatorEngine.SimulationExecutive, SimulatorEngine,
                Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</string>
      </EntityConstructorParameter>
      <EntityConstructorParameter>
        <string>prioritizeType</string>
        <string>System.Int32, mscorlib, Version=2.0.0.0, Culture=neutral,
                PublicKeyToken=b77a5c561934e089</string>
        <int>1</int>
      </EntityConstructorParameter>
    </ArrayOfEntityConstructorParameter>
  </EntityConstructorInformation>
...
<SchedulerMetrics>
    <ArrayOfSchedulerSingleMetric>
<SchedulerSingleMetric>
   <anyType xsi:type="xsd:string">Utilization</anyType>
   <anyType xsi:type="xsd:double">0.78839200075685167</anyType>
   <ArrayOfMetricHistory>
       <MetricHistory>
          <clock>0</clock>
          <metricValue>0</metricValue>
       </MetricHistory>
       <MetricHistory>
          <clock>1975598</clock>
          <metricValue>0.63749538085775459</metricValue>
       </MetricHistory>
…
```

Slika A.5 XML datoteka sa opisom jedne simulacije

iza vrijednosti koju je metrika ostvarila na kraju simulacije. U XML elementu *ArrayOfMetricHistory* nalazi se povijest metrike *Utilization* u različitim trenucima simulacije.

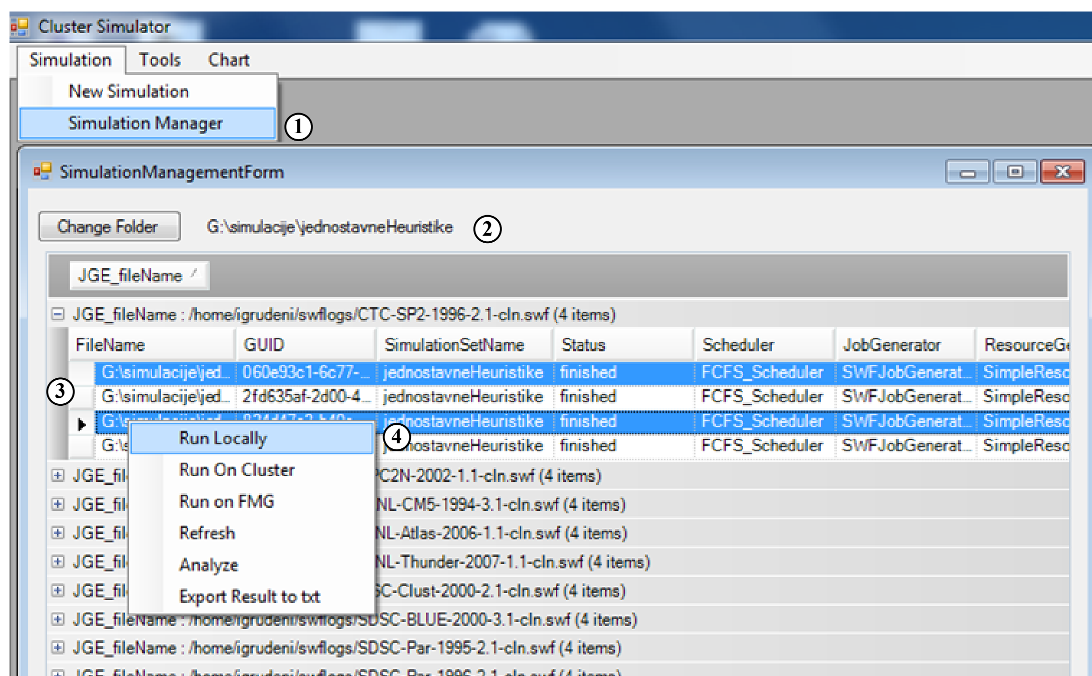
Budući da su simulacije spremljene u XML datotekama jednostavno ih je prebacivati između više računala, mijenjati, te izvoditi njima definirane simulacije na više računala.

A.3.2 Izvođenje simulacija

Izvođenje simulacija grozda računala omogućeno je primjenskim programom *Simulator.exe* koji ostvaruje komponentu *Upravitelj simulacija*. Izvođenje simulacija provodi se iz grafičkog sučelja koje omogućava i pregled postojećih simulacija i njihovih rezultata. Da bi se otvorio prozor iz kojeg je omogućen pregled i izvođenje simulacija, potrebno je iz glavnog menija odabrati opciju *Simulation→Simulation Manager* kako je prikazano na slici A.6 (1).

Odabirom te opcije otvara se prozor *SimulationManagementForm*. Nakon otvaranja prozora potrebno je navesti mapu u kojoj se nalaze definirane simulacije pomoću dugmeta "*Change Folder*" (2). Odabir mape uzrokuje automatsku analizu svih datoteka u dotičnoj mapi i njoj pridruženim mapama te generiranje popisa svih simulacija, njima pripadajućih parametara, te konačnih rezultata svih praćenih metrika učinkovitosti raspoređivača poslova za završene simulacije.

Da bi se obavila neka radnja nad jednom ili više simulacija potrebno ih je prethodno označiti. Označavanje skupa simulacija provodi se držanjem tipke *CTRL* i odabirom simulacija pomoću lijeve tipke miša (3). Nakon odabira simulacija potrebno je nad nekom od

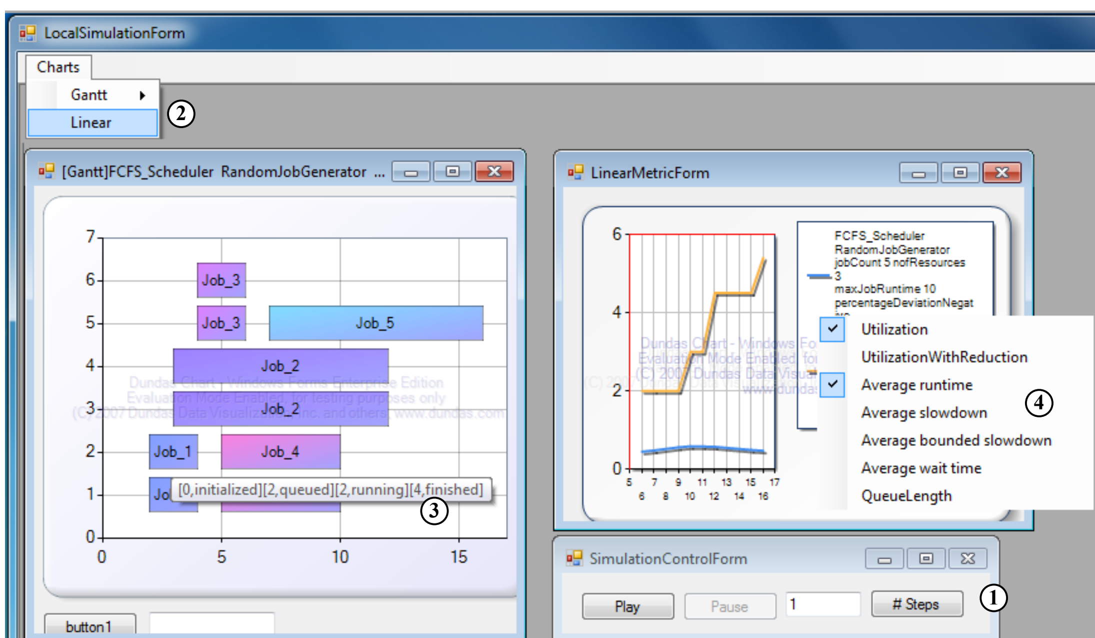


Slika A.6 Pregled i pokretanje simulacija pomoću *Upravitelja Simulacija*

odabranih simulacija pritisnuti desnu tipku miša pri čemu se dobiva kontekstni izbornik (4). U kontekstnom izborniku se nudi više različitih operacija koje je moguće izvesti nad odabranim simulacijama. Odabirom opcije *RunLocally* pokreće se skup odabranih simulacija na lokalnom računalu. Simulacije se mogu pokrenuti i na udaljenom računalu pri čemu je takvo izvršavanje omogućeno opcijama *Run On Cluster* i *Run on FMG*. Opcija *Run On Cluster* pokreće skup simulacija na računalnom grozdu Isabella [88], dok odabir opcije *Run On FMG* pokreće simulaciju na predefiniranom poslužitelju. Postupak konfiguracije sustava koji omogućava promjenu udaljenih računala na kojima će se vršiti simulacija prikazan je u odjeljku A.3.3.

Opcija *Refresh* omogućava ponovno čitanje podataka o skupu simulacija što je posebno korisno kod provjere stanja udaljenih i lokalno pokrenutih simulacija. Usporedna analiza odabranog skupa simulacija provodi se odabirom opcije *Analyze*. Detaljni opis usporedne analize podataka prikazan je u odjeljku A.3.4. U slučaju da se označeni podaci žele analizirati u nekom drugom primjenskom programu omogućena je pretvorba podataka označenog simulacijskog skupa u tekstualnu datoteku. Pretvorba se provodi odabirom opcije *Export Results to txt*.

Kod izvođenja lokalnih simulacija odabirom opcije *Run Locally* za svaku od simulacija u odabranom simulacijskom skupu otvara se prozor *LocalSimulationForm* kako je prikazano na slici A.7. Upravljanje tokom simulacije omogućeno je skupom kontrola u prozoru *SimulationControlForm* (1). Odabirom dugmeta *Play* pokreće se simulacija koju je moguće



Slika A.7 Izvođenje lokalnih simulacija

privremeno zaustaviti odabirom dugmeta *Pause*. Određeni broj koraka simulacije moguće je izvršiti pritiskom na dugme *#Steps*, pri čemu se broj koraka upisuje u polje koje prethodi tom dugmetu.

Praćenje toka simulacije omogućeno je pomoću dvije vrste grafikona koje je moguće uključiti odabirom glavnog izbornika *Charts* (2). Odabir opcije *Gantt* otvara prozor *[Gantt]* u kojem je prikazan ganttov dijagram, pri čemu se na osi apscise nalazi vrijeme, a na osi ordinate resursi grozda računala. U samom prostoru dijagrama iscrtani su poslovi koji zauzimaju određenu površinu ovisno o trajanju i količini potrebnih resursa. Detaljnije podatke o nekom poslu poput trenutka dolaska posla u sustav, početka izvršavanja posla, te trajanja posla moguće je dobiti prelaskom pokazivača preko određenog posla (3).

Uz ganttov dijagram, moguće je pratiti promjenu različitih metrika u sustavu linearnim dijagramom koji je prikazan u prozoru *LinearMetricForm*. Odabir metrika koje je potrebno usporedno pratiti provodi se pritiskom na desnu tipku miša, te označavanjem potrebnih metrika u kontekstnom izborniku (4).

A.3.3 Simulacije na udaljenim računalima

Izvršavanje simulacija na udaljenim računalima uključuje prebacivanje opisnika simulacija na udaljeno računalo, pokretanje simulacije na udaljenom računalu, te dohvat rezultata simulacije na lokalno računalo. Cijeli niz od tri operacije izvršava se automatski iz grafičkog sučelja *Upravitelja simulacija*, međutim potrebno je prethodno ispravno podesiti udaljena računala i *Upravitelja simulacija*.

Podešavanje *Upravitelja simulacija* provodi se u konfiguracijskoj datoteci *app.config* projekta *Simulator*. Budući da *Upravitelj simulacija* podržava rad s dva različita udaljena sustava koji se konfiguriraju na isti način u ovom odjeljku opisana je konfiguracija samo jednog od njih. Primjer isječka konfiguracijske datoteke koji sadrži konfiguraciju *Upravitelja simulacija* za rad s udaljenim sustavom dan je na slici A.8.

XML element *sftpCopyExecutable* koristi se za definiciju primjenskog programa kojim će se prebacivati XML opisnici simulacija s lokalnog na udaljeno računalo. Parametri tog programa nužni za prebacivanje podataka na udaljeno računalo i vraćanje podataka na lokalno računalo podešavaju se unutar XML elemenata *sftpCopyToClusterArguments* i *sftpCopyFromClusterArguments*. Prilikom podešavanja tih elemenata izrazi *LOCALFILE i REMOTEFILE* označavaju imena XML datoteke na lokalnom i udaljenom računalu koja generira *Upravitelj simulacija* na temelju imena datoteka simulacija koje je potrebno prenositi.

```
<setting name="sftpCopyExecutable" serializeAs="String">
  <value>"C:\Program Files (x86)\PuTTY\pscp.exe"</value>
</setting>
<setting name="sftpCopyToClusterArguments" serializeAs="String">
  <value>-l igrudeni -pw ****** "LOCALFILE"
         tannat.srce.hr:/home/igrudeni/simulacije/REMOTEFILE
  </value>
</setting>
<setting name="sftpCopyFromClusterArguments" serializeAs="String">
  <value>-l igrudeni -pw ******
          tannat.srce.hr:/home/igrudeni/simulacije/REMOTEFILE  "LOCALFILE"
  </value>
</setting>
<setting name="plinkExecuteQsubParameters" serializeAs="String">
  <value>tannat.srce.hr -l igrudeni -pw ****** qsub -o
         ~/simulationOutputs/FILENAME.out -e ~/simulationOutputs/FILENAME.err
         -v xmlfile=~/simulacije/FILENAME  simskripta.sh
  </value>
</setting>
<setting name="plinkExecutable" serializeAs="String">
  <value>"C:\Program Files (x86)\PuTTY\plink.exe"</value>
</setting>
```

Slika A.8 XML datoteka sa konfiguracijom *Upravitelja simulacija* za rad sa udaljenim sustavom

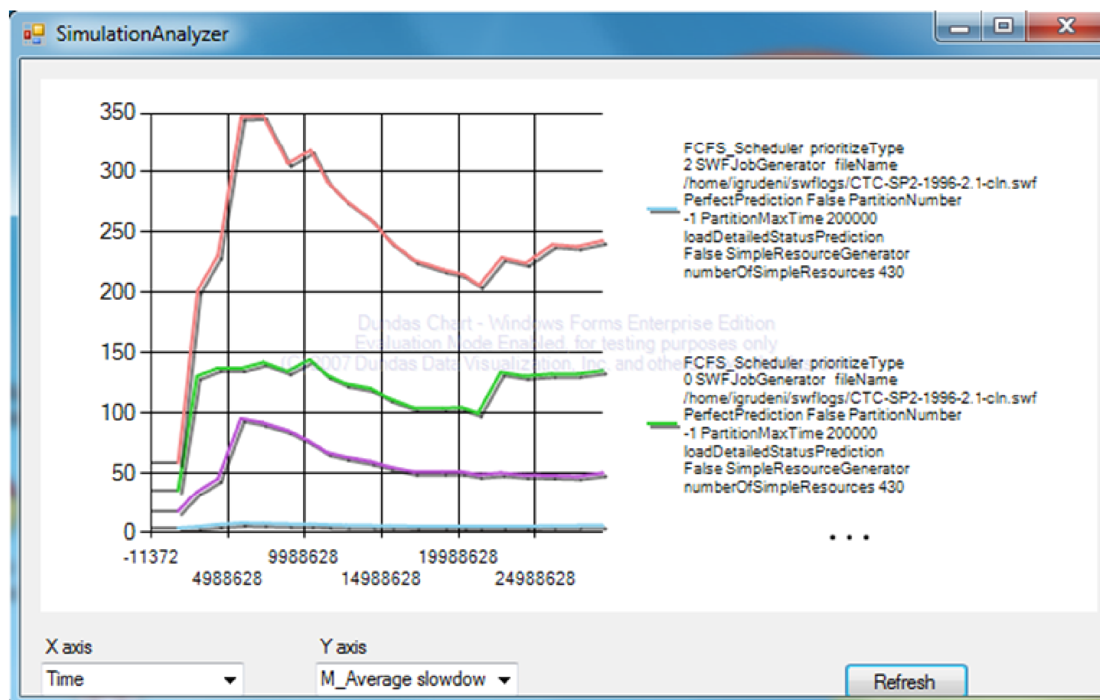
Pokretanje poslova na udaljenom računalu odvija se na način definiran XML elementima *plinkExecutable* i *plinkExecuteQsubParameters*. XML element *plinkExecutable* označava naziv primjenskog programa koji se koristi da bi se *Upravitelj simulacije* spojio na udaljeno računalo. Svi parametri potrebni za izvođenje udaljene simulacije koji se predaju programu namijenjenom za spajanje na udaljeno računalo definiraju se u XML elementu *plinkExecuteQsubParameters*.

Primjer na slici A.8. prikazuje konfiguraciju namijenjenu izvršavanju poslova na računalnom grozdu Isabella s pristupnim računalom *tannat.srce.hr*.

A.3.4 Analiza rezultata simulacija

Analiza rezultata simulacija jedna je od funkcionalnosti *Upravitelja simulacijama*. Nakon definiranja skupa simulacija u *SimulationManagementForm* prozoru, te odabira opcije *Analyze* u kontekstnom izborniku na ekranu se prikazuje prozor *SimulationAnalyzer* kako je prikazano na slici A.9.

Različite komponente simulacija kao što su simulacijski entiteti, parametri simulacijskih entiteta i vrijeme mogu se staviti na osi grafikona prikazanog na slici. Vrsta dijagrama na slici se mijenja ovisno o podacima koje je potrebno prikazati. Podatke koji se postavljaju na



Slika A.9 Analiza rezultata simulacija komponentom *Upravitelj Simulacija*

osi apscisa i osi ordinata određuje korisnik odabirom iz padajućih listi označenih sa *"X axis"* i *"Y axis"*. Nakon promjene podataka u padajućim listama potrebno je pritisnuti dugme *Refresh*

Legenda s desne strane dijagrama prikazuje sve detalje o simulacijama čiji parametri se analiziraju na dijagramu.

DODATAK B – Programsko sučelje SARG sustava

U ovome dodatku prikazano je programsko sučelje najznačajnijih dijelova SARG sustava koji su dostatni za izgradnju programskog rješenja vlastite simulacije grozda računala. Prikazano sučelje uključuje komponente *Jezgru Simulatora*, *Predloške simulacijskih entiteta za simulaciju grozda računala* i *Ostvarene entitete za simulaciju grozda računala*. Komponente vezane uz upravljanje simulacijama grafičkim sučeljem i prediktivnu statističku analizu izostavljene su iz ovog prikaza.

Sadržaj dodatka B

Stablo razreda

Stablo naslijeđivanja je složeno približno po abecedi:

SimulatorEngine::Activities::Activity ... 123
SimulatorEngine::Activities::ActivityB ... 123
SimulatorEngine::Activities::ActivityC ... 124
SimulatorEngine::ActivityCHandler ... 124
SimulatorEngine::ActivityControl ... 125
SimulatorEngine::Entities::SimulationEntity ... 141
SimulatorEngine::Entities::JobEntity ... 129
Simulator::EntityImplementations::JobEntities::SimpleJobEntity ... 140
Simulator::EntityImplementations::JobEntities::BackfillingJobEntity ... 125
SimulatorEngine::Entities::JobGeneratorEntity ... 131
Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator ... 134
Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGenerator ... 145
Simulator::EntityImplementations::JobGeneratorEntities::TopologyJobGenerator ... 146
SimulatorEngine::Entities::ResourceEntity ... 135
Simulator::EntityImplementations::ResourceEntities::SimpleResourceEntity ... 141
SimulatorEngine::Entities::SchedulerEntity ... 136
Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling ... 127
Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler ... 128
Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler ... 133
Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling ... 144
SimulatorEngine::SimulationExecutive ... 142
SimulatorEngine::LazyCollection-g< T > ... 132
Simulator::EntityImplementations::ResourceEntities::ResourceGenerator ... 136
Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator ... 141

Dokumentacija razreda

Opis razreda SimulatorEngine::Activities::Activity

Implementira osnovni skup funkcionalnosti koje svaka aktivnost u simulaciji treba ostvariti.
Naslijeđena u SimulatorEngine::Activities::ActivityBSimulatorEngine::Activities::ActivityC.

Public članovi

- Activity (ActivityControl parent)
- abstract void **Action** ()
- void Remove ()
- long getClock ()

Protected članovi

- void addActivity ()

Svojstva (property)

- Boolean assignedToActCtrl `[get]`

Detaljno objašnjenje

Implementira osnovni skup funkcionalnosti koje svaka aktivnost u simulaciji treba ostvariti.

Dokumentacija konstruktora i destruktora

SimulatorEngine::Activities::Activity::Activity (ActivityControl parent) `[inline]`

Inicijalizira aktivnost.

Parametri:

parent Objekt koji će voditi brigu o pravovremenom izvođenju aktivnosti

Dokumentacija funkcija

long SimulatorEngine::Activities::Activity::getClock () `[inline]`

Trenutno vrijeme simulacije.

Povratne vrijednosti:

trenutno vrijeme simulacije

void SimulatorEngine::Activities::Activity::Remove () `[inline]`

Uklanjanje aktivnosti iz simulacije.

Dokumentacija svojstava

Boolean SimulatorEngine::Activities::Activity::assignedToActCtrl `[get]`

Određuje da li aktivnost ima pridružen objekt koji se brine o njenom izvršavanju.

Opis razreda SimulatorEngine::Activities::ActivityB

Ostvaruje funkcionalnost fiksiranih aktivnosti u trostupanjskoj diskretnoj simulaciji.
Naslijeđuje od SimulatorEngine::Activities::Activity.
Naslijeđena u Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::EstimatedRuntimeElapsedB, Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::SystemPredictedRuntimeElapsedB, SimulatorEngine::Entities::JobEntity::ProgressReportB, SimulatorEngine::Entities::JobEntity::QueueJobB, SimulatorEngine::Entities::JobGeneratorEntity::QueueNextJobB, SimulatorEngine::Entities::ResourceEntity::ActivateResourceB, SimulatorEngine::Entities::ResourceEntity::BackInServiceB, SimulatorEngine::Entities::ResourceEntity::OutOfServiceBSimulatorEngine::Entities::SchedulerEntity::ThinkingB.

Public članovi

- ActivityB (long scheduledTime, Boolean relative, SimulationEntity parent)

Protected atributi

- long schTime

Svojstva (property)

- long scheduledTime `[get]`

Detaljno objašnjenje

Ostvaruje funkcionalnost fiksiranih aktivnosti u trostupanjskoj diskretnoj simulaciji.

Dokumentacija konstruktora i destruktora

SimulatorEngine::Activities::ActivityB::ActivityB (long scheduledTime*, Boolean* relative*,* SimulationEntity parent*)* `[inline]`

Inicijalizira fiksiranu aktivnost.

Parametri:

scheduledTime Vrijeme u koje se aktivnost treba izvesti
relative Određuje da li parametar scheduledTime označava apsolutno vijeme ili relativni pomak u odnosu na sadašnji trenutak.
parent Objekt koji vodi brigu o pravovremenom izvođenju aktivnosti

Dokumentacija svojstava

long SimulatorEngine::Activities::ActivityB::scheduledTime `[get]`

Vrijeme za koje je zakazano izvođenje aktivnosti.

Opis razreda SimulatorEngine::Activities::ActivityC

Implementira uvjetne aktivnosti u trostupanjskoj diskretnoj simulaciji.
Naslijeđuje od SimulatorEngine::Activities::Activity.
Naslijeđena u SimulatorEngine::Entities::SchedulerEntity::RescheduleC.

Public članovi

- ActivityC (ActivityControl parent)
- abstract bool Condition ()

Detaljno objašnjenje

Implementira uvjetne aktivnosti u trostupanjskoj diskretnoj simulaciji.

Dokumentacija konstruktora i destruktora

*SimulatorEngine::Activities::ActivityC::ActivityC (*ActivityControl parent*)* `[inline]`

Inicijalizira uvjetnu aktivnost.

Parametri:

parent Objekt koji vodi brigu o pravovremenom izvođenju aktivnosti

Dokumentacija funkcija

abstract bool SimulatorEngine::Activities::ActivityC::Condition () `[pure virtual]`

Ostvaruje uvjet koji je potrebno ispuniti da bi se aktivnost izvela.

Povratne vrijednosti:

true, ako je uvjet ostvaren

Opis razreda SimulatorEngine::ActivityCHandler

Upravlja skupom uvjetnih aktivnosti.

Public članovi

- int findAndExecuteConditionalActivities ()
- void deleteActivity (ActivityC activity)
- void addActivity (ActivityC activity)

Protected atributi

- LazyCollection< Activity > **activities** = new LazyCollection<Activity>()

Svojstva (property)

- int Count `[get]`

Detaljno objašnjenje

Upravlja skupom uvjetnih aktivnosti.

Dokumentacija funkcija

*void SimulatorEngine::ActivityCHandler::addActivity (*ActivityC activity*)* `[inline]`

Dodaje aktivnost u skup aktivnosti.

Parametri:

activity Aktivnost koja se dodaje.

void SimulatorEngine::ActivityCHandler::deleteActivity (ActivityC activity) `[inline]`

Briše aktivnost iz skupa aktivnosti.

Parametri:

activity Aktivnost koja se briše

int SimulatorEngine::ActivityCHandler::findAndExecuteConditionalActivities () `[inline]`

Pronalazi i izvršava sve uvjetne aktivnosti za koje je uvjet zadovoljen.

Povratne vrijednosti:

Broj izvršenih uvjetnih aktivnosti.

Dokumentacija svojstava

int SimulatorEngine::ActivityCHandler::Count `[get]`

Broj aktivnosti u skupu.

Opis sučelja (interface) SimulatorEngine::ActivityControl

Sučelje koje trebaju ostvariti svi objekti koji upravljaju, odnosno brinu za izvođenje aktivnosti.
Naslijeđena u SimulatorEngine::Entities::SimulationEntitySimulatorEngine::SimulationExecutive.

Public članovi

- void deleteActivity (Activity activity)
- void addActivity (Activity activity)
- long getClock ()
- Boolean simulationStarted ()

Detaljno objašnjenje

Sučelje koje trebaju ostvariti svi objekti koji upravljaju, odnosno brinu za izvođenje aktivnosti.

Dokumentacija funkcija

void SimulatorEngine::ActivityControl::addActivity (Activity activity)

Dodaje/aktivira aktivnost.

Parametri:

activity Aktivnost koja se dodaje.

Implementirano u SimulatorEngine::Entities::SimulationEntitySimulatorEngine::SimulationExecutive.

void SimulatorEngine::ActivityControl::deleteActivity (Activity activity)

Briše/deaktivira aktivnost.

Parametri:

activity Aktivnost koja se briše.

Implementirano u SimulatorEngine::Entities::SimulationEntitySimulatorEngine::SimulationExecutive.

long SimulatorEngine::ActivityControl::getClock ()

Trenutno vrijeme simulacije.

Implementirano u SimulatorEngine::Entities::SimulationEntitySimulatorEngine::SimulationExecutive.

Boolean SimulatorEngine::ActivityControl::simulationStarted ()

Označava da li je simulacija započela sa izvođenjem.

Implementirano u SimulatorEngine::Entities::SimulationEntitySimulatorEngine::SimulationExecutive.

Opis razreda Simulator::EntityImplementations::JobEntities::BackfillingJobEntity

Nasljeđuje od Simulator::EntityImplementations::JobEntities::SimpleJobEntity.

Strukture

- class EstimatedRuntimeElapsedB
- class SystemPredictedRuntimeElapsedB

Public članovi

- BackfillingJobEntity (SimulationExecutive exc, SchedulerEntity schE, long duration, long userEstimatedDuration, int resCount, String name)

- BackfillingJobEntity (SimulationExecutive exc, SchedulerEntity schE, long duration, long userEstimatedDuration, int resCount, String name, int userId, int applicationId)
- void DefineStatusPredictions (PredictionItem statusPrediction)
- void DefineRuntimePredictions (PredictionItem runtimePrediction)
- override void start ()
- override void start (long SystemRuntimePrediction)
- override string ToString ()

Public atributi

- long SchedulerEstimatedStartTime

Svojstva (property)

- int UserID `[get]`
- int ApplicationID `[get]`
- PredictionItem StatusPrediction `[get]`
- PredictionItem RuntimePrediction `[get]`
- long UserEstimatedDuration `[get]`
- override string Name `[get]`

Dokumentacija konstruktora i destruktora

*Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::BackfillingJobEntity (*SimulationExecutive exc*,* SchedulerEntity schE*, long* duration*, long* userEstimatedDuration*, int* resCount*, String* name*)* `[inline]`

Inicijalizira novi entitet posla.

Parametri:

exc Jezgra simulatora kojoj posao pripada
schE Raspoređivač poslova zadužen za posao
duration Trajanje posla
userEstimatedDuration Korisnička procjena trajanja posla
resCount Količina resursa potrebna za izvršavanje posla
name Naziv posla

*Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::BackfillingJobEntity (*SimulationExecutive exc*,* SchedulerEntity schE*, long* duration*, long* userEstimatedDuration*, int* resCount*, String* name*, int* userId*, int* applicationId*)* `[inline]`

Inicijalizira novi entitet posla.

Parametri:

exc Jezgra simulatora kojoj posao pripada
schE Raspoređivač poslova zadužen za posao
duration Trajanje posla
userEstimatedDuration Korisnička procjena trajanja posla
resCount Količina resursa potrebna za izvršavanje posla
name Naziv posla
userId Identifikator korisnika koji je stvorio posao
applicationId Identifikator primjenskog programa

Dokumentacija funkcija

void Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::DefineRuntimePredictions (PredictionItem runtimePrediction*)* `[inline]`

Definira sistemski određenu predikciju trajanja posla.

Parametri:

runtimePrediction Predviđeno trajanje

void Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::DefineStatusPredictions (PredictionItem statusPrediction*)* `[inline]`

Definira sistemski određenu predikciju načina završetka (statusa) posla.

Parametri:

statusPrediction Predviđeni status

override void Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::start (long SystemRuntimePrediction*)* `[inline, virtual]`

Pokreće posao, uz davanje sistemski generirane predikcije njegovog trajanja. U slučaju da je sistemska predikcija manja od stvarnog trajanja posla i korisničke procjene trajanja posla nakon isteka sistemske procjene generira se novi događaj za obradu u raspoređivaču poslova.

Parametri:

SystemRuntimePrediction Trajanje sistemski određenog vremena izvođenja posla.

Reimplementirano od SimulatorEngine::Entities::JobEntity.

override void Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::start () `[inline, virtual]`

Pokreće posao, pri čemu je trajanje posla jednako manjoj od vrijednosti {korisnička procjena trajanja posla, trajanje posla}.

Reimplementirano od SimulatorEngine::Entities::JobEntity.

override string Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::ToString () `[inline]`

Vraća osnovne podatke o poslu.

Dokumentacija svojstava

int Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::ApplicationID `[get]`

Identifikator primjenskog programa.

override string Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::Name `[get]`

Naziv posla.

Reimplementirano od SimulatorEngine::Entities::SimulationEntity.

PredictionItem Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::RuntimePrediction `[get]`

Sistemski predviđeno trajanje posla.

PredictionItem Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::StatusPrediction `[get]`

Sistemski predviđeni status posla.

long Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::UserEstimatedDuration `[get]`

Korisnička procjena trajanja posla.

int Simulator::EntityImplementations::JobEntities::BackfillingJobEntity::UserID `[get]`

Identifikator vlasnika posla.

Opis razreda Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling

Ostvaruje postupak raspoređivanja poslova na grozdu računala temeljem unazadnog popunjavanja praznina uz konfigurabilni broj rezervacija, mogućnost predviđanja statusa i trajanja poslova.

Nasljeđuje od SimulatorEngine::Entities::SchedulerEntity.

Strukture

- class TwoRuntimes

Public članovi

- ConservativeBackfilling (SimulationExecutive exc, long maxReservations, Boolean UseBestFit, Boolean ForceFailed, int PredictRuntimes)
- override void queueJobHandling (JobEntity job)
- override string ToString ()

Protected članovi

- override long scheduleMakeDecisions ()
- override void handleJobProgressReport (JobEntity jobE, double progress)
- override void handleActivatedResurce (ResourceEntity resE)
- override void handleJobEstimatedTimeRanOut (JobEntity jobE)
- override void handleJobSystemPredictedRuntimeRanOut (JobEntity jobE)

Detaljno objašnjenje

Ostvaruje postupak raspoređivanja poslova na grozdu računala temeljem unazadnog popunjavanja praznina uz konfigurabilni broj rezervacija, mogućnost predviđanja statusa i trajanja poslova.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::ConservativeBackfilling (SimulationExecutive exc*, long* maxReservations*, Boolean* UseBestFit*, Boolean* ForceFailed*, int* PredictRuntimes*)* `[inline]`

Parametri:

exc Jezgra simulatora u kojoj se izvodi raspoređivač poslova.
maxReservations Maksimalni broj dozvoljenih rezervacija
UseBestFit true, ako je kod popunjavanja praznina potrebno koristiti strategiju best fit (false za first fit).
ForceFailed true, za podizanje prioriteta poslova koji će vjerojatno završiti sa pogreškom
PredictRuntimes Način predviđanja vremena izvođenja poslova. 0 za korisničke procjene, 1 za korištenje bayesovih mreža kod predviđanja trajanja posla, 2 za korištenje prosjeka zadnja 2 slična posla.

Dokumentacija funkcija

override void Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::handleActivatedResurce (ResourceEntity resE*)* `[inline, protected, virtual]`

Obrađuje aktivaciju novog resursa.

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::handleJobEstimatedTimeRanOut (JobEntity jobE*)* `[inline, protected, virtual]`

Obrađuje istek korisničke procjene trajanja posla.

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::handleJobProgressReport (JobEntity jobE*, double* progress*)* `[inline, protected, virtual]`

Obrađuje prijavu stupnja dovršenosti posla (podržana samo 100tna dovršenost).

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::handleJobSystemPredictedRuntimeRanOut (JobEntity jobE*)* `[inline, protected, virtual]`

Obrađuje istek sistemske procjene trajanja posla.

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::queueJobHandling (JobEntity job*)* `[inline, virtual]`

Obrađuje dolazak novog posla.

Implementira SimulatorEngine::Entities::SchedulerEntity.

override long Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::scheduleMakeDecisions () `[inline, protected, virtual]`

Implementira donošenje odluka o raspoređivanju poslova.

Povratne vrijednosti:

uvijek 0 (modelira trenutno donošenje odluka)

Implementira SimulatorEngine::Entities::SchedulerEntity.

override string Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling::ToString () `[inline]`

Daje opis temeljnih značajki raspoređivača poslova.

Opis razreda Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler

Ostvaruje nekoliko jednostavnih heuristika raspoređivanja poslova temeljenih na izvršavanju posla sa najvećim prioritetom (PDP).
Nasljeđuje od SimulatorEngine::Entities::SchedulerEntity.

Public članovi

- FCFS_Scheduler (SimulationExecutive exc, int prioritizeType)
- override void queueJobHandling (JobEntity jobE)
- override string ToString ()

Protected članovi

- override long scheduleMakeDecisions ()
- override void handleJobProgressReport (JobEntity jobE, double progress)
- override void handleActivatedResurce (ResourceEntity resE)
- override void handleJobEstimatedTimeRanOut (JobEntity jobE)
- override void handleJobSystemPredictedRuntimeRanOut (JobEntity jobE)

Detaljno objašnjenje

Ostvaruje nekoliko jednostavnih heuristika raspoređivanja poslova temeljenih na izvršavanju posla sa najvećim prioritetom (PDP).

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::FCFS_Scheduler (SimulationExecutive exc, *int* prioritizeType*)* `[inline]`

Inicijalizira raspoređivač poslova.

Parametri:

exc Jezgra simulatora odgovorna za izvođenje raspoređivača poslova.
prioritizeType Odabir postupka raspoređivanja poslova: 0 - Prema redoslijedu dospijeća (FCFS) 1 - Postupak prioritiziranja najkraćeg posla (PNP) 2 - Postupak prioritiziranja najužeg posla (PUP) 3 - Postupak prioritiziranja najšireg posla (PŠP)

Dokumentacija funkcija

override void Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::handleActivatedResurce (ResourceEntity resE*)* `[inline, protected, virtual]`

Obrađuje aktivaciju novog resursa.

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::handleJobEstimatedTimeRanOut (JobEntity jobE*)* `[inline, protected, virtual]`

Nije implementirano (vraća iznimku).

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::handleJobProgressReport (JobEntity jobE, *double* progress*)* `[inline, protected, virtual]`

Obrađuje prijavu stupnja dovršenosti posla (podržana samo 100tna dovršenost).

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::handleJobSystemPredictedRuntimeRanOut (JobEntity jobE*)* `[inline, protected, virtual]`

Nije implementirano (vraća iznimku).

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::queueJobHandling (JobEntity jobE*)* `[inline, virtual]`

Obrada dolaska novog posla.

Implementira SimulatorEngine::Entities::SchedulerEntity.

override long Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::scheduleMakeDecisions () `[inline, protected, virtual]`

Implementira donošenje odluka o raspoređivanju poslova.

Povratne vrijednosti:

uvijek 0 (modelira trenutno donošenje odluka)

Implementira SimulatorEngine::Entities::SchedulerEntity.

override string Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler::ToString () `[inline]`

Daje opis temeljnih značajki raspoređivača poslova.

Opis razreda SimulatorEngine::Entities::JobEntity

Implementira osnovnu funkcionalnost posla na grozdu računala, koji čini dio diskretne simulacije.
Naslijeđuje od SimulatorEngine::Entities::SimulationEntity.

Naslijeđena u Simulator::EntityImplementations::JobEntities::SimpleJobEntity.

Strukture

- class JobHistory
- class ProgressReportB
- Handles progress report to the scheduler. class **QueueJobB**

Public članovi

- JobEntity (SimulationExecutive exc, SchedulerEntity schEnt)
- void queueUp (long time)
- void cancel (List< ResourceEntity > allocatedRes)
- virtual void start ()
- virtual void start (long SystemRuntimePrediction)
- String ToString_History ()
- long queueTime ()
- long startTime ()
- long endTime ()
- double slowdown ()
- double boundedSlowdown (double bound)
- long runTime ()
- double waitTime ()

Protected članovi

- abstract void nextProgressPoint (out long time, out double progress)

Protected atributi

- int resCount

Svojstva (property)

- int **ResCount** `[get]`
- double currentProgress `[get]`
- jobState jState `[get]`

Detaljno objašnjenje

Implementira osnovnu funkcionalnost posla na grozdu računala, koji čini dio diskretne simulacije.

Dokumentacija konstruktora i destruktora

SimulatorEngine::Entities::JobEntity::JobEntity (SimulationExecutive exc*, SchedulerEntity* schEnt*)* `[inline]`

Inicijalizira posao u simulaciji.

Parametri:

exc Jezgra simulacije nadležna za simuliranje posla
schEnt Raspoređivač poslova odgovoran za raspoređivanje posla u sustavu

Dokumentacija funkcija

double SimulatorEngine::Entities::JobEntity::boundedSlowdown (double bound*)* `[inline]`

Ograničeno usporenje posla.

Parametri:

bound Vrijednost koja se uzima kao minimalno trajanje posla ukoliko je posao kraći od te vrijednosti

void SimulatorEngine::Entities::JobEntity::cancel (List< ResourceEntity > allocatedRes*)* `[inline]`

Otkazuje posao. Raspoređivač poslova je obaviješten da taj posao nije potrebno izvršavati. Posao koji se otkazuje mora biti u redu čekanja raspoređivača poslova ili se treba izvoditi u trenutku otkazivanja.

long SimulatorEngine::Entities::JobEntity::endTime () `[inline]`

Vrijeme završetka posla.

abstract void SimulatorEngine::Entities::JobEntity::nextProgressPoint (out long time*, out double* progress*)* `[protected, pure virtual]`

Definira slijedeću točku prijave stupnja dovršenosti posla.

Parametri:

time Vrijeme u kojem treba obaviti slijedeću prijavu stupnja dovršenosti posla.
progress Stupanj dovršenosti posla kojeg je potrebno prijaviti.

Implementirano u Simulator::EntityImplementations::JobEntities::SimpleJobEntity.

long SimulatorEngine::Entities::JobEntity::queueTime () `[inline]`

Vrijeme u kojem je posao stavljen u red čekanja raspoređivača poslova.

void SimulatorEngine::Entities::JobEntity::queueUp (long time*)* `[inline]`

Stavlja posao u red čekanja raspoređivača poslova definiranog u konstruktoru objekta. Definicija raspoređivača poslova u konstruktoru ne dodaje automatski posao u red čekanja.

Parametri:

time Trenutak u kojem je potrebno dodati posao u red čekanja raspoređivača poslova

long SimulatorEngine::Entities::JobEntity::runTime () `[inline]`

Vrijeme izvođenja posla.

double SimulatorEngine::Entities::JobEntity::slowdown () `[inline]`

Usporenje posla.

virtual void SimulatorEngine::Entities::JobEntity::start (long SystemRuntimePrediction*)* `[inline, virtual]`

Pokreće posao.

Parametri:

SystemRuntimePrediction Sistemsko predviđanje trajanja posla

Reimplementacija u Simulator::EntityImplementations::JobEntities::BackfillingJobEntity.

virtual void SimulatorEngine::Entities::JobEntity::start () `[inline, virtual]`

Pokreće posao. Tipično raspoređivač poslova zove ovu funkciju. Automatski se pokreće aktivnost za periodičku dojavu stupnja dovršenosti posla.

Reimplementacija u Simulator::EntityImplementations::JobEntities::BackfillingJobEntity.

long SimulatorEngine::Entities::JobEntity::startTime () `[inline]`

Vrijeme pokretanja posla.

String SimulatorEngine::Entities::JobEntity::ToString_History () `[inline]`

Povijest promjena stanja posla.

Povratne vrijednosti:

Tekstualni opis svih promjena stanja posla sa pripadajućim vremenskim trenucima.

double SimulatorEngine::Entities::JobEntity::waitTime () `[inline]`

Vrijeme koje je posao čekao na izvođenje.

Dokumentacija svojstava

double SimulatorEngine::Entities::JobEntity::currentProgress `[get, protected]`

jobState SimulatorEngine::Entities::JobEntity::jState `[get]`

Trenutno stanje posla.

Opis razreda SimulatorEngine::Entities::JobGeneratorEntity

Ostvaruje generator poslova kao entitet trostupanjske diskretne simulacije.
Naslijeđuje od SimulatorEngine::Entities::SimulationEntity.
Naslijeđena u Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator, Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGeneratorSimulator::EntityImplementations::JobGeneratorEntities::TopologyJobGenerator.

Strukture

- class QueueNextJobB

Public članovi

- JobGeneratorEntity (SimulationExecutive exc, SchedulerEntity schEnt)
- void start ()
- abstract Boolean GetNextJob (out long time, out JobEntity job)

Svojstva (property)

- SchedulerEntity **schedulerEntity** `[get]`

Detaljno objašnjenje

Ostvaruje generator poslova kao entitet trostupanjske diskretne simulacije.

Dokumentacija konstruktora i destruktora

SimulatorEngine::Entities::JobGeneratorEntity::JobGeneratorEntity (SimulationExecutive exc, *SchedulerEntity* schEnt*)* `[inline]`

Inicijalizira generator poslova.

Parametri:

exc Jezgra simulacije nadležna za simuliranje generatora poslova
schEnt Raspoređivač poslova odgovoran za raspoređivanje posla u sustavu

Dokumentacija funkcija

abstract Boolean SimulatorEngine::Entities::JobGeneratorEntity::GetNextJob (out long time, *out JobEntity* job*)* `[pure virtual]`

Vraća vrijeme u kojem je potrebno generirati novi posao, te posao koji je potrebno generirati.

Parametri:

time Vremenski trenutak u kojem se generira posao
job Posao koji je potrebno generirati

Povratne vrijednosti:

Implementirano u Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator, Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGeneratorSimulator::EntityImplementations::JobGeneratorEntities::TopologyJobGenerator.

void SimulatorEngine::Entities::JobGeneratorEntity::start () `[inline]`

Pokreće generator poslova.

Opis razreda SimulatorEngine::LazyCollection-g< T >

Ostvaruje kolekciju objekata koji se mogu izuzimati iz kolekcije dok se po njoj vrši iteracija. Iterator će preskočiti objekte koji su obrisani tijekom iteracije.

Strukture

- class LazyEnumerator-g

Public članovi

- LazyCollection ()
- void Refresh ()
- void AddItem (T item)
- void RemoveItem (T item)
- Boolean isKilled (T item)

Svojstva (property)

- int Count `[get]`

Detaljno objašnjenje

template<T> class SimulatorEngine::LazyCollection-g< T >

Ostvaruje kolekciju objekata koji se mogu izuzimati iz kolekcije dok se po njoj vrši iteracija. Iterator će preskočiti objekte koji su obrisani tijekom iteracije.

Template Parameters:

T Tip objekta koji će se nalaziti u kolekciji

Dokumentacija funkcija

template<T > void SimulatorEngine::LazyCollection-g< T >::AddItem (T item*)* `[inline]`

Dodaje objekt u kolekciju. Dodani objekt se ne pojavljuje za vrijeme iteracije kroz kolekciju, osim u slučaju da je funkcija Refresh pozvana prije početka iteracije kolekcijom.

Parametri:

item Objekt koji se dodaje u kolekciju

template<T > Boolean SimulatorEngine::LazyCollection-g< T >::isKilled (T item*)* `[inline]`

Provjera da li je neki objekt izbrisan iz kolekcije.

Parametri:

item Objekt za kojeg se vrši provjera

Povratne vrijednosti:

true, ako je objekt izbrisan iz kolekcije

template<T > SimulatorEngine::LazyCollection-g< T >::LazyCollection () `[inline]`

Inicijalizira potrebne strukture podataka.

template<T > void SimulatorEngine::LazyCollection-g< T >::Refresh () `[inline]`

Briše obrisane objekta iz liste aktivnih objekata i dodaje nove objekte u listu dodanih objekata.

template<T > void SimulatorEngine::LazyCollection-g< T >::RemoveItem (T item*)* `[inline]`

Briše objekt iz kolekcije. Objekt se još uvijek nalazi u listi aktivnih objekata, ali će biti preskočen za vrijeme iteracije nad kolekcijom. Objekt je trajno uklonjen iz kolekcije pozivom funkcije Refresh.

Parametri:

item Objekt koji se briše iz kolekcije

Dokumentacija svojstava

template<T > int SimulatorEngine::LazyCollection-g< T >::Count `[get]`

Broj objekata u kolekciji.

Opis razreda Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler

Ostvaruje raspoređivač poslova koji resurse dijeli u više skupova, pri čemu su poslovi prema količini resursa koja traže dodijeljeni na raspoređivanje na jednom od skupova resursa. Unutar jednog skupa resursa koristi se postupak raspoređivanja poslova unazadnim popunjavanjem praznina.
Naslijeđuje od SimulatorEngine::Entities::SchedulerEntity.

Public članovi

- MultiProfileScheduler (SimulationExecutive exc, double allowedPercentage)
- override void queueJobHandling (JobEntity job)
- void StartJob (SchedulerProfile profile, BackfillingJobEntity job, List< ResourceEntity > resources)

Protected članovi

- override long scheduleMakeDecisions ()
- override void handleJobEstimatedTimeRanOut (JobEntity jobE)
- override void handleActivatedResurce (ResourceEntity resE)
- override void handleJobProgressReport (JobEntity jobE, double progress)
- override void handleJobSystemPredictedRuntimeRanOut (JobEntity jobE)

Detaljno objašnjenje

Ostvaruje raspoređivač poslova koji resurse dijeli u više skupova, pri čemu su poslovi prema količini resursa koja traže dodijeljeni na raspoređivanje na jednom od skupova resursa. Unutar jednog skupa resursa koristi se postupak raspoređivanja poslova unazadnim popunjavanjem praznina.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::MultiProfileScheduler (SimulationExecutive exc*, double* allowedPercentage*)* `[inline]`

Inicijalizira raspoređivač poslova.

Parametri:

exc jezgra simulatora odgovorna za izvođenje raspoređivača poslova.
allowedPercentage Postotak resursa koji se smije odvojiti u manje skupove za raspoređivanje poslova

Dokumentacija funkcija

override void Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::handleActivatedResurce (ResourceEntity resE*)* `[inline, protected, virtual]`

Obrađuje aktivaciju novog resursa.

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::handleJobEstimatedTimeRanOut (JobEntity jobE*) [inline, protected, virtual]*

Nije implementirano (vraća iznimku).

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::handleJobProgressReport (JobEntity jobE, *double* progress*) [inline, protected, virtual]*

Obrađuje prijavu stupnja dovršenosti posla (podržana samo 100tna dovršenost).

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::handleJobSystemPredictedRuntimeRanOut (JobEntity jobE*) [inline, protected, virtual]*

Nije implementirano (vraća iznimku).

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::queueJobHandling (JobEntity job*) [inline, virtual]*

Obrada dolaska novog posla.

Implementira SimulatorEngine::Entities::SchedulerEntity.

override long Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::scheduleMakeDecisions () [inline, protected, virtual]

Implementira donošenje odluka o raspoređivanju poslova.

Povratne vrijednosti:

uvijek 0 (modelira trenutno donošenje odluka)

Implementira SimulatorEngine::Entities::SchedulerEntity.

void Simulator::EntityImplementations::SchedulerEntities::MultiProfileScheduler::StartJob (SchedulerProfile profile, *BackfillingJobEntity* job, *List< ResourceEntity >* resources*) [inline]*

Pokreće posao na grupi resursa.

Parametri:

profile Profil resursa koji je zadužen za praćenje skupova resursa.
job Posao koji se pokreće.
resources Lista resursa na kojima se pokreće posao.

Opis razreda Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator

Generator poslova koji generira poslove slučajnog trajanja sa slučajnim vremenima između dolazaka poslova, koji zauzimaju slučajnu količinu resursa pri čemu se za sve slučajne vrijednosti uzimaju podaci iz uniformne distribucije ograničene zadanim intervalima.
Naslijeđuje od SimulatorEngine::Entities::JobGeneratorEntity.

Public članovi

- RandomJobGenerator (SimulationExecutive exc, SchedulerEntity schE, long jobCount, int nofResources, long maxJobRuntime, double percentageDeviationNegative, double percentageDeviationPositive)
- override bool GetNextJob (out long time, out JobEntity job)
- override string ToString ()

Detaljno objašnjenje

Generator poslova koji generira poslove slučajnog trajanja sa slučajnim vremenima između dolazaka poslova, koji zauzimaju slučajnu količinu resursa pri čemu se za sve slučajne vrijednosti uzimaju podaci iz uniformne distribucije ograničene zadanim intervalima.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator::RandomJobGenerator (SimulationExecutive exc, *SchedulerEntity* schE, *long* jobCount, *int* nofResources, *long* maxJobRuntime, *double* percentageDeviationNegative, *double* percentageDeviationPositive*) [inline]*

Parametri:

exc Jezgra simulatora kojoj pripada generator poslova.
schE Raspoređivač poslova zadužen za prihvat generiranih poslova.

jobCount Broj poslova koje je potrebno generirati.
nofResources Maksimalna količina resursa koji poslovi smiju zauzeti.
maxJobRuntime Najdulje dozvoljeno trajanje poslova.
percentageDeviationNegative Maksimalna negativna devijacija korisničke procjene trajanja posla od njegovog stvarnog trajanja.
percentageDeviationPositive Maksimalna pozitivna devijacija korisničke procjene trajanja posla od njegovog stvarnog trajanja.

Dokumentacija funkcija

override bool Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator::GetNextJob (out long time*, out* JobEntity job*)* `[inline, virtual]`

Vraća vremenski trenutak u kojem je potrebno generirati novi posao, te posao koji je potrebno generirati.

Povratne vrijednosti:

true ako ima još poslova za generirati, false inače

Implementira SimulatorEngine::Entities::JobGeneratorEntity.

override string Simulator::EntityImplementations::JobGeneratorEntities::RandomJobGenerator::ToString () `[inline]`

Generira tekstualni opis svojstava slučajnog generatora poslova.

Opis razreda SimulatorEngine::Entities::ResourceEntity

Ostvaruje resurs kao entitet trostupanjske diskretne simulacije.
Naslijeđuje od SimulatorEngine::Entities::SimulationEntity.
Naslijeđena u Simulator::EntityImplementations::ResourceEntities::SimpleResourceEntity.

Strukture

- class ActivateResourceB
- class BackInServiceB
- class OutOfServiceB
- class outOfServiceInterval

Public članovi

- ResourceEntity (SimulationExecutive exc, SchedulerEntity schEnt)
- void activate (long time)
- void setStateToBusy ()
- void setStateToIdle ()
- void setOutOfService (long beginTime, long endTime)

Svojstva (property)

- long TimeInService `[get]`
- long TimeUtilized `[get]`
- ResourceState resState `[get]`

Detaljno objašnjenje

Ostvaruje resurs kao entitet trostupanjske diskretne simulacije.

Dokumentacija konstruktora i destruktora

*SimulatorEngine::Entities::ResourceEntity::ResourceEntity (*SimulationExecutive exc*,* SchedulerEntity schEnt*)* `[inline]`

Inicijalizira resurs.

Parametri:

exc Jezgra simulacije nadležna za simuliranje resursa
schEnt Raspoređivač poslova odgovoran za raspoređivanje posla u sustavu

Dokumentacija funkcija

void SimulatorEngine::Entities::ResourceEntity::activate (long time*)* `[inline]`

Aktivira resurs. Obavještava raspoređivač poslova da može koristiti resurs.

Parametri:

time Vrijeme u koje će resurs postati aktivan

void SimulatorEngine::Entities::ResourceEntity::setOutOfService (long beginTime*, long* endTime*)* `[inline]`

Određuje vremenski raspon u kojem će resurs biti nedostupan.

Parametri:

beginTime Trenutak u kojem će resurs postati nedostupan
endTime Vrijeme u kojem će resurs opet postati dostupan

void SimulatorEngine::Entities::ResourceEntity::setStateToBusy () `[inline]`

Označava da resurs postaje zauzet. Raspoređivač poslova obično vrši ovo postavljanje.

void SimulatorEngine::Entities::ResourceEntity::setStateToIdle () `[inline]`

Označava da resurs postaje slobodan. Raspoređivač poslova obično vrši ovo postavljanje.

Dokumentacija svojstava

ResourceState SimulatorEngine::Entities::ResourceEntity::resState `[get, protected]`

Interno stanje resursa.

long SimulatorEngine::Entities::ResourceEntity::TimeInService `[get]`

Vrijeme aktivnosti resursa (dostupan raspoređivaču poslova).

long SimulatorEngine::Entities::ResourceEntity::TimeUtilized `[get]`

Vrijeme koje je resurs bio zauzet (izvodio poslove).

Opis razreda Simulator::EntityImplementations::ResourceEntities::ResourceGenerator

Ostvaruje generator resursa.
Naslijeđena u Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator.

Public članovi

- ResourceGenerator (SimulationExecutive exc, SchedulerEntity schE)
- abstract ResourceEntity[] GetResources ()

Detaljno objašnjenje

Ostvaruje generator resursa.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::ResourceEntities::ResourceGenerator::ResourceGenerator (SimulationExecutive exc, SchedulerEntity schE) `[inline]`

Inicijalizacija generatora resursa.

Parametri:

exc Jezgra simulacije kojoj će pripadati generirani resursi
schE Raspoređivač poslova koji će koristiti generirane resurse.

Dokumentacija funkcija

abstract ResourceEntity [] Simulator::EntityImplementations::ResourceEntities::ResourceGenerator::GetResources () `[pure virtual]`

Dohvaća popis resursa za koje je zadužen generator resursa.

Implementirano u Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator.

Opis razreda SimulatorEngine::Entities::SchedulerEntity

Implementira osnovnu funkcionalnost raspoređivača poslova na grozdu računala, koji čini dio diskretne simulacije.
Naslijeđuje od SimulatorEngine::Entities::SimulationEntity.
Naslijeđena u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

Strukture

- class RescheduleC
- Implements conditional action that fires when system state changes and calls the decision method of the. class **ThinkingB**

Public članovi

- delegate void **ResourceInServiceDelegate** (ResourceEntity resource)
- delegate void **ScheduledDelegate** (JobEntity jobE, List< ResourceEntity > resList)

- delegate void **JobTerminatedDelegate** (JobEntity jobE)
- SchedulerEntity (SimulationExecutive exc)
- abstract void queueJobHandling (JobEntity job)
- void cancelJob (JobEntity job, List< ResourceEntity > allocatedResoures)
- SchedulerMetrics GetSchedulerMetrics ()
- void queueJob (JobEntity job)
- void handleJobProgressReportInternal (JobEntity jobE, double progress)
- void handleJobEstimatedTimeRanOutInternal (JobEntity jobE)
- void handleJobSystemPredictedRuntimeRanOutInternal (JobEntity jobE)
- String GetStatisticsByJobSize ()
- void handleActivatedResurceInternal (ResourceEntity resE)
- void handleOutOfServiceResurceInternal (ResourceEntity resE)
- void handleBackInServiceResurceInternal (ResourceEntity resE)

Protected članovi

- abstract long scheduleMakeDecisions ()
- virtual void scheduleEnforceDecisions ()
- virtual void handleJobProgressReport (JobEntity jobE, double progress)
- void StartJob (JobEntity jobE, List< ResourceEntity > allocatedResources)
- void StartJob (JobEntity jobE, List< ResourceEntity > allocatedResources, long predictedRuntime)
- abstract void handleJobEstimatedTimeRanOut (JobEntity jobE)
- abstract void handleJobSystemPredictedRuntimeRanOut (JobEntity jobE)
- void calculateUtilization (out double utilization, out double utilizationWithReduction)
- virtual void handleActivatedResurce (ResourceEntity resE)
- virtual void handleOutOfServiceResurce (ResourceEntity resE)
- virtual void handleBackInServiceResurce (ResourceEntity resE)

Protected atributi

- Boolean **jobStateChanged** = false
- Boolean resourceStateChanged = false
- SchedulerMetrics **schedulerMetrics** = new SchedulerMetrics()

Događaji

- ResourceInServiceDelegate **ResourceInServiceEvent**
- ScheduledDelegate **scheduledEvent**
- JobTerminatedDelegate **JobFinishedEvent**
- JobTerminatedDelegate **JobCanceledEvent**

Detaljno objašnjenje

Implementira osnovnu funkcionalnost raspoređivača poslova na grozdu računala, koji čini dio diskretne simulacije.

Dokumentacija konstruktora i destruktora

SimulatorEngine::Entities::SchedulerEntity::SchedulerEntity (SimulationExecutive exc*)* `[inline]`

Inicijalizira raspoređivač poslova.

Parametri:

exc Jezgra simulacije nadležna za simuliranje raspoređivača poslova.

Dokumentacija funkcija

void SimulatorEngine::Entities::SchedulerEntity::calculateUtilization (out double utilization*, out double* utilizationWithReduction*)* `[inline, protected]`

Parametri:

utilization iskorištenost sustava
utilizationWithReduction iskorištenost sustava umanjena za poslove koji su prekinuti od strane sustava

void SimulatorEngine::Entities::SchedulerEntity::cancelJob (JobEntity job*, List< ResourceEntity >* allocatedResoures*)* `[inline]`

Otkazuje posao koji je u sustavu.

Parametri:

job Posao koji se otkazuje
allocatedResoures Resursi koje je posao zauzimao

SchedulerMetrics SimulatorEngine::Entities::SchedulerEntity::GetSchedulerMetrics () `[inline]`

Dohvaća metrike (mjere učinkovitosti) raspoređivača poslova.

String SimulatorEngine::Entities::SchedulerEntity::GetStatisticsByJobSize () `[inline]`

Računa statistiku učinkovitosti raspoređivača poslova za različita trajanja poslova.

Povratne vrijednosti:

Tekstualni prikaz statistike

*virtual void SimulatorEngine::Entities::SchedulerEntity::handleActivatedResurce (*ResourceEntity resE*)* `[inline, protected, virtual]`

Obrađuje aktivaciju resursa u sustavu.

Parametri:

resE Resurs koji se aktivira

Reimplementacija u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

*void SimulatorEngine::Entities::SchedulerEntity::handleActivatedResurceInternal (*ResourceEntity resE*)* `[inline]`

Obrađuje aktivaciju resursa u sustavu. Poziva se automatski iz odgovarajućeg resursa. Nije je potrebno koristiti, već samo implementirati funkciju handleActivatedResurce.

Parametri:

resE Resurs koji se aktivira

*virtual void SimulatorEngine::Entities::SchedulerEntity::handleBackInServiceResurce (*ResourceEntity resE*)* `[inline, protected, virtual]`

Obrađuje ponovnu aktivaciju resursa u sustavu.

Parametri:

resE Resurs koji se aktivira

*void SimulatorEngine::Entities::SchedulerEntity::handleBackInServiceResurceInternal (*ResourceEntity resE*)* `[inline]`

Obrađuje ponovnu aktivaciju resursa u sustavu. Poziva se automatski iz odgovarajućeg resursa. Nije je potrebno koristiti, već samo implementirati funkciju handleBackInServiceResurce.

Parametri:

resE Resurs koji se aktivira

*abstract void SimulatorEngine::Entities::SchedulerEntity::handleJobEstimatedTimeRanOut (*JobEntity jobE*)* `[protected, pure virtual]`

Ostvaruje obradu trenutka u kojem istekne korisnički predviđeno trajanje posla.

Parametri:

jobE Posao za koji je isteklo korisnički predviđeno trajanje posla

Implementirano u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

*void SimulatorEngine::Entities::SchedulerEntity::handleJobEstimatedTimeRanOutInternal (*JobEntity jobE*)* `[inline]`

Interna funkcija za obradu trenutka u kojem istekne korisnički predviđeno trajanje posla. Nju nije potrebno pozivati, već samo implementirati vlastitu funkciju obrade tog događa sa nazivom handleJobEstimatedTimeRanOut.

Parametri:

jobE Posao za koji je isteklo korisnički predviđeno trajanje posla

*virtual void SimulatorEngine::Entities::SchedulerEntity::handleJobProgressReport (*JobEntity jobE, *double* progress*)* `[inline, protected, virtual]`

Obrađuje prijavu stupnja dovršenosti posla.

Parametri:

jobE Posao za koji je izvršena prijava dovršenosti
progress Stupanj dovršenosti posla (izražen u postocima)

Reimplementacija u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

void SimulatorEngine::Entities::SchedulerEntity::handleJobProgressReportInternal (JobEntity jobE*, double* progress*) [inline]*

U slučajevima kad je posao potpuno završen mijenja se vrijednost mjera učinkovitosti postupka raspoređivanja poslova. Poziva se automatski od strane posla, te je nije potrebno koristiti.

Parametri:

jobE Posao koji završava
progress Stupanj dovršenosti posla

abstract void SimulatorEngine::Entities::SchedulerEntity::handleJobSystemPredictedRuntimeRanOut (JobEntity jobE*) [protected, pure virtual]*

Ostvaruje obradu trenutka u kojem istekne sistemski (od strane raspoređivača poslova) predviđeno trajanje posla.

Parametri:

jobE Posao za koji je isteklo sistemski predviđeno trajanje posla

Implementirano u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

void SimulatorEngine::Entities::SchedulerEntity::handleJobSystemPredictedRuntimeRanOutInternal (JobEntity jobE*) [inline]*

Interna funkcija za obradu trenutka u kojem istekne sistemski predviđeno trajanje posla. Nju nije potrebno pozivati, već samo implementirati vlastitu funkciju obrade tog događa sa nazivom handleJobSystemPredictedRuntimeRanOut.

Parametri:

jobE Posao za koji je isteklo sistemski predviđeno trajanje posla

virtual void SimulatorEngine::Entities::SchedulerEntity::handleOutOfServiceResurce (ResourceEntity resE*) [inline, protected, virtual]*

Obrađuje deaktivaciju resursa u sustavu.

Parametri:

resE Resurs koji se deaktivira

void SimulatorEngine::Entities::SchedulerEntity::handleOutOfServiceResurceInternal (ResourceEntity resE*) [inline]*

Obrađuje deaktivaciju resursa u sustavu. Poziva se automatski iz odgovarajućeg resursa. Nije je potrebno koristiti, već samo implementirati funkciju handleOutOfServiceResurce.

Parametri:

resE Resurs koji se deaktivira

void SimulatorEngine::Entities::SchedulerEntity::queueJob (JobEntity job*) [inline]*

Određuje da se promijenilo stanje poslova u sustavu, te zove funkciju queueJobHandling za obradu novopridošlog posla.

Parametri:

job Posao koji je došao u sustav.

abstract void SimulatorEngine::Entities::SchedulerEntity::queueJobHandling (JobEntity job*) [pure virtual]*

Ostvaruje obradu dolaska novog posla u sustav.

Parametri:

job Posao koji dolazi u sustav

Implementirano u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

virtual void SimulatorEngine::Entities::SchedulerEntity::scheduleEnforceDecisions () `[inline, protected, virtual]`

Provodi odluke o raspoređivanju poslova donesene u funkciji. scheduleMakeDecisions čime se modelira trajanje procesa donošenja odluka. Prekrivanje ove funkcije je dozvoljeno uz obavezno pozivanje funkcije iz baznog razreda.

abstract long SimulatorEngine::Entities::SchedulerEntity::scheduleMakeDecisions () `[protected, pure virtual]`

Donosi odluke o raspoređivanju poslova.

Povratne vrijednosti:

Vrijeme nakon kojeg je potrebno provesti donesene odluke

Implementirano u Simulator::EntityImplementations::SchedulerEntities::ConservativeBackfilling, Simulator::EntityImplementations::SchedulerEntities::FCFS_Scheduler, Simulator::EntityImplementations::SchedulerEntities::MultiProfileSchedulerSimulator::EntityImplementations::SchedulerEntities::SmartBackfilling.

void SimulatorEngine::Entities::SchedulerEntity::StartJob (JobEntity jobE*, List<* ResourceEntity *>* allocatedResources*)* `[inline, protected]`

Pokreće posao na skupu resursa.

Parametri:

jobE Posao koji se pokreće.
allocatedResources Skup resursa koji su dodijeljeni poslu.

void SimulatorEngine::Entities::SchedulerEntity::StartJob (JobEntity jobE*, List<* ResourceEntity *>* allocatedResources*, long* predictedRuntime*)* `[inline, protected]`

Pokreće posao na skupu resursa.

Parametri:

jobE Posao koji se pokreće.
allocatedResources Skup resursa koji su dodijeljeni poslu.
predictedRuntime Predviđeno trajanje posla.

Opis razreda Simulator::EntityImplementations::JobEntities::SimpleJobEntity

Ostvaruje jednostavni entitet koji predstavlja posao u simulaciji. Od funkcionalnosti je dostupna jedino prijava potpune dovršenosti posla.
Naslijeđuje od SimulatorEngine::Entities::JobEntity.
Naslijeđena u Simulator::EntityImplementations::JobEntities::BackfillingJobEntity.

Public članovi

- SimpleJobEntity (SimulationExecutive exc, SchedulerEntity schE, long duration, int resCount)

Protected članovi

- override void nextProgressPoint (out long time, out double progress)

Detaljno objašnjenje

Ostvaruje jednostavni entitet koji predstavlja posao u simulaciji. Od funkcionalnosti je dostupna jedino prijava potpune dovršenosti posla.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::JobEntities::SimpleJobEntity::SimpleJobEntity (SimulationExecutive exc*, SchedulerEntity* schE*, long* duration*, int* resCount*)* `[inline]`

Inicijalizira entitet koji predstavlja posao.

Parametri:

exc Jezgra simulatora kojoj posao pripada
schE Raspoređivač poslova zadužen za posao
duration Trajanje posla
resCount Količina resursa potrebna za izvršavanje posla

Dokumentacija funkcija

override void Simulator::EntityImplementations::JobEntities::SimpleJobEntity::nextProgressPoint (out long time*, out double* progress*)* `[inline, protected, virtual]`

Definira postupak prijave stupnja dovršenosti posla, pri čemu se dojavljuje samo trenutak potpunog završetka posla.

Parametri:

time Vrijeme potpune dovršenosti posla
progress Stupanje (uvijek 100%) dovršenosti posla

Implementira SimulatorEngine::Entities::JobEntity.

Opis razreda Simulator::EntityImplementations::ResourceEntities::SimpleResourceEntity

Ostvaruje jednostavan resurs koji se trenutno aktivira i uvijek ostaje aktivan.
Naslijeđuje od SimulatorEngine::Entities::ResourceEntity.

Public članovi

- SimpleResourceEntity (SimulationExecutive exc, SchedulerEntity schE)

Detaljno objašnjenje

Ostvaruje jednostavan resurs koji se trenutno aktivira i uvijek ostaje aktivan.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::ResourceEntities::SimpleResourceEntity::SimpleResourceEntity (SimulationExecutive exc, SchedulerEntity schE) `[inline]`

Inicijalizira i aktivira resurs.

Parametri:

exc Jezgra simulatora unutar koje se resurs simulira
schE Raspoređivač poslova zadužen za upravljanje resursom.

Opis razreda Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator

Generator resursa koji se aktiviraju odmah po inicijalizaciji.
Naslijeđuje od Simulator::EntityImplementations::ResourceEntities::ResourceGenerator.

Public članovi

- SimpleResourceGenerator (SimulationExecutive exc, SchedulerEntity schE, int numberOfSimpleResources)
- override SimulatorEngine.Entities.ResourceEntity[] GetResources ()
- override string ToString ()

Detaljno objašnjenje

Generator resursa koji se aktiviraju odmah po inicijalizaciji.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator::SimpleResourceGenerator (SimulationExecutive exc, SchedulerEntity schE, int numberOfSimpleResources) `[inline]`

Inicijalizira generator resursa.

Parametri:

exc Jezgra simulatora unutar koje se resursi simuliraju.
schE Raspoređivač poslova zadužen za korištenje resursa.
numberOfSimpleResources Broj resursa koje je potrebno generirati

Dokumentacija funkcija

override SimulatorEngine.Entities.ResourceEntity [] Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator::GetResources () `[inline, virtual]`

Generira popis aktiviranih resursa.

Implementira Simulator::EntityImplementations::ResourceEntities::ResourceGenerator.

override string Simulator::EntityImplementations::ResourceEntities::SimpleResourceGenerator::ToString () `[inline]`

Generira opis svojstava generatora resursa.

Opis razreda SimulatorEngine::Entities::SimulationEntity

Implementira osnovnu funkcionalnost simulacijskog entiteta u trostupanjskoj diskretnoj simulaciji. Simulacijski entitet sadrži fiksirane i uvjetne aktivnosti koje su semantički vezane uz njegovu internu strukturu.
Naslijeđuje od SimulatorEngine::ActivityControl.
Naslijeđena u SimulatorEngine::Entities::JobEntity, SimulatorEngine::Entities::JobGeneratorEntity,

SimulatorEngine::Entities::ResourceEntitySimulatorEngine::Entities::SchedulerEntity.

Public članovi

- SimulationEntity (SimulationExecutive exc)
- void Remove ()
- int findAndExecuteConditionalActivities ()
- void deleteActivity (Activity activity)
- void addActivity (Activity activity)
- long getClock ()
- bool simulationStarted ()

Svojstva (property)

- SimulationExecutive **simulationExecutive** `[get]`
- Boolean cActivityMngAllowed `[get]`
- virtual String Name `[get]`

Detaljno objašnjenje

Implementira osnovnu funkcionalnost simulacijskog entiteta u trostupanjskoj diskretnoj simulaciji. Simulacijski entitet sadrži fiksirane i uvjetne aktivnosti koje su semantički vezane uz njegovu internu strukturu.

Dokumentacija konstruktora i destruktora

SimulatorEngine::Entities::SimulationEntity::SimulationEntity (SimulationExecutive exc) `[inline]`

Inicijalizira simulacijski entitet.

Parametri:

exc Jezgra simulacije nadležna za upravljanje simulacijskim entitetom

Dokumentacija funkcija

void SimulatorEngine::Entities::SimulationEntity::addActivity (Activity activity) `[inline]`

Dodaje aktivnost simulacijskom entitetu.

Implementira SimulatorEngine::ActivityControl.

void SimulatorEngine::Entities::SimulationEntity::deleteActivity (Activity activity) `[inline]`

Briše aktivnost koja pripada simulacijskom entitetu.

Implementira SimulatorEngine::ActivityControl.

int SimulatorEngine::Entities::SimulationEntity::findAndExecuteConditionalActivities () `[inline]`

Izvodi sve uvjetne aktivnosti čiji su uvjeti ispunjeni.

Povratne vrijednosti:

long SimulatorEngine::Entities::SimulationEntity::getClock () `[inline]`

Trenutno vrijeme u simulaciji.

Implementira SimulatorEngine::ActivityControl.

void SimulatorEngine::Entities::SimulationEntity::Remove () `[inline]`

Briše simulacijski entitet iz njemu pridružene jezgre simulacije.

bool SimulatorEngine::Entities::SimulationEntity::simulationStarted () `[inline]`

Provjerava da li je simulacija započela.

Implementira SimulatorEngine::ActivityControl.

Dokumentacija svojstava

Boolean SimulatorEngine::Entities::SimulationEntity::cActivityMngAllowed `[get]`

Provjerava da li entitet sadrži uvjetne aktivnosti.

virtual String SimulatorEngine::Entities::SimulationEntity::Name `[get]`

Određuje naziv simulacijskog entiteta, ukoliko se funkcija ne implementira u izvedenim razredima entiteti dobivaju standardizirana imena ovisno o vrsti izvedenog razreda i broju entiteta u tom razredu.

Reimplementacija u Simulator::EntityImplementations::JobEntities::BackfillingJobEntity.

Opis razreda SimulatorEngine::SimulationExecutive

SimulationExecutive ostvaruje temeljnu funkcionalnost trostupanjske jezgre simulatora temeljenog na diskretnim

događajima.
Nasljeđuje od SimulatorEngine::ActivityControl.

Public članovi

- SimulationExecutive ()
- void addEntity (SimulationEntity entity)
- void deleteEntity (SimulationEntity entity)
- int step ()
- void deleteActivity (Activity activity)
- void addActivity (Activity activity)
- long getClock ()
- Boolean simulationStarted ()

Svojstva (property)

- long nOfExecutedBActivities `[get]`
- long nOfExecutedCActivities `[get]`
- long nOfExecutedActivities `[get]`

Detaljno objašnjenje

SimulationExecutive ostvaruje temeljnu funkcionalnost trostupanjske jezgre simulatora temeljenog na diskretnim događajima.

Dokumentacija konstruktora i destruktora

SimulatorEngine::SimulationExecutive::SimulationExecutive () `[inline]`

Inicijalizira kolekcije entiteta i aktivnosti nužne za rad jezgre simulatora.

Dokumentacija funkcija

void SimulatorEngine::SimulationExecutive::addActivity (Activity activity) `[inline]`

Dodaje aktivnost u simulaciju. Složenost O(1).

Parametri:

activity Aktivnost koja se dodaje u simulaciju

Implementira SimulatorEngine::ActivityControl.

void SimulatorEngine::SimulationExecutive::addEntity (SimulationEntity entity) `[inline]`

Dodaje novi entitet u simulaciju.

Parametri:

entity Entitet koji se dodaje simulaciji

void SimulatorEngine::SimulationExecutive::deleteActivity (Activity activity) `[inline]`

Briše aktivnost iz simulacije. Složenost O(1).

Parametri:

activity Aktivnost koju je potrebno pobrisati.

Implementira SimulatorEngine::ActivityControl.

void SimulatorEngine::SimulationExecutive::deleteEntity (SimulationEntity entity) `[inline]`

Briše entitet iz simulacije.

Parametri:

entity Entitet koji je potrebno izbrisati

long SimulatorEngine::SimulationExecutive::getClock () `[inline]`

Vraća trenutno vrijeme simulacije.

Povratne vrijednosti:

Trenutno vrijeme simulacije

Implementira SimulatorEngine::ActivityControl.

Boolean SimulatorEngine::SimulationExecutive::simulationStarted () `[inline]`

Određuje da li je simulacija već započela.

Povratne vrijednosti:

True ako je simulacija već započela, false inače

Implementira SimulatorEngine::ActivityControl.

int SimulatorEngine::SimulationExecutive::step () `[inline]`

Izvršava jedan korak simulacije koji se sastoji od tri stupnja (faze) izvođenja.

Povratne vrijednosti:

Broj ukupno izvršenih aktivnosti (fiksiranih i uvjetnih). Vrijednost 0 označava da više nema odgovarajućih aktivnosti za izvršavanja i da je simulacija gotova.

Dokumentacija svojstava

long SimulatorEngine::SimulationExecutive::nOfExecutedActivities `[get]`

Broj ukupno izvršenih aktivnosti.

long SimulatorEngine::SimulationExecutive::nOfExecutedBActivities `[get]`

Broj ukupno izvršenih fiksiranih aktivnosti.

long SimulatorEngine::SimulationExecutive::nOfExecutedCActivities `[get]`

Broj ukupno izvršenih uvjetnih aktivnosti.

Opis razreda Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling

Implementira postupak unazadnog popunjavanja praznina uz optimizaciju dinamičkim programiranjem (UPPDP).
Nasljeđuje od SimulatorEngine::Entities::SchedulerEntity.

Public članovi

- **SmartBackfilling** (SimulationExecutive exc, long maxReservations)
- SmartBackfilling (SimulationExecutive exc, long maxReservations, int EligibleJobsOrderType)
- override void queueJobHandling (JobEntity job)
- override string **ToString** ()

Protected članovi

- override long scheduleMakeDecisions ()
- override void handleJobProgressReport (JobEntity jobE, double progress)
- override void handleActivatedResurce (ResourceEntity resE)
- override void handleJobEstimatedTimeRanOut (JobEntity jobE)
- override void handleJobSystemPredictedRuntimeRanOut (JobEntity jobE)

Detaljno objašnjenje

Implementira postupak unazadnog popunjavanja praznina uz optimizaciju dinamičkim programiranjem (UPPDP).

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::SmartBackfilling (SimulationExecutive exc, *long* maxReservations, *int* EligibleJobsOrderType*)* `[inline]`

Inicijalizira postupak raspoređivanja poslova.

Parametri:

exc Jezgra simulatora odgovorna za izvođenje raspoređivača poslova.
maxReservations Maksimalni broj dozvoljenih rezervacija
eligibleJobsOrderType Odabir strategije raspoređivanja UPPDP sa prioritiziranjem 0 - prema redoslijedu dolaska (UPPDP_PR) 1 - uskih poslova (UPPDP_UP) 2 - širokih poslova (UPPDP_ŠP)

Dokumentacija funkcija

override void Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::handleActivatedResurce (ResourceEntity resE*)* `[inline, protected, virtual]`

Obrađuje aktivaciju novog resursa.

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::handleJobEstimatedTimeRanOut (JobEntity jobE*)* `[inline, protected, virtual]`

Obrađuje istek korisničke procjene trajanja posla.

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::handleJobProgressReport (JobEntity jobE, *double* progress*)* `[inline, protected, virtual]`

Obrađuje prijavu stupnja dovršenosti posla (podržana samo 100tna dovršenost).

Reimplementirano od SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::handleJobSystemPredictedRuntimeRanOut (JobEntity jobE*)* `[inline, protected, virtual]`

Nije implementirano (vraća iznimku).

Implementira SimulatorEngine::Entities::SchedulerEntity.

override void Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::queueJobHandling (JobEntity job*)* `[inline, virtual]`

Obrada dolaska novog posla

Implementira SimulatorEngine::Entities::SchedulerEntity.

override long Simulator::EntityImplementations::SchedulerEntities::SmartBackfilling::scheduleMakeDecisions () `[inline, protected, virtual]`

Implementira donošenje odluka o raspoređivanju poslova.

Povratne vrijednosti:

uvijek 0 (modelira trenutno donošenje odluka)

Implementira SimulatorEngine::Entities::SchedulerEntity.

Opis razreda Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGenerator

Generira poslove na temelju zadane datoteke oblikovane u SWF formatu.
Nasljeđuje od SimulatorEngine::Entities::JobGeneratorEntity.

Public članovi

- **SWFJobGenerator** (SimulationExecutive exc, SchedulerEntity schE, String fileName, Boolean PerfectPrediction, int PartitionNumber, int PartitionMaxTime, Boolean loadDetailedStatusPrediction)
- **SWFJobGenerator** (SimulationExecutive exc, SchedulerEntity schE, String fileName, Boolean PerfectPrediction, double PredictionMultiplier, int PartitionNumber, int PartitionMaxTime, Boolean loadDetailedStatusPrediction)
- SWFJobGenerator (SimulationExecutive exc, SchedulerEntity schE, String fileName, int PartitionNumber, int PartitionMaxTime, Boolean loadDetailedStatusPrediction, Boolean loadRuntimePredictions)
- override bool GetNextJob (out long time, out JobEntity job)
- override string ToString ()

Detaljno objašnjenje

Generira poslove na temelju zadane datoteke oblikovane u SWF formatu.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGenerator::SWFJobGenerator (SimulationExecutive exc*, SchedulerEntity* schE*, String* fileName*, int* PartitionNumber*, int* PartitionMaxTime*, Boolean* loadDetailedStatusPrediction*, Boolean* loadRuntimePredictions*)* `[inline]`

SWF Job generator.

Parametri:

exc Jezgra simulatora koja izvodi generator poslova.
schE Raspoređivač poslova zadužen za raspoređivanje generiranih poslova.
fileName Naziv SWF datoteke sa opisom poslova.
perfectPrediction Ukoliko je true umjesto korisničkih predikcija trajanja poslova koriste se stvarna vremena trajanja poslova.
PartitionNumber Identifikator particije grozda računala koja se simulira. Ukoliko se stavi vrijednost -1 generiraju se poslovi svih particija.
PartitionMaxTime Standardizirano maksimalno vrijeme izvođenja posla na ciljanoj particiji koje se koristi umjesto korisničke procjene trajanja posla ukoliko ona nije zadana.
useDetailedStatusPrediction Ukoliko je true, koriste se sistemski generirana predviđanja statusa poslova.
useDetailedStatusPrediction Ukoliko je true, koriste se sistemski generirana predviđanja trajanja poslova.

Dokumentacija funkcija

override bool Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGenerator::GetNextJob (out long time*, out JobEntity* job*)* `[inline, virtual]`

Vraća vremenski trenutak u kojem je potrebno generirati novi posao, te posao koji je potrebno generirati.

Povratne vrijednosti:

true ako ima još poslova za generirati, false inače

Implementira SimulatorEngine::Entities::JobGeneratorEntity.

override string Simulator::EntityImplementations::JobGeneratorEntities::SWFJobGenerator::ToString () `[inline]`

Generira tekstualni opis svojstava generatora poslova.

Opis razreda Simulator::EntityImplementations::JobGeneratorEntities::TopologyJobGenerator

Generira poslove na temelju opisa opterećenja navedenog u XML datoteci.
Nasljeđuje od SimulatorEngine::Entities::JobGeneratorEntity.

Public članovi

- TopologyJobGenerator (SimulationExecutive exc, SchedulerEntity schE, String fileName)
- override bool GetNextJob (out long time, out JobEntity job)

Detaljno objašnjenje

Generira poslove na temelju opisa opterećenja navedenog u XML datoteci.

Dokumentacija konstruktora i destruktora

Simulator::EntityImplementations::JobGeneratorEntities::TopologyJobGenerator::TopologyJobGenerator (SimulationExecutive exc*, SchedulerEntity* schE*, String* fileName*)* `[inline]`

Inicijalizira generator poslova.

Parametri:

exc Jezgra simulatora koja izvodi generator poslova.
schE Raspoređivač poslova zadužen za raspoređivanje generiranih poslova.
fileName Naziv XML datoteke na temelju koje će se generirati poslovi.

Dokumentacija funkcija

override bool Simulator::EntityImplementations::JobGeneratorEntities::TopologyJobGenerator::GetNextJob (out long time*, out JobEntity* job*)* `[inline, virtual]`

Vraća vremenski trenutak u kojem je potrebno generirati novi posao, te posao koji je potrebno generirati.

Povratne vrijednosti:

true ako ima još poslova za generirati, false inače

Implementira SimulatorEngine::Entities::JobGeneratorEntity.

POPIS LITERATURE

[1] M. Flynn, *Some Computer Organizations and Their Effectiveness*, IEEE Transactions on Computers, Vol. C-21, Issue 9, pp. 948-960, 1972.

[2] I. Grudenić, N. Bogunović, *Modeling and Verification of MPI Based Distributed Software*, Lecture Notes in Computer Science, Vol. 4192, pp. 123-132, 2006.

[3] N. Bogunović, I. Grudenić, E. Pek, *A Synthesized Framework for Formal Verification of Computing Systems*, Journal on Systemics, Cybernetics and Informatics, Vol. 1, No. 6, pp. 6-11, 2003.

[4] *InfiniBand Architecture Specification*, Infiniband Trade Association, Vol.1, Rel. 1.2.1, 2007.

[5] I. Grudenić, I.Bakarčić, N.Bogunović, *Computer Cluster Workload Analysis*, In the Proceedings of the Joint Conferences Computers in Technical systems and Intelligent systems, pp. 43-46, 2010.

[6] D.G. Feitelson, L. Rudolph, *Toward Convergence in Job Schedulers for Parallel Supercomputers*, In Jobs Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Vol. 1911, pp. 1-17, 2000

[7] V.S. Sunderam, *PVM: A Framework for Parallel Distributed Computing*, Concurrency: Practice and Experience, Vol. 2, pp. 315-339, 1990.

[8] J.C. Sancho, F. Petrini, K.Davis, R.Gioiosa, S. Jiang, *Current Practice and a Direction Forward in Checkpoint/Restart Implementations for Fault Tolerance*, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Vol. 19, p. 300.2-, 2005.

[9] J.S. Plank, M. Beck, G. Kingsley, K. Li, *Libckpt: Transparent Checkpointing under Unix*, In Proceedings of the Usenix Winter 1995 Technical Conference, pp. 213-223, 1995.

[10] M. Litzkow, T. Tannenbaum, J. Basney, M. Livny, *Checkpoint and migration of UNIX processes in the Condor distributed processing system*, Technical Report CS-TR-199701346, University of Wisconsin, Madison, 1997.

[11] S. Sankaran, J.M. Squyres, B. Barrett, A.Lumsdaine, *The LAM/MPI checkpoint/restart framework: System-initiated checkpointing*, In Proceedings of the LACSI Symposium, pp. 479-493, 2003.

[12] A. Papoulis, S.Unnikrishna Pillai, *Probability, Random Variables and Stochastic Processes*, McGraw Hill Higher Education, 2002.

[13] D. G. Feitelson, *Packing Schemes for Gang Scheduling*, Lecture Notes in Computer Science, Vol. 1162, 1996.

[14] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, J. Riordan, *Modeling of workload in MPPs*, Lecture Notes in Computer Science, Vol. 1291, 1997.

[15] A.B. Downey, *A parallel workload model and its implications*, In the Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, 1997.

[16] U. Lublin, D.G. Feitelson, *The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs*, Journal of Parallel and Distributed Computing, Vol. 63, 2001.

[17] D. G. Feitelson, *Metrics for parallel job scheduling and their convergence*, Lecture Notes in Computer Science, 2001, Vol. 2221, pp. 188-205, 2001.

[18] I. Grudenić, S. Groš, N. Bogunović, *Load Balancing MPI Algorithm for High Throughput Applications*, In the Proceedings of the 30th International Conference on Information Technology Interfaces, pp. 839-843, 2008.

[19] I. Grudenić, N. Bogunović, *Symbolic Boolean Function Representation and Handling on Distributed Computing Systems*, In the Proceedings of the 13th International Conference on Information Systems Analysis and Synthesis, pp. 127-132, 2007.

[20] I. Grudenić, N. Bogunović, *Caching in Parallel BDD Package*, In the Proceedings of the 27th International Conference on Information Techology Interfaces, (Lužar-Stiffler, Vesna ; Hljuz Dobrić, Vesna ur.), pp. 631-635, 2005.

[21] E.G. Coffman Jr, M.R. Garez, D.S. Johnson, R.E. Tarjan, *Performance bounds for level-oriented two-dimensional packing algorithms*, SIAM Journal on Computing, Vol. 9, pp.808 826, 1980.

[22] I. Grudenić,N. Bogunović, *Analysis of Scheduling Algorithms for Computer Clusters*, In the Proceedings of the Joint Conferences Computers in Technical systems and Intelligent systems, pp. 59-64 2008.

[23] J.H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman and Hall, 2004.

[24] D. Zotkin, P.J. Keleher, *Job-length Estimation and Performance in Backfilling Schedulers*, In the Proceedings of the 8th International Symposium on High Performance Distributed Computing, 1999.

[25] C. B. Lee, A. E. Snavely, *Precise and realistic utility functions for user-centric performance analysis of schedulers*, In the Proceedings of the 16th international symposium on High performance distributed computing, pp. 107-116, 2007.

[26] J. Ousterhout, *Scheduling Techniques for Concurrent Systems*, In Proceedings of the 3rd International Conference on Distributed Computing Systems, pp. 22-30, 1982.

[27] B.B. Zhou, D. Walsh, R.P. Brent, *Resource Allocation Schemes for Gang Scheduling*, Lecture Notes In Computer Science, Vol. 1911, pp. 74-86, 2000.

[28] J. Corbalan, X. Martorell, J. Labarta, *Improving Gang Scheduling through Job Performance Analysis and Malleability*, In the Proceedings of the 15th International Conference on Supercomputing, pp. 301-311, 2001.

[29] D.G. Feitelson, L. Rudolph, *Evaluation of Design choices for gang Scheduling using Distributed Hierarchical Control*, Journal of Parallel and Distributed Computing, Vol. 35, pp. 18-34, 1996.

[30] A.Batat, D.G. Feitelson, *Gang scheduling with memory considerations*, In the Proceedings of the International Parallel and Distributed Processing Symposium, 2000, pp. 109-114.

[31] O. Arndt, B. Freisleben, T. Kielmann, F.Thilo, *A comparative study of online scheduling algorithms for networks of workstations*, Cluster Computing, Vol. 3, Issue 2, pp. 95-112, 2000.

[32] U. Schwiegelshohn, R. Yahyapour, *Analysis of first-come-first-serve parallel job scheduling*, In the Proceedings of the 9th annual ACM-SIAM symposium on Discrete algorithms, pp. 629-638, 1998.

[33] S. Eilon, I.G. Chowdhury, *Minimising Waiting Time Variance in the Single Machine Problem*, Management Science, Vol. 23, pp. 567-575, 1977.

[34] M. Harchol-Balter, K. Sigman, A. Wierman, *Understanding the slowdown of large jobs in an M/GI/1 system*, ACM SIGMETRICS Performance Evaluation Review, Vol. 30, Issue 3, pp. 9-11, 2002.

[35] S. Majumdar, D.L. Eager, R.B. Bunt, *Scheduling in multiprogrammed parallel systems*, ACM SIGMETRICS Performance Evaluation Review, Vol. 16, Issue 1, pp. 204-113, 1988.

[36] A.W. Mu'alem, D.G. Feitelson, *Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling*, IEEE Transactions on Parallel and Distributed Systems, Vol. 12, pp. 529-543, 2001.

[37] D.A. Lifka, *The ANL IBM SP Scheduling System*, Lecture Notes in Computer Science, Vol. 949, pp. 295-303, 1995.

[38] C. Ernemann, M. Krogmann, R. Yahyapour, *Scheduling on the Top 50 machines*, Lecture Notes in Computer Science, Vol. 3277, pp. 17-46, 2004.

[39] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, *Selective Reservation Strategies for Backfill Job Scheduling*, Lecture Notes in Computer Science, Vol. 2537, pp. 55-71, 2002.

[40] B.G. Lawson, *Self-adapting backfilling scheduling for parallel systems*, In Proceedings of the International Conference on Parallel Processing, pp. 583-592, 2002.

[41] E. Shmueli and D.G. Feitelson, *Backfilling with lookahead to Optimize the Performance of Paralell Job Scheduling*, Lecture Notes in Computer Science, Vol. 2862, pp. 228-251, 2003.

[42] J. Hungershofer, *On the Combined Scheduling of Malleable and Rigid Jobs*, In Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing, pp. 206-213, 2004.

[43] W. Cirne, C. Grande, F. Berman, *When the Herd is Smart Aggregate Behavior in the Selection of Job Request*, IEEE Transactions in Parallel and Distributed Systems, Vol. 14, pp. 181-192, 2003.

[44] L. Barsanti, A. Sodan, *Adaptive Job Scheduling via Predictive Job Resource Allocation*, Lecture Notes in Computer Science, Vol. 4376, pp. 115-140, 2007.

[45] B.N. Chun, *Market-based cluster resource management*, PhD thesis, University of California, Berkley, 2001.

[46] R. Raman, M. Livny and M. Solomon, *Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching*, In the Proceedings of International Symposium on High Performance Distributed Computing, pp. 80-89, 2003.

[47] A.J. Page, T.J. Naughton, *Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing*, In the Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing, 2005.

[48] M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, 1998.

[49] I. Grudenić, N.Bogunović, *Computer Cluster and Grid Simulator*, In the Proceedings of the Joint Conferences Computers in Technical systems and Intelligent systems, pp. 44-49, 2009.

[50] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley Professional, 2003.

[51] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, *The WEKA data mining software: an update*, ACM SIGKDD Explorations Newsletter, Vol. 1, Issue 1, pp. 10-18, 2009.

[52] W. H . Bell, D. G . Cameron, L. Capozza, A.P. Millar et al., *OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies*, International Journal of High Performance Computing Applications, Vol. 17, 2003.

[53] H. Casanova, *SimGrid: A toolkit for the simulation of application scheduling*, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 430-437, 2001.

[54] R. Buyya, M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing* , The Journal of Concurrency and Computation: Practice and Experience (CCPE), Vol. 14, 2002.

[55] A. Takefusa, H. Casanova, S. Matsuoka, F. Berman, *A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid*, In Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing, pp. 406-415, 2001.

[56] F. Howell, R. Mcnab, *simjava: A discrete event simulation library for Java*, In International Conference on Web-Based Modeling and Simulation, pp. 51-56, 1998.

[57] M. Pidd, *Computer Simulation in Management Science*, John Wiley & Sons, 2004.

[58] M. Pidd, R. A. Cassel, *Three phase simulation in Java*, In Proceedings of the 1998 Winter Simulation Conference, pp. 367-371, 1998.

[59] J. Han, M. Kamber, J. Pei, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2005.

[60] L. Breiman, J. Friedman, C.J. Stone, R.A.Olshen, *Classification and Regression Trees*, Chapman and Hall, 1984.

[61] J.R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann, 1993

[62] J.R. Quinlan, *Induction of Decision Trees*, Machine Learning, pp. 81-106, 1986.

[63] G. Biau , L. Devroye , G. Lugosi, *Consistency of Random Forests and Other Averaging Classifiers*, The Journal of Machine Learning Research, Vol. 9, pp. 2015-2033, 2008.

[64] L. Breiman, *Random Forests*, Machine Learning, pp. 5-32, 2001.

[65] D.G. Feitelson, B. Nitzberg, *Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*, Lecture Notes in Computer Science, Vol. 949, pp. 337-360, 2006.

[66] C.B. Lee, Y. Schwartzman, J. Hardy, A. Snavely, *Are user runtime estimates inherently inaccurate?*, Lecture Notes in Computer Science, Vol. 3277, p. 253, 2005.

[67] R. Gibbons, *A Historical Application Profiler for Use by Parallel Schedulers*, Proceedings of the Job Scheduling Strategies for Parallel Processing, pp.58-77, 1997.

[68] W. Smith, I.T. Foster, V.E. Taylor, *Predicting Application Run Times Using Historical Information*, In the Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, pp.122-142, 1998.

[69] S. Krishnaswamy, S. W. Loke, A. Zaslavsky, *Estimating computation times of data-intensive applications,* IEEE Distributed Systems Online, Vol. 5, Issue 4, 2004.

[70] L. Polkowski, *Rough Sets: Mathematical Foundations*, Physica-Verlag HD, 2002.

[71] N.H. Kapadia, J.A.B. Fortes, C.E. Brodley, *Predictive Application-Performance Modeling in a Computational Grid Environment*, In the Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, 1999.

[72] D. Tsafrir , Y. Etsion , D.G. Feitelson, *Backfilling Using System-Generated Predictions Rather than User Runtime Estimates*, IEEE Transactions on Parallel and Distributed Systems, Vol.18, Issue 6, pp.789-803, 2007.

POPIS INTERNET ADRESA

(Dostupnost dokumenata s Internet adresa provjerena je u rujnu 2010.)

[73] *TOP500 Supercomputing Sites*, http://www.top500.org/

[74] *IEEE 802.3 LAN/MAN CSMA/CD (Ethernet) Access Method*, http://standards.ieee.org/getieee802/802.3.html

[75] *MPI-2: Extensions to the Message-Passing Interface*, http://www.mpi-forum.org/docs/mpi2-report.pdf

[76] *The Open Group -- the Single UNIX Specification, 2008 Edition Core Volumes*, http://www.unix.org/2008edition/

[77] *Parallel Workloads Archive*, http://www.cs.huji.ac.il/labs/parallel/workload/

[78] Moab Workload Manager - Administrator's Guide, http://www.clusterresources.com/products/mwm/docs/MWMAdminGuide5_4.pdf

[79] Maui Administrator's Guide, http://www.clusterresources.com/products/maui/docs/mauiadmin.pdf

[80] OpenPBS administrator guide, http://www.nas.nasa.gov/Software/PBS/pbsdocs/admin_guide.ps

[81] Administering grid engine, http://wikis.sun.com/display/gridengine62u6/Administering

[82] IBM Tivoli Workload Scheduler LoadLeveler - Using and Administering, http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.loadl41j.admin.doc/c2366811.pdf

[83] *.NET Framework Conceptual Overview,* http://msdn.microsoft.com/library/zw4w595w.aspx

[84] *IKVM.NET Home Page*, http://www.ikvm.net

[85] *Main Page – Mono*, http://mono-project.com/Main_Page

[86] *BeoSim - Multicluster Computational Grid Simulator for Parallel Job Scheduling Research*, http://www.parl.clemson.edu/~wjones/research/

[87] Visual Studio 2008 Team System, http://www.microsoft.com/croatia/developers/visualstudio2008/teamsystem.mspx

[88] Srce: Klaster Isabella, http://www.srce.hr/isabella/

POPIS KORIŠTENIH OZNAKA

D	Skup instanci za učenje klasifikatora
X	Vektor vrijednosti atributa određene instance kod postupka klasifikacije
$A_1..A_n$	Atributi skupa instanci kod postupka klasifikacije
A_c	Ciljni atribut instanci u postupku klasifikacije
$C_1..C_n$	Klase (razredi) ciljnog atributa A_c
$P(A\|B)$	Uvjetna vjerojatnost događaja A ukoliko je poznat događaj B
$\|C_{i,D}\|$	Broj instanci klase C_i u skupu za učenje D
$\alpha \mid \beta \mid \gamma$	Postupak raspoređivanja poslova tipa β na vrsti resursa α, uz ciljnu metriku γ
p, p_i	Posao u računalnom sustavu
o_i	Vrijeme otpuštanja posla p_i
promjenjivo_nt	Vrsta posla kod koje nije poznato trajanje u trenutku dolaska u sustav
promjenjivo_pt	Vrsta posla kod je poznato trajanje u trenutku dolaska u sustav
P_m	Homogeni paralelni sustav sa m resursa
P	Skup svih poslova
$T(p)$	Trajanje posla p
$C(p)$	Vrijeme proteklo od dolaska posla p u sustav do početka njegovog izvođenja

POPIS KORIŠTENIH KRATICA

SARG	*Sustav za simulaciju i analizu računalnog grozda*
SISD	*Single instruction, single data*
SIMD	*Single instruction, multiple data*
MISD	*Multiple instruction, single data*
MIMD	*Multiple instruction, multiple dana*
UMA	*Uniform memory access*
NUMA	*Non-uniform memory access*
ccNUMA	*Cache coherent non-uniform memory access*
COMA	*Cache only memory architecture*
MPP	*Massively parallel processor*
PVM	*Parallel virtual machine*
MPI	*Message Passing Interface*
CART	*Classification and Regression Trees*
SWF	*Standard Workload Format*
LFA	*Lista fiksiranih aktivnosti*
LUA	*Lista uvjetnih aktivnosti*
PDP	*Prioritiziranje (trenutka) dolaska posla (engl. FCFS)*
PŠP	*Prioritiziranje (naj)šireg posla*
PUP	*Prioritiziranje (naj)užeg posla*
PNP	*Prioritiziranje najkraćeg posla (engl. SJF)*
KUPP	*Konzervativni (postupak) unazadnog popunjavanja praznina*
AUPP	*Agresivni (postupak) unazadnog popunjavanja praznina*
UPPX	*Unazadno popunjavanje praznina (sa) X (fiksnih rezervacija)*
AppLes	*Application Level Scheduler*
GAOVU	*Genetski algoritam (sa) ograničenjem veličine ulaza*
UPPDP	*Unazadno popunjavanje praznina dinamičkim programiranjem*

UPPDP_PR *Unazadno popunjavanje praznina dinamičkim programiranjem (uz preferenciju) prioriteta*

UPPDP_PR *Unazadno popunjavanje praznina dinamičkim programiranjem (uz preferenciju) uskih poslova*

UPPDP_SP *Unazadno popunjavanje praznina dinamičkim programiranjem (uz preferenciju) širokih poslova*

HOPP *Heuristički određivanje profila poslova*

DKTP *Dinamički (postupak) klasifikacije trajanja poslova*

UPP1MX *Inačica X postupka unazadnog popunjavanja praznina s jednom fiksnom rezervacijom*

SAŽETAK

Prilagodljivo dinamičko raspoređivanje skupnih poslova na grozdu računala

U disertaciji su razmatrani različiti aspekti vezani uz raspoređivanje poslova na grozdovima računala. Istraživanje postupaka raspoređivanja poslova oslanja se na simulacije paralelnih računalnih sustava te je u sklopu rada oblikovan simulator koji omogućava aktivno sudjelovanje poslova u simuliranom sustavu. Provedena je analiza postojećih postupaka raspoređivanja poslova i izgrađen je učinkovit izvorni algoritam zasnovan na dinamičkom programiranju i postupku unazadnog popunjavanja praznina. Poslovi koji dolaze na računalne grozdove okarakterizirani su lošim procjenama trajanja, pri čemu je nezanemariv dio tih poslova neispravan. U svrhu poboljšanja učinkovitosti raspoređivanja analizirane su mogućnosti predviđanja neispravnih poslova i predviđanja trajanja poslova statističkim metodama za dubinsku analizu podataka. Dobiveni rezultati predviđanja iskorišteni su u modificiranim postupcima raspoređivanja te su izmjerena poboljšanja učinkovitosti tih postupaka.

Ključne riječi: grozd računala, postupci raspoređivanja, trajanje poslova, neispravni poslovi, statističke metode za dubinsku analizu podataka.

ABSTRACT

Adaptive dynamic batch job scheduling in computer cluster

This thesis addresses different aspects of computer cluster scheduling. Discrete event simulator that enables representation of jobs as active simulation entities is designed to enable research of scheduling algorithms. Theoretical and experimental analysis of the existing scheduling algorithms is made, which is accompanied with a designed of the new and efficient scheduling algorithm based on dynamic programming and backfilling. Jobs in computer clusters are characterized by inaccurate runtime estimates and ineligible amount of jobs fails while being executed. In order to overcome this, methods for runtime and failure prediction are designed and their efficiency is measured. These predictions are used to improve efficiency of cluster schedulers that are modified to accommodate them.

Keywords: computer cluster, scheduling algorithms, runtime prediction, failure prediction, data mining.

ŽIVOTOPIS

Igor Grudenić rođen je 1979. godine u Rijeci gdje je pohađao osnovnu i srednju školu. Nakon mature 1997. godine upisuje Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu. Tijekom studija primio je nagradu „Josip Lončar“ kao jedan od najboljih studenata na trećoj godini studija. Diplomirao je 2002. godine s radom naslovljenim „Lokalni zastupnici u middleware sustavima“ te je zaposlen na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu. Iste godine upisao je poslijediplomski studij računarstva i 2006. godine magistrirao na temu „Paralelni algoritmi izgradnje i manipulacije binarnim dijagramima odlučivanja“.

U svom znanstvenom radu bavi se područjem formalne verifikacije i postupaka raspoređivanja poslova u paralelnim računalnim sustavima. U sklopu istraživanja sudjeluje na projektima „Raspodijeljeni ugrađeni računalni sustavi“ (0036051) i „Računalne okoline za sveprisutne raspodijeljene sustave“ (036-0362980-1921).

Tijekom znanstvenog rada objavio je dva članka u časopisima te jedanaest članaka na međunarodnim znanstvenim skupovima.

BIOGRAPHY

Igor Grudenić was born in Rijeka 1979 where he attended elementary school and high school. After graduation he enrolled the Faculty of Electrical Engineering and Computing at University of Zagreb. As a student he received „Josip Lončar“ award for being one of the best third year students. He graduated in 2002 with thesis „User agents in middleware systems”. That same year he joined Department of Electronics, Microelectronics, Computer and Intelligent Systems at Faculty of Electrical Engineering and Computing, Zagreb. He received MS degree in Computer Science in 2006 with thesis „Parallel algorithms for construction and manipulation of binary decision diagrams”.

His research interests include formal verification methods and scheduling for distributed systems. He was involved in research projects „Distributed embedded systems” (0036051) and „Computing Environments for Ubiquitous Distributed Systems” (036-0362980-1921) supported by the Croatian Ministry of Science and Technology.

His publications include several conference and journal papers.