Izgradnja modela za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa poslužitelja

Alagić, Dino

Doctoral thesis / Disertacija

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:971877

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Dino Alagić

IZGRADNJA MODELA ZA AUTOMATIZIRANU I POBOLJŠANU ISKORISTIVOST POSTOJEĆIH RAČUNALNIH RESURSA POSLUŽITELJA

DOKTORSKI RAD

Varaždin, 2019.

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Dino Alagić

IZGRADNJA MODELA ZA AUTOMATIZIRANU I POBOLJŠANU ISKORISTIVOST POSTOJEĆIH RAČUNALNIH RESURSA POSLUŽITELJA

DOKTORSKI RAD

Mentor: Izv. prof. dr. sc. Ivan Magdalenić

Varaždin, 2019.

University of Zagreb

Faculty of Organization and Informatics

Dino Alagić

BUILDING A MODEL FOR AUTOMATED AND IMPROVED UTILIZATION OF EXISTING SERVER RESOURCES

DOCTORAL DISSERTATION

Supervisor: Assoc. Prof. Ivan Magdalenić, Ph. D.

Varaždin, 2019.

Informacija o mentoru

Ivan Magdalenić rođen je 17. travnja 1977. godine u Čakovcu. Nakon završetka prirodoslovno-matematičke gimnazije 1995. godine upisao je diplomski studij na Fakultetu elektrotehnike i računarstva, Sveučilište u Zagrebu. Diplomirao je 2000. godine na smjeru Telekomunikacije i informatika. Magistrirao je 2003. godine s temom magistarskog rada "Elektronička razmjena poslovnih dokumenata". Od 2000. do 2004. godine radio je kao znanstveni novak na Fakultetu elektrotehnike i računarstva Sveučilište u Zagrebu. Od 2004. godine radi kao asistent na Fakultetu organizacije i informatike Sveučilište u Zagrebu. Doktorirao je na Fakultetu elektrotehnike i računarstva u znanstvenom polju Računarstvo s temom doktorske disertacije "Dinamičko generiranje ontološki podržanih usluga Weba za dohvat podataka". Izabran je u znanstveno nastavno zvanje docent 2012. godine, a u znanstveno-nastavno zvanje izvanredni profesor 2018. godine.

Područje znanstvenog istraživanja Ivana Magdalenića uključuje automatsko i generativno programiranje, elektroničko poslovanje, Semantički web i druge napredne tehnologije weba. Ivan Magdalenić je autor sveučilišnog udžbenika, objavio je poglavlje u znanstvenoj knjizi te je autor devetnaest radova objavljenih u znanstvenim časopisima te dvadeset i dva rada objavljenih na međunarodnim konferencijama.

Aktivno sudjeluje u znanstvenim i stručnim projektima uvođenja elektroničkog poslovanja u Republici Hrvatskoj. Također je član nacionalnih tijela kojima je zadatak podizanje svijesti o elektroničkom poslovanju i uvođenje elektroničkog poslovanja u gospodarstvo i javnu upravu i to Član Nacionalnog više dioničkog Foruma za e-račun, Član Nacionalnog vijeća za elektroničko poslovanje, Član Sektorskog odbora za akreditaciju davatelja usluga certificiranja u području primjene e-potpisa pri Hrvatskoj akreditacijskoj agenciji te Član povjerenstva za e-račun i voditelj Tehničkog odbora projekta e-Račun pri Ministarstvo gospodarstva, rada i poduzetništva. Pristupnik je sudjelovao na nizu stručnih konferencija na tematiku podizanja znanja i popularizaciji elektroničkog poslovanja e-biz i CUC.

Obnašao je dužnost pročelnika Katedre za informatičke tehnologije i računarstvo na Fakultetu organizacije i informatike Sveučilište u Zagrebu u razdoblju od 2013.-2017.

Dobitnik nagrade za mladog znanstvenika na Fakultetu organizacije i informatike Sveučilište u Zagreb 2012. godine.

Zahvala

"Inventas vitam iuvat excoluisse per artes."
("Let us improve life through science and art.")

- Vergilije, rimski pjesnik

PODACI O DOKTORSKOM RADU

I. AUTOR

Ime i prezime	Dino Alagić
Datum i mjesto rođenja	10.03.1989. Bihać, BiH
Naziv fakulteta i datum diplomiranja na VII/I stupnju	Fakultet organizacije i informatike, Varaždin, Sveučilište u Zagrebu, 6.7.2010.
Naziv fakulteta i datum diplomiranja na VII/II stupnju	Fakultet organizacije i informatike, Varaždin, Sveučilište u Zagrebu, 19.6.2012.
Sadašnje zaposlenje	NTH Mobile d.o.o.

II. DOKTORSKI RAD

Naslov	Izgradnja modela za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa poslužitelja
Broj stranica, slika, tabela, priloga, bibliografskih podataka	131 stranica, 5 tablica, 70 slika, 3 priloga i 96 bibliografskih podataka
Znanstveno područje i polje iz kojeg je postignut doktorat znanosti	Društvene znanosti, informacijske znanosti
Mentori ili voditelji rada	Izv. prof. dr. sc. Ivan Magdalenić
Fakultet na kojem je obranjen doktorski rad	Fakultet organizacije i informatike
Oznaka i redni broj rada	149

III. OCJENA I OBRANA

Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena tema	19.09.2017.
Datum predaje rada	09.03.2018.
Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena pozitivna ocjena rada	11.12.2018.
Sastav povjerenstva koje je rad ocijenilo	Prof. dr. sc. Dragutin Kermek (predsjednik Povjerenstva) Prof. dr. sc. Marin Golub Doc. dr. sc. Nikola Ivković
Datum obrane doktorskog rada	23.01.2019.
Sastav povjerenstva pred kojim je rad obranjen	Prof. dr. sc. Dragutin Kermek (predsjednik Povjerenstva) Prof. dr. sc. Marin Golub Doc. dr. sc. Nikola Ivković
Datum promocije	

Sažetak

Zbog ubrzanog razvoja informacijske tehnologije su nastali složeni sustavi kao što su računarstvo u oblaku. Navedeni sustavi najčešće moraju imati visoku razinu dostupnosti podataka, odnosno moraju osigurati neprekidni rad poslovnih sustava. Kako bi se to postiglo prisutni su visoki kapitalni i operativni troškovi podatkovnih centara koji su neophodni za ovakvu vrstu usluga. Mnogobrojna istraživanja na ovu temu ukazuju kako su poslužitelji glavni uzročnici visokih troškova podatkovnih centara. Upravo se zbog toga njihovi resursi nastoje što učinkovitije iskoristiti. U istraživanju su navedene Web-farme kao primjer iz prakse koji potvrđuje da postoje sustavi čiji poslužitelji nedovoljno iskorištavaju svoje resurse, ali ih moraju imati kako bi osigurali visoku razinu dostupnosti sustava. Spomenuti visoki troškovi i nedovoljna iskoristivost postojećih računalnih resursa su glavna motivacija za ovo istraživanje. Nakon proučavanja dosadašnjih znanstvenih istraživanja, ali i rješenja iz prakse, utvrđeno je da ne postoji rješenje koje bi dovoljno učinkovito riješilo ovaj problem. U radu se predlaže novi model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa bez potrebe za ponovnim pokretanjem poslužitelja koji rješava navedeni problem. Na temelju modela je napravljena aplikacija koja je validirana na primjeru Web-poslužitelja gdje je ovaj problem prepoznat. U radu se koristi istraživačka paradigma znanost o dizajnu (engl. *Design Science Research Methodology*, *DSRM*), koja se temelji na kreiranju novog artefakta što u ovom slučaju predstavlja novi model.

Ključne riječi: računarstvo u oblaku, virtualizacija, model, Web-farma, HTTP, algoritam, poslužitelj

Proširenisažetak

Informacijska tehnologija je pod stalnim pritiskom inovacija kako bi omogućila što veću razinu dostupnosti podataka, odnosno neprekidni rad poslovnih sustava. Upravo je to rezultiralo ubrzanim razvojem složenih sustava koji se zovu računarstvo u oblaku. Jedan od zadataka takvih rješenja je osiguravanje visoke razine dostupnosti složenih sustava i arhitektura. Kako bi takva rješenja ispravno funkcionirala prisutni su visoki kapitalni i operativni troškovi podatkovnih centara koji su neophodni za ovu vrstu usluga. Postoje mnogobrojna istraživanja koja ukazuju na to da su glavni uzročnici visokih troškova podatkovnih centara poslužitelji. Zbog toga se poslužitelji odnosno njihovi resursi nastoje što učinkovitije iskoristiti.

U radu su navedeni primjeri iz prakse koji potvrđuju da postoje sustavi čiji poslužitelji nedovoljno iskorištavaju svoje resurse, ali ih moraju imati zbog svoje važnosti. Konkretan primjer ovog problema su Web-farme gdje je u svrhu postizanja veće dostupnosti sustava prisutna veća količina resursa nego li je to stvarno potrebno što potvrđuju i alati za mjerenja opterećenosti poslužitelja. Ovaj pristup omogućava da sustav lakše podnese iznenadna opterećenja što povećava razinu dostupnosti sustava. Negativan učinak ovakvog pristupa predstavlja povećanje kapitalnih i operativnih troškova zbog veće količine računalnih resursa. Spomenuti visoki troškovi i nedovoljna iskoristivost postojećih računalnih resursa su ujedno i glavna motivacija za ovo istraživanje. Kako bi se ovaj problem riješio potrebno je imati sustav koji bi automatski dodjeljivao onoliko računalnih resursa koliko je potrebno sustavu ovisno o njegovoj opterećenosti pazeći pritom na njegovu dostupnost i konzistentnost.

Tijekom detaljnog proučavanja dosadašnjih znanstvenih istraživanja, ali i rješenja iz prakse, utvrđeno je da ne postoji rješenje koje bi dovoljno učinkovito riješilo ovaj problem što je ujedno bila i dodatna motivacija da se ovo istraživanje provede. Nedostatak postojećih rješenja je to što se oni ne bave kako učinkovitije iskoristiti postojeće resurse već u kritičnim situacijama najčešće dodaju nove ili vrše migraciju virtualnih poslužitelja na druge fizičke poslužitelje za što je potrebno još više računalnih resursa. Drugi pristup rješavanju ovog problema je prioritizacija procesa, odnosno da se poslužiteljima kojima su prijeko potrebni resursi dodjeli najveći prioritet u izvršavanju procesa. Nedostatak ovog pristupa je to što se resursi ne mogu povećati ili smanjiti već samo prioritizirati što i dalje rezultira da su prisutni resursi koji se ne koriste. Jedan od nedostataka postojećih rješenja je to što nije moguće obaviti dodavanja i oduzimanje računalnih resursa (CPU i memorije) bez da je potrebno ponovno pokretanje poslužitelja. Veliki broj postojećih rješenja ima fokus samo na CPU ili memoriju, ali ne i na oboje. Zbog svega navedenog odlučeno je da se izgradi novi model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa. Na temelju modela je napravljena

aplikacije koja ujedno služi i za validaciju na primjeru Web-poslužitelja gdje je ovaj problem i prepoznat.

U provođenju ovog istraživanja koristila se istraživačka paradigma znanost o dizajnu (engl. *Design Science Research Methodology*, *DSRM*), koja se koristi u informacijskim tehnologijama i nudi specifične smjernice za procjenu i iteraciju unutar istraživačkih projekata. Metodologija se temelji na kreiranju novog artefakta što u ovom slučaju predstavlja novi model koji rješava navedene složene probleme. Istraživačka paradigma znanost o dizajnu se sastoji od šest slijednih procesnih koraka, a oni su: prepoznavanje problema i motivacija, definiranje ciljeva, dizajn i razvoj, prikaz rješenja, vrednovanje i komunikacija. Tijekom ovih koraka koristile su se mnogobrojne metode i tehnike kao što su: uspoređivanje, evaluacija/validacija, analiza sadržaja, eksperiment, tehnike modeliranja (UML), dijagramske tehnike (dijagram uzročno-posljedičnih veza), strukturna analiza procesa (dijagrami dekompozicije, dijagrami toka podataka i blok dijagram), programiranje (pseudojezik i skriptni jezici (BASH i PHP)) i mnoge druge.

U pogledu znanstvenog doprinosa, ovo istraživanje je rezultiralo s novim modelom za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa bez potrebe za ponovnim pokretanjem poslužitelja, ali i jasno definiranim slučajevima i ograničenjima kada je novi model primjenjiv. Istraživanje je pokazalo da primjena novog modela omogućava učinkovitije iskorištenje postojećih računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja. Također su navedene preporuke za ostvarivanje modela u odabranom programskom jeziku i postupak vrednovanja modela na eksperimentima. U svrhu društvenog doprinosa cijelo rješenje je otvorenog koda što je ujedno i jedan od glavnih ciljeva ovog istraživanja. Ovo rezultira lakšom primjenom rješenja, ali i ponovljivost ispitivanja kako bi se omogućila što lakša daljnja unapređenja i istraživanja na ovu temu.

Ključne riječi: računarstvo u oblaku, virtualizacija, model, Web-farma, HTTP, algoritam, poslužitelj

Extended abstract

Information technology is under constant innovation pressure to provide the highest level of data availability, i.e. the continuous functioning of operating systems. This is the very reason for an accelerated development of complex systems called cloud computing. One of the tasks of such solutions is to ensure high-level availability of complex systems and architecture. In order for such solutions to function properly, the high capital and operational costs of data centers are essential for this type of service. There are numerous studies which indicate that servers are the main cause of data centers' high cost. As a result, the aim is to use servers, i.e. their resources more efficiently.

This paper shows the examples from practice which confirm that there are systems whose servers insufficiently exploit their resources, but they must have them due to their importance. A concrete example of this problem are the Web Farms where, in order to achieve greater system availability, there is a greater amount of resources than is really needed, as confirmed by tools for measuring server loads. This approach allows the system to withstand sudden loads, which increases the level of system availability. The negative effect of such an approach is the increase in capital and operating costs due to a higher amount of computer resources. The mentioned high costs and inadequate utilization of the existing computer resources are at the same time the main motivation for this research. To solve this problem, it is necessary to have a system which would automatically allocate as much computer resources as the system needs, depending on its load and thereby taking into account its availability and consistency.

During a detailed study of the current scientific research, as well as practical solutions, it has been found that there is no effective solution to this problem, which also served as an additional motivation for this research to be carried out. The existing solutions are lacking in that they are not dealing with how to use the existing resources more efficiently, but in adding new or migrating virtual servers to other physical servers in critical situations, which requires even larger numbers of computer resources. The second approach to solving this problem is process prioritization, i.e. that servers with the greatest need for resources are given the highest priority in the execution of the process. The disadvantage of this approach is that resources cannot be increased nor decreased, but only prioritized, which still results in the presence of unused resources. One of the disadvantages of the existing solutions is that it is not possible to add and subtract computer resources (CPU and memory) without the need to restart the server. A large number of existing solutions focus only on CPU or memory, but not on both. Due to all this, a decision was made to build a new model for an automated and improved utilization of the existing computing resources. The model will be verified by building an application that will also serve for validation on the Web server example where this problem was recognized.

The research paradigm used in this research is the Design Science Research Methodology (DSRM), which has specific guidelines for evaluation and iteration within research projects. The methodology is based on the creation of a new artifact. In this case that is a new model which addresses these complex problems mentioned in this case. The Design Science Research Methodology consists of six sequential process steps, which are: identification of problems and motivations, a definition of goals, design, and development, presentation of solutions, evaluation and communication. Throughout these steps, numerous methods and techniques were used such as: comparison, evaluation/validation, content analysis, experiment, modelling techniques (UML), diagram techniques (causal relationship diagrams), structural analysis of processes (decomposition diagrams, data flow charts, and block diagram), programming (pseudocode and scripting languages (BASH and PHP), as well as many others.

With regard to scientific contributions, this research has resulted with a new model for an automated and improved utilization of the existing computing resources without the need to restart the server, as well as in clearly defined cases and constraints regarding the new model's application. The research has shown that the application of a new model enables a more efficient utilization of the existing computing resources (CPU and memory) without the need to restart the server. The research also provides recommendations for the implementation of the model in the selected programming language, and the process of evaluating the model in the experiments. In view of the social contribution, the whole solution is open source, which is also one of the main goals of this research. This results in an easier application of the solution and the repeatability of the testing to facilitate further improvement and research on this topic.

Keywords: cloud computing, virtualization, model, Web farm, HTTP, algorithm, server

Sadržaj

1 Uvod ... 1
2 Predmet istraživanja ... 5
3 Pregled postojećih rješenja i dosadašnjih istraživanja ... 10
4 Svrha i motivacija za istraživanje ... 20
5 Ciljevi, istraživačka pitanja i hipoteze ... 21
6 Znanstveni i društveni doprinos rada ... 22
7 Model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa ... 24
7.1 Utvrđivanje problema i motivacija ... 32
7.2 Cilj rješenja ... 33
7.3 Dizajn i razvoj ... 33
7.3.1 Konceptualni model ... 35
7.3.1.1 Konfiguracija ... 37
7.3.1.2 Agent ... 41
7.3.1.3 Ograničenja sustava ... 43
7.3.1.4 Metoda dodjele resursa ... 44
7.3.1.5 Kontrola sustava ... 47
7.4 Prikaz rješenja ... 51
7.5 Vrednovanje ... 55
7.5.1 Aktivnost 1. - Određivanje testnog okruženja ... 56
7.5.2 Aktivnost 2. - Određivanje kriterija prihvaćanja ... 59
7.5.3 Aktivnost 3. - Planiranje i dizajn testova ... 61
7.5.4 Aktivnost 4. - Konfiguracija testnog okruženja ... 62
7.5.5 Aktivnost 5. - Ostvarivanje testnog dizajna ... 65
7.5.6 Aktivnost 6. - Provesti testiranje ... 71
7.5.6.1 Faza I - validacija prve hipoteze ... 72
7.5.6.2 Faza II - validacija druge hipoteze ... 77
7.5.6.3 Faza III - validacija treće hipoteze ... 78
7.5.7 Aktivnost 7. - Analizirati rezultate, izvješća i ponoviti testiranje ... 80
7.5.7.1 Faza I - Rezultat validacije prve hipoteze ... 80
7.5.7.2 Faza II - Rezultat validacije druge hipoteze ... 87
7.5.7.3 Faza III - Rezultat validacije treće hipoteze ... 91
7.6 Komunikacija ... 96
8 Zaključak i smjernice za buduće istraživanje ... 97
Literatura ... 99
Popis priloga ... 104
Životopis ... 113
Popis radova ... 114

Popis slika

Slika 1.1. - Ukupni trošak vlasništva podatkovnog centra [6] 1
Slika 1.2. - Dijagram uzročno-posljedičnih veza sustava 2
Slika 2.1. - Shematski prikaz komunikacije između krajnjeg korisnika i Web-poslužitelja 5
Slika 2.2. - Shematski prikaz komunikacije između krajnjeg korisnika i Web-farme 6
Slika 2.3. - Arhitektura Web-farme 10
Slika 7.1. - Procesni model istraživačke paradigme znanosti o dizajnu [90] 24
Slika 7.2. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog dana (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin* 26
Slika 7.3. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog tjedna (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin* 26
Slika 7.4. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog mjeseca (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin* 27
Slika 7.5. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jedne godine (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin* 27
Slika 7.6. - Grafički prikaz dijela Linux sučelja nakon pokretanja naredbe *htop* 28
Slika 7.7. - Grafički prikaz opterećenja mosta 30
Slika 7.8. - Prikaz stvarnog prosječnog opterećenja procesora za četiri Web-klastera tijekom 24 sata prema naredbi *htop* 31
Slika 7.9. - Simulacija ukupnog stvarnog prosječnog opterećenja procesora prema za četiri Web-klastera tijekom 24 sata prema naredbi *htop* 32
Slika 7.10. - Arhitektura Web-farmi 34
Slika 7.11. - Metamodel sustava automatskog upravljanja 36
Slika 7.12. - ERA model novog modela 36
Slika 7.13. - Pseudo jezik za početne parametre 39
Slika 7.14. - Raspodjela agenata na poslužiteljima 42
Slika 7.15. - Postupak izračuna ukupnog broja jezgri i memorije fizičkog poslužitelja koji su dodijeljeni virtualnim poslužiteljima 42
Slika 7.16. - Postupak za provjeru resursa virtualnog poslužitelja 43
Slika 7.17. - Postupak za LA (Load Average) mehanizam 45
Slika 7.18. - Postupak za mehanizam INCREMENT 47
Slika 7.19. - Postupak za slučaj kada za CPU nije definiran LA način dodjele resursa 47
Slika 7.20. - Postupak provjere i održavanja neprekidnosti sustava 48
Slika 7.21. - Postupak za provjeru resursa fizičkog poslužitelja 49
Slika 7.22. - Postupak za provjeru resursa virtualnog poslužitelja 50
Slika 7.23. - Postupak za kontrolni mehanizam koji šalje obavijesti 50
Slika 7.24. - Slojevita arhitektura novog modela 51
Slika 7.25. - Dijagram aktivnosti novog modela 52
Slika 7.26. - Prikaz novog rješenja na primjeru Web-poslužitelja 54
Slika 7.27. - Arhitektura testnog okruženja nad kojem su provedeni eksperimenti 57
Slika 7.28. - Prikaz testnog okruženja nad kojim je ostvaren novi model 62
Slika 7.29. - Tijek podataka u testnom okruženju između krajnjeg korisnika i Web-stranice. 62
Slika 7.30. - Početni prikaz WordPress Web-stranice 64
Slika 7.31. - Prikaz postavki eksperimenta u alatu *Apache JMeter* 66

Slika 7.32. - Broj zahtjeva u sekundi za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta) 67
Slika 7.33. - Vrijeme odziva za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja vrijeme odziva izraženo u milisekundama, a apscisa vrijeme trajanja eksperimenta) 68
Slika 7.34. - Broj učitavanja u sekundi za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj učitavanja zahtjeva u sekundi, a apscisa vrijeme) 68
Slika 7.35. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu (apscisa) za izvršeni eksperiment u alatu *Munin* 69
Slika 7.36. - Iskoristivost resursa procesora za izvršeni eksperiment (ordinata predstavlja postotak opterećenja procesora po broju jezgri gdje 100 predstavlja jednu jezgru, a apscisa vrijeme) u alatu *Munin* 70
Slika 7.37. - Iskoristivost memorije za izvršeni eksperiment (ordinata predstavlja memoriju izraženu u MB, a apscisa vrijeme) u alatu *Munin* 71
Slika 7.38. - Prikaz postavki u alatu *Apache JMeter* za prvi eksperiment 73
Slika 7.39. - Prikaz parametara u alatu *Apache JMeter* za prvi eksperiment 74
Slika 7.40. - Prikaz postavki u alatu *Apache JMeter* za drugi eksperiment 75
Slika 7.41. - Prikaz parametara u alatu *Apache JMeter* za drugi eksperiment 75
Slika 7.42. - Prikaz postavki u alatu *Apache JMeter* za treći eksperiment za *apache* Web-klaster 76
Slika 7.43. - Prikaz postavki u alatu *Apache JMeter* za treći eksperiment za *nginx* Web-klaster 77
Slika 7.44. - Primjer parametara i vrijednosti virtualnog poslužitelja koje dohvaća fizički poslužitelj 78
Slika 7.45. - Prikaz parametara u alatu *Apache JMeter* 79
Slika 7.46. - Broj zahtjeva u sekundi za prvi eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj izvršenih zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta) 80
Slika 7.47. - Iskoristivost resursa procesora za prvi eksperiment (ordinata predstavlja postotak opterećenja jezgri gdje 100 predstavlja jednu jezgru, a apscisa vrijeme) u alatu *Munin* 81
Slika 7.48. - Iskoristivost memorije za prvi eksperiment (ordinata predstavlja memoriju izraženu u GB, a apscisa vrijeme) u alatu *Munin* 81
Slika 7.49. - Pregled računalnih resursa poslužitelja (dt-web1) pomoću naredbe *htop* za prvi eksperiment: prije testa (gornja slika), tijekom testa (srednja slika) i nakon testa (donja slika) 82
Slika 7.50. - Broj zahtjeva u sekundi za drugi eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj izvršenih zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta) 83
Slika 7.51. - Iskoristivost resursa procesora za drugi eksperiment (ordinata predstavlja postotak opterećenja jezgri gdje 100 predstavlja jednu jezgru, a apscisa vrijeme) u alatu *Munin* 83
Slika 7.52. - Iskoristivost memorije za drugi eksperiment (ordinata predstavlja memoriju izraženu u GB, a apscisa vrijeme) u alatu *Munin* 84
Slika 7.53. - Pregled računalnih resursa poslužitelja (dt-web1) pomoću naredbe *htop* za drugi eksperiment: prije testa (gornja slika), tijekom testa (srednja slika) i nakon testa (donja slika) 85
Slika 7.54. - Prikaz dijela zapisa za fizički poslužitelj 2 (host2) 86
Slika 7.55. - Prikaz dijela zapisa za fizički poslužitelj 3 (host3) 87

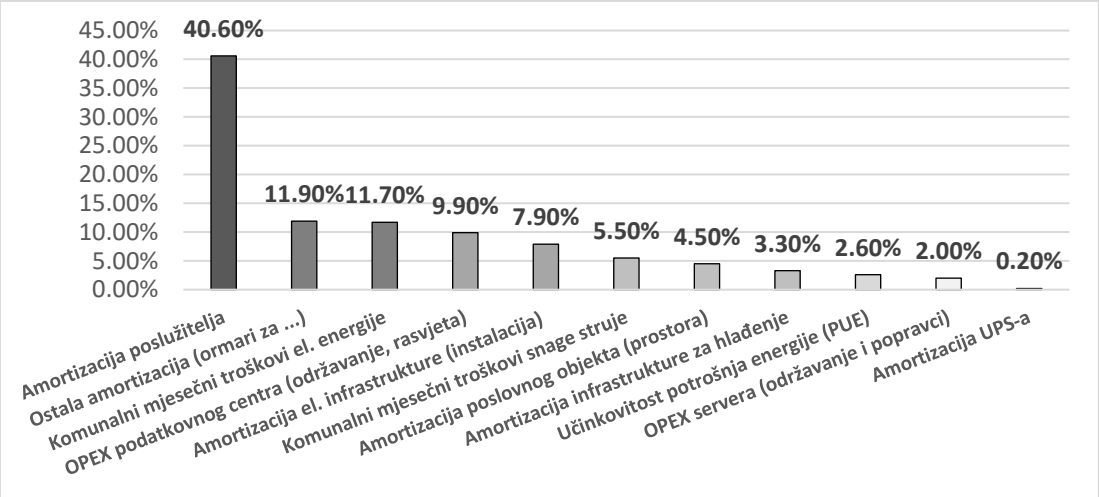
Slika 7.56. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno oduzimanje resursa procesora nakon čega je potvrđeno da Web-poslužitelj radi ispravno 88
Slika 7.57. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno dodavanja resursa procesora nakon čega je potvrđeno da Web-poslužitelj radi ispravno 89
Slika 7.58. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno oduzimanje memorije nakon čega je potvrđeno da Web-poslužitelj radi ispravno 90
Slika 7.59. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno dodavanje memorije nakon čega je potvrđeno da Web-poslužitelj radi ispravno..... 91
Slika 7.60. - Broj zahtjeva u sekundi za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj izvršenih zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta) 92
Slika 7.61. - Prikaz stvarnog prosječnog opterećenja procesora prema broju opterećenih jezgri po naredbi *htop* za izvršeni eksperiment bez primjene aplikacije koja je razvijena na temelju novog modela 93
Slika 7.62. - Prikaz opterećenja memorije i dodijeljene memorije prema naredbi *htop* za izvršeni eksperiment bez primjene aplikacije koja je razvijena na temelju novog modela 93
Slika 7.63. - Ulazni parametri modela koji su korišteni u eksperimentu 94
Slika 7.64. - Prikaz stvarnog prosječnog opterećenja procesora prema broju opterećenih jezgri po naredbi *htop* za izvršeni eksperiment s primjenom aplikacije koja je razvijena na temelju novog modela 95
Slika 7.65. - Prikaz opterećenja memorije i dodijeljene memorije prema naredbi *htop* za izvršeni eksperiment s primjenom aplikacije koja je razvijena na temelju novog modela 95

Popis tablica

Tablica 3.1. - Usporedba osobina postojećih komercijalnih i nekomercijalnih rješenja s novim predloženim rješenjem 18
Tablica 7.1. - Broj korisničkih zahtjeva u sekundi iz stvarnog sustava prema različitim vremenskim periodima za jednu Web-farmu 28
Tablica 7.2. - Sažetak osnovnih aktivnosti prema Microsoftov priručniku za testiranje Web-aplikacija [94] 56
Tablica 7.3. - Popis virtualnih poslužitelja i njihovih početnih karakteristika 58
Tablica 7.4. - Popis Web-klastera i pripadajući Web-domena 64

1 Uvod

U zadnjem desetljeću pojam računarstva u oblaku je sve češće u uporabi i može se reći da je kao takvo postalo važna karika svakog modernog poslovnog sustava. Računarstvo u oblaku predstavlja vrstu računarstva, koje se oslanja na dijeljenje računalnih resursa, počevši od računala i aplikacija pa sve do raznih srodnih usluga [1][2]. Takvi sustavi se najčešće sastoje od velikog broja poslužitelja (engl. *server*) i vrlo složene IT infrastrukture koja se nalazi u podatkovnim centrima. Razlog zbog čega su takvi sustavi složeni je globalizacija i liberalizacija tržišta na kojima nije prihvatljivo imati informacijski sustav koji nema visoku razinu dostupnosti. Upravo zbog toga, takvi podatkovni centri nisu jeftini i da bi jedan takav centar danas mogao opstati na tržištu potrebna su neprekidna ulaganja u rast i unapređenje sustava, odnosno prisutni su kapitalni troškovi (CAPEX) i operativni troškovi (OPEX) [3][4][5]. Primjer CAPEX troškova jednog podatkovnog centra predstavlja investicija u novu opremu, kao što su: poslužitelji, centralni sustav za pohranu podataka (engl. *storage*) ili mrežna oprema, dok OPEX predstavlja troškove održavanja takvog sustava, a to su: električna energija, klimatizacija prostora, Internet poveznice, održavanje i zamjena hardvera, najam prostora i sl. Sljedeći grafikon (Slika 1.1.) predstavlja ukupni trošak vlasništva jednog podatkovnog centra.



Slika 1.1. - Ukupni trošak vlasništva podatkovnog centra [6]

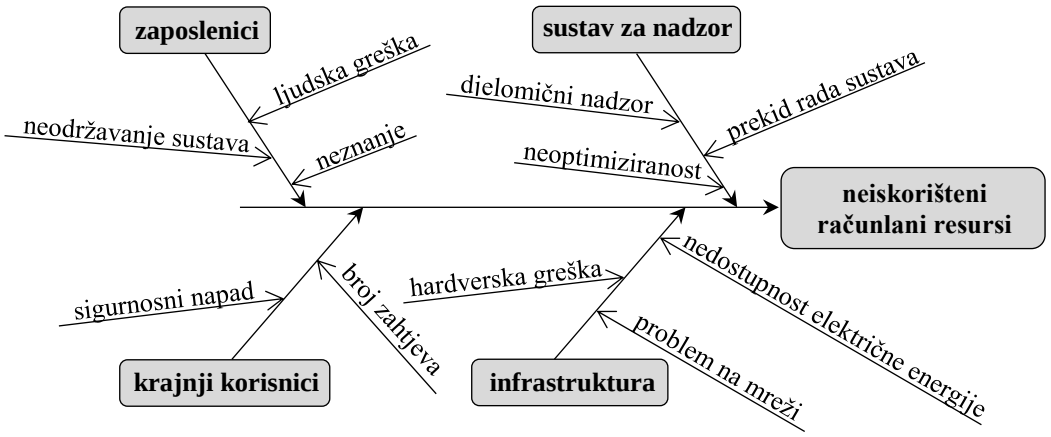
Zbog svih navedenih troškova, pružatelji takvih usluga nastoje već postojeće resurse višestruko iskoristiti uz pomoć automatizacije i optimizacije sustava. Postoje mnogobojne analize i istraživanja, koja potvrđuju da su poslužitelji najskuplja komponenta računarstva u oblaku [6][7][8]. Neki od razloga su:

- OPEX troškovi zbog njihovog održavanja (zamjena diskova, strujnog napajanja, ventilatora i sl.), električne energije, ali i hlađenja prostora koje raste sukladno povećanjem broja poslužitelja, … [9][10].

- Jedan od problema je što poslužitelji imaju najveću stopu amortizacije za razliku od ostatka opreme i infrastrukture jednog podatkovnog centra. Drugim riječima, CAPEX troškovi su visoki, a oprema se u potpunosti amortizira kroz mali broj godina obzirom na današnju tehnologiju i brzi razvoj [11][12][13].

Prije više od deset godina ovaj problem je rješavan virtualizacijom računalnih resursa gdje su se resursi jednog fizičkog uređaja, odnosno poslužitelja dijelili na više manjih virtualnih okruženja, tj. logičkih ili aplikacijskih procesa [14]. Danas je virtualizacija prisutna u skoro svim podatkovnim centrima i kao takva više ne predstavlja inovaciju već nužni standard kako bi mogli biti konkurentni na tržištu.

Zbog toga se postavlja pitanje je li moguće postojeće računalne resurse još učinkovitije iskoristiti? Ako je to moguće, smanjili bi se postojeći troškovi, ali i postigla veća iskoristivost budućih resursa, što je ujedno i jedan od ciljeva ovog istraživanja. Drugim riječima ako na učinkovit način djeluje na varijablu ili varijable, u ovom slučaju na računalne resurse, moguće su višestruke uštede. Kako bi problem nepravilnog korištenja računalnih resursa mogli bolje razumjeti napravljen je dijagram uzročno-posljedičnih veza (Slika 1.2.) koji omogućava lakše prepoznavanje potencijalnih problema te uzroka zbog kojih nastaju.



Slika 1.2. - Dijagram uzročno-posljedičnih veza sustava

Kao što je vidljivo iz dijagrama na slici 1.2. postoji jako veliki broj uzroka koji mogu uzrokovati nepravilno korištenje računalnih resursa. Najčešći uzročnici su:

- ***Zaposlenici*** - dosta često zaposlenici svojim utjecajem nenamjerno prouzrokuju da resursi budu nedovoljno iskorišteni. Najčešće razlozi su:
  - *neodržavanje sustava* - najčešći uzrok ovome je nedostatak vremena ili nedovoljan broj IT stručnjaka, odnosno s puno opreme (npr. poslužitelja) po jednom zaposleniku.

- *ljudska pogreška* - razlog je najčešće stres potaknut od stane uprave odnosno kupaca da se sustavi i rješenja što prije isporuče. Ovo najčešće rezultira da se zanemari izrada dokumentacije i detaljno planiranje resursa. U ovakvim situacijama sustavi kroz određeno vrijeme najčešće budu potkapacitirani ili je prisutna velika količina resursa koja se nedovoljno koristi.
- *neznanje* - predstavlja veliki problem jer danas postoji veliki broj rješenja za optimizaciju sustava i potrebno je obavljati redovitu obuku IT zaposlenika na način da se zaposlenicima omogući da učestvuju na raznim IT konferencija ili specijalizacijama za određeno područje. Drugi problem je što se najčešće od IT zaposlenika zahtjeva da su istovremeno odgovorni za puno različitih područja kao što su: mreža, sigurnost, virtualizacija i sl. Kako bi se resursi određenog područja što bolje iskoristila potrebno je da su takva područja odvojena, odnosno da imaju svoje stručnjake koji su specijalizirani za to područje što u praksi najčešće nije slučaj.

- ***Sustav za nadzor*** - poznato je da su podatkovni centri vrlo složeni te da je za njihov nadzor potreban veliki broj alata. Takvi sustavi se moraju nadzirati s različitih aspekata, a neki od njih su:
  - *infrastruktura* - ovo se odnosi na: okolinu odnosno prostor (temperatura i vlaga), sustav za neprekidni izvor energije (engl. *Uninterruptible Power Supply, UPS*), klima uređaje, agregat za proizvodnju električne energije i sl.,
  - *mreža* - oprema kao što je: usmjernik (engl. *router*), prespojnik (engl. *switch*), vatroštit (engl. *firewall*) i sl.,
  - *poslužitelji i centralni sustavi za pohranu podataka* i
  - *instalacije i produkti* - primjer predstavljaju: baze podataka, Web-poslužitelji, Web-aplikacije i sl.

  S obzirom na različite skupine problema potrebno je koristiti veći broj alata kako bi se obavljao pravilni nadzor cjelokupne infrastrukture podatkovnog centra. Takvi alati su najčešće složeni i vrlo skupi zbog čega su prisutni dijelovi sustava koji su bez nadzora i njihovi resursi su najčešće nedovoljno iskorišteni. Sustavi koji su bez nadzora mogu prouzrokovati neočekivani prekid u radu sustava što kasnije stvara probleme prilikom prepoznavanja odnosno lociranja uzroka.

- ***Krajnji korisnici*** - mogu namjerno i nenamjerno prouzrokovati probleme. Pod namjernim se podrazumijevaju razni oblici sigurnosnih napada koji mogu

prouzrokovati nedostupnost sustava (engl. *Denial of Service, DoS*). U takvim slučajevima dođe do naglog opterećenja računalnih resursa određenog dijela sustava što rezultira potpuni kolaps cijelog sustava pa i onih dijelova gdje resursi uopće nisu iskorišteni. Nenamjerni korisnici su najčešće kupci koji svojim zahtjevima mogu uzrokovati nagla opterećenja sustava (npr. nagradne igre na Web-stranicama), što u konačnici također može rezultirati nedostupnost sustava. Zbog toga su potrebne prediktivna mjerenja i kontrole koje bi osigurale dovoljno računalnih resurse za ovakve scenarije.

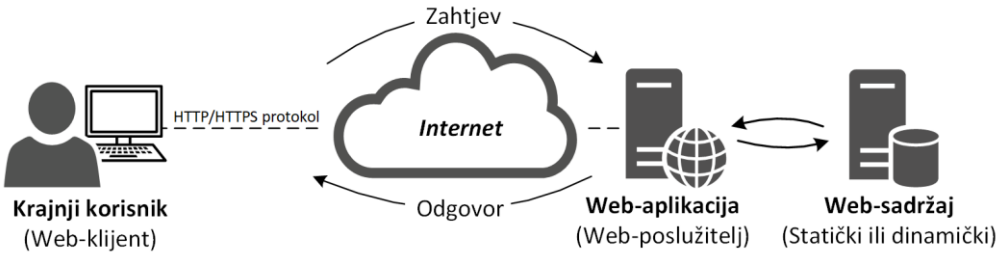
- ***Infrastruktura*** - već je spomenuto da je važno infrastrukturu nadzirati, međutim uvijek su prisutni određeni neočekivani problemi kao što su: nedostupnost električne energije, hardverska greška, problem na mreži i sl.

Kao što je vidljivo postoji veliki broj problema koji uzrokuju da se računalni resursi nedovoljno iskorištavaju. S obzirom da je prisutan veliki broj problema koji su prouzrokovani od strane čovjeka potrebno je načiniti automatizirani sustav koji bi samostalno djelovao i donosio odluke, odnosno obavljao raspodjelu resursa na temelju unaprijed definiranih parametara i ograničenja. Kako bi navedeni problem mogli riješiti potrebno je definirati interesno područje i ograničenja, odnosno vrstu tehnologija i usluga koje će biti predmet istraživanja.

U drugom poglavlju kao predmet istraživanja se detaljnije opisuju Web-farme koje predstavljaju složene i skupe sustave s velikim brojem komponenti od kojih Web-poslužitelji imaju jednu od glavnih uloga. Treće poglavlje predstavlja pregled postojećih rješenja i dosadašnjih istraživanja koji se odnose na temu nedovoljne iskoristivosti postojećih računalnih resursa. Rezultat tog istraživanja je ujedno i jedan od motiva koji je opisan u četvrtom poglavlju. U petom poglavlju su predstavljeni ciljevi, istraživačka pitanja i hipoteze ovog istraživanja, dok je u šestom poglavlju naveden znanstveni i društveni doprinos rada. Sedmo poglavlje predstavlja detaljnu razradu samog istraživanja, odnosno izgradnja modela za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa. Navedeno poglavlje se sastoji od šest cjelina jer je istraživanje rađeno prema istraživačkoj paradigmi znanosti o dizajnu (engl. *Design Science Research Methodology, DSRM*) koje se sastoji od šest procesnih koraka. U posljednjem poglavlju se navode zaključci i smjernice za buduća istraživanja.

2 Predmet istraživanja

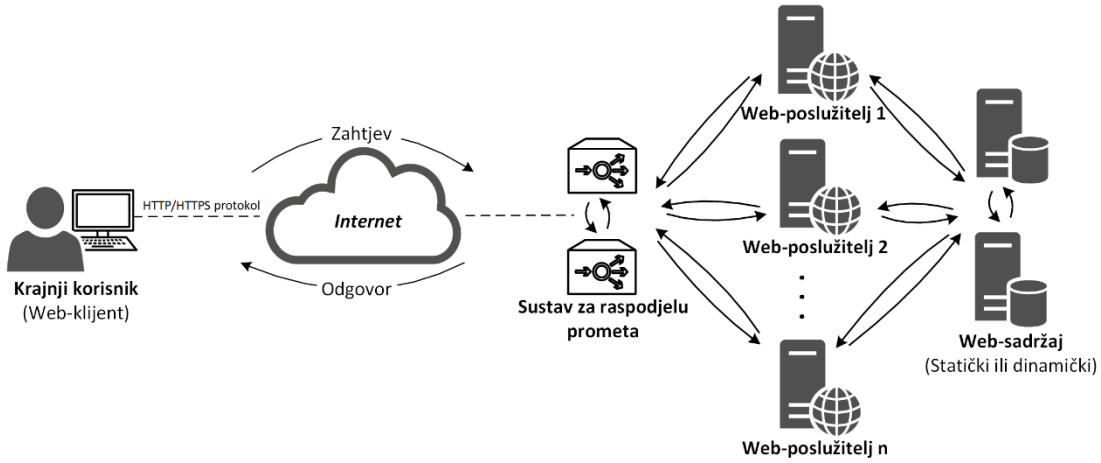
S obzirom da je računarstvo u oblaku vrlo širok pojam, u ovom istraživanju glavni fokus je na uslugama PaaS (Platform-as-a-Service) i SaaS (Software-as-a-Service) koje se zasnivaju na protokolima HTTP/HTTPS [15][16][17]. Web-stranice predstavljaju konkretan primjer ovakvih servisa odnosno usluga. Razvojem tehnologija i sve složenijim korisničkim zahtjevima usluge koje se pružaju pomoću Web-poslužitelja postaju sve složenije, a samim time i njihova arhitektura. Primjer takvog složenog rješenja predstavljaju Web-farme, koje se mogu sastojati od jednog ili više Web-klastera unutar kojih se nalazi više Web-poslužitelja čija arhitektura i dizajn ovise o tehnologiji i vrsti sadržaja (statički ili dinamički). Kako bi bolje razumjeli glavni uzrok potrošnje računalnih resursa moramo krenuti od samog početka, odnosno instrukcija tj. informacija koje je potrebno obraditi u određenom vremenu. Na slici 2.1. prikazan je tijek obrade HTTP/HTTTPS zahtjeva.



Slika 2.1. - Shematski prikaz komunikacije između krajnjeg korisnika i Web-poslužitelja

Krajnji korisnik putem Internet poveznice i domenskog sustava imena (engl. *Domain Name System, DNS*) svojim zahtjevima dolazi do tražene Web-stranice koja se u ovisnosti o svojoj složenosti nalazi na jednom ili više Web-poslužitelja. Za svaku Web-stranicu neophodno je imati jedan od Web-poslužitelja (*apache, nginx*, i sl.) i u ovisnosti o složenosti same stranice odvojenu bazu podataka na kojoj je pohranjen sadržaj stranice.

Današnji sustavi više nisu jednostavni kao na prethodnoj slici s obzirom da su danas Web-stranice (aplikacije) važna karika skoro svakog poslovnog sustava i nije prihvatljivo da budu nedostupne s obzirom na već spomenute razloge. Upravo zbog toga takvi sustavi postaju sve složeniji kako bi se omogućila njihova neprestana dostupnost (engl. *High Availability*). Kako bi se postigla veća dostupnost sustava koristi se veći broj Web-poslužitelja koji zajedno čine Web-klaster. Takva rješenja za razliku od jednostavnih rješenja na ulazu imaju sustave za raspodjelu prometa (engl. *load balancer*), odnosno HTTP/HTTPS zahtjeva što pokazuju mnogobrojni radovi i istraživanja na ovu temu [18][19][20][21]. Web-klaster (jedan ili više njih) zajedno s ostalim komponentama kao što su: baze podatka, alati i servisi za podršku i sl. čine Web-farmu. Takvi sustavi zahtijevaju puno više računalnih resursa od jednostavnih rješenja s prethodne slike. Na slici 2.2. je prikazan primjer komunikacije između krajnjeg korisnika i Web-poslužitelja.



Slika 2.2. - Shematski prikaz komunikacije između krajnjeg korisnika i Web-farme

Kao što je vidljivo iz prethodne slike sustav postaje složeniji odnosno ima veći broj komponenata koje omogućavaju veću dostupnost sustava. Drugim riječima ovakav sustav ima puno manje kritičnih komponenti (engl. s*ingle point of failure*) za razliku od jednostavnih tradicionalnih rješenja gdje niti jedna komponenta nije redundantna, odnosno njenim ispadom sustav u potpunosti prestaje raditi.

Složenost i potrebna količina resursa ovakvog sustava nije jedini problem, već činjenica da jedan Web-klaster najčešće nije dovoljan. Neki od razloga su:

- ***Kompatibilnost*** - postoji jako veliki broj Web-tehnologija i programskih jezika za razvoj Web-stranica. Samim time javlja se potreba za većim brojem Web- klastera jer na jednom klasteru nije moguće imati instalirano paralelno više okruženja, odnosno programa potrebnih za pravilan rad Web-stranica (npr. istovremeno više PHP verzija). Problem se također javlja ako želimo istovremeno imati više Web-sustava za upravljanje sadržajem (engl. *Content Management System, CMS*). Isti problem je također prisutan ako želimo istovremeno imati više različitih vanjskih sučelja primjenskih programa (engl. *Application Program Interface, API*), jer većina njih zahtijeva posebne postavke i izmjene u glavnoj PHP konfiguraciji koja je specifična pa se javlja problem kompatibilnosti s ostalim rješenjima.
- ***Vrsta sadržaja*** - sadržaj se dijeli na statički ili dinamički i ovo predstavlja važnu stavku svakog Web-klastera ako se žele postići što bolje performanse sa što manje resursa.
- ***Web-poslužitelji*** - za pravilan rad Web-stranica potreban je Web-poslužitelj odnosno aplikacija koja omogućava obradu HTTP/HTTPS zahtjeva kao što su: nginx, apache, IIS, GWS i sl. Ovisno o tehnologiji i vrsti Web-sadržaja koriste se različiti Web-poslužitelji. Na jednom poslužitelju nije moguće (odnosno nije preporučeno) kombinirati više vrsta Web-poslužitelja zbog resursa i

konfiguracije. Stoga se najčešće po Web-klasteru koristi po jedan ili dva Web-poslužitelja ako se žele postići najbolji rezultati.

- ***Vrsti poslovanja*** - većina pružatelja usluga za Web-udomljavanje (engl. *hosting*) dijeli Web-stranice na više različitih Web-klastera ovisno o vrsti njihovog sadržaja odnosno poslovanja (korporativne, financijske, igre i sl.). Postoji više razloga za ovo, a neki su:
  - *sigurnost* - neke Web-stranice imaju puno više sigurnosnih napada za razliku od drugih Web-stranica. Zbog toga se koristi više različitih Web-klastera kako napad na jedan ne bih uzrokovao ispad cijelog sustava.
  - *privatnost* - veliki broj kupaca želi sačuvati svoju privatnost i želi posebne uvjete (posebni IP ili izdvojeni Internet poslužitelj, izdvojene DNS poslužitelje i sl.).
- ***Resursi*** - veliki broj kupaca želi izdvojene resurse odnosno Web-klastere samo za sebe na koje najčešće imaju posebne privilegije i pristupe sustavu kako bi mogli sami obavljati izmjene nad Web-stranicama.
- ***Performanse*** - ako je riječ o većem broju Web-stranica onda se javljaju uska grla (engl. *bottlenecks*) na Web-klasterima gdje računalni resursi najčešće nisu problem već aplikativna razina (npr. broj veza, broj zahtjeva u određenom periodu i sl.).
- ***Dostupnost*** - danas "nije dovoljno" imati samo visoku razinu dostupnosti sustava, već se od pružatelja usluga Web-udomljavanja zahtijevaju posebni uvjeti kao što su: georedundancija (replika cjelokupnog rješenja u drugom podatkovnom centru), certifikacija (veliki broj različitih standarda kao što su: ISO, Tier i sl.), sustav za oporavak od katastrofe (engl. *Disaster Recovery*) i sl.

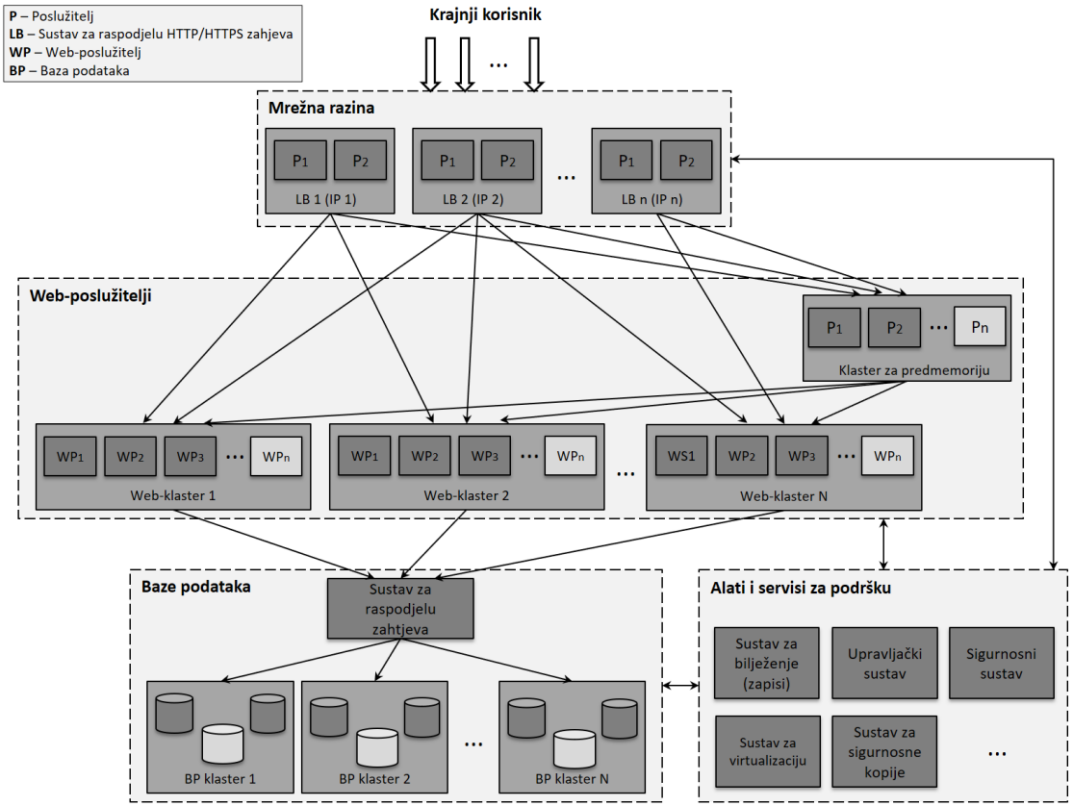
Zbog navedenih razloga, potreban je veći broj Web-klastera kako bi se postigla što veća konkurentnost na tržištu. Kako bi se to postiglo, potreban je veći broj poslužitelja, a samim time još složenija infrastruktura za njihov pravilan rad, odnosno veći prostor, bolje hlađenje, više električne energije i sl. Sve ovo rezultira još većim OPEX i CAPEX troškovima.

Pored svega navedenog, jedan od glavnih razlog zbog kojeg je arhitektura ovih sustava složenija od tradicionalnih rješenja je to što se nastoji pružiti što veća razina dostupnosti sustava sa što većim performansama. Drugim riječima pored Web-poslužitelja kao glavne komponente sustava postoje i mnogobrojne druge potporne komponente kao što su: potporni alati, sustav za priručnu memoriju, klaster za baze podataka i sl. Drugim riječima, takva složena arhitektura se može podijeliti u četiri kategorije odnosno razine, a one su:

- ***Mreža*** - u ovu kategoriju spadaju sustavi za raspodjelu HTTP/HTTPS zahtjeva odnosno prometa. Sustavi su najčešće ostvareni u paru jer se želi postići veća razina dostupnosti. Oni se razlikuju prema:
    - *vrsti algoritma* - prema kojem raspodjeljuju zahtjeve dalje na Web-poslužitelje (čvorove).
    - *kategoriji poslovanja* - najčešće ostvareno uz pomoć više različitih javnih IP-adresa od više različitih pružatelja Internet usluga bilo to izdvojenim vezama ili više njih uz pomoć BGP (Border Gateway Protocol). Ovim se postiže veća razina dostupnosti sustava jer su Web-klasteri podijeljeni na više mreža i samim time nisu centralizirani na jednoj javnoj IP-adresi. Drugim riječima, ako se desi sigurnosni napad na jednu od Web-stranica odnosno jednu Web-farmu druge bi trebale raditi bez prekida.
    - *vrsti tehnologije (rješenja)* - ovdje ima više podjela, a one su: vrsti zahtjeva (HTTP, HTTPS ili oboje), komercijalno ili nekomercijalno rješenje i sl.
- ***Web*** - ova razina se sastoji od dvije komponente a one su:
    - *Web-klasteri* koji su podijeljeni uz pomoć sustava za raspodjelu HTTP/HTTPS zahtjeva i
    - *sustava za priručnu memoriju* koji služi za brži dohvat i obradu podataka. Glavna svrha ovih sustava je da se često korišteni podaci pohrane u sustav tj. u memoriju kako bi se smanjilo vrijeme obrade zahtjeva odnosno dohvaća podataka.
- ***Baze podataka*** - u ovakvim sustavima više nije riječ o jednoj bazi podataka već klasteru poslužitelja koji predstavljaju uvezanu grupu poslužitelja čiji je glavni cilj postizanje što boljih rezultata prilikom obrade podataka i dostupnosti sustava.
- ***Alati i servisi za podršku*** - kako bi jedan ovakav složeni sustav mogao pravilno funkcionirati potreban je veći broj alata za njegovu podršku i nadzor, a neki od njih su:
    - *sustav za bilježenje (zapisi)* - kako bi se postigli što bolji rezultati potrebno je pratiti izmjene u sustavu odnosno obavljati zapise. Ovakvi alati omogućuju: kraće vrijeme pronalaska grešaka prilikom ispada sustava, praćenje prometa i izmjena, lakše prepoznavanje "uskih grla" sustava i sl. Dosta često su ovakvi sustavi zakonom obvezni. Primjer ovakvih sustava su Web-stranice gdje se prikupljaju podaci o izvršenim naplatama odnosno transakcijama (vrsta pretplate, datum, vrijeme, usluga i sl. Neki od poznatih rješenja otvorenog koda su: Elasticsearch, Graylog, Syslog, ...

- *sustav za sigurnosne kopije* - ovaj sustav je potrebno imati iz više razloga. On omogućava brzi povrat podataka u kritičnim situacijama ili ispadima, bilo da je riječ o dijelu podataka ili cijelom virtualnom poslužitelju. Drugi razlog za ovakvim sustavima je taj što je potrebno produkcijski dio sustava "čistiti" od velikog broja podataka te se obavlja automatizirano pohranjivanje starijih podataka po određenim pravilima i propisima. Bitno je napomenuti da kod nekih vrsta poslovnih sustava koji koriste Web-stranice potrebno zakonski arhivirati i čuvati podatke do nekoliko godina, npr. ako je riječ o audio snimkama, podacima o financijskim transakcijama i sl.
- *upravljački sustav* - kako bi se postigla visoka razina centralizacije i automatizacije potrebno je imati ovakav sustav jer on omogućava izmjene i održavanje sustavom s jednog centralnog mjesta (npr. nadogradnja, pokretanje naredbi, sl.). Neki od poznatih rješenja otvorenog koda su: Puppet, Chef, Docker, …
- *sigurnosni sustav* - za neprekidni rad Web-farmi potrebna je sigurnosna zaštita na više razina (mreža, poslužitelji, aplikacije, …). Ako Web-farme nisu zaštićene nekim izdvojenim rješenjem za sigurnosne napade onda se najčešće koriste kombinacije više programski rješenja, kao što su: AlienVault, WanGuard, NetFlow i slično.
- *sustav za praćenje i nadzor* - ako se želi ostvariti pravovremena informiranost o svim opterećenjima i ispadima unutar sustava onda je potrebno imati ovakav sustav. S obzirom da je IT infrastruktura vrlo složena i sastoji se od više spomenutih razina najčešće se koristi kombinacija više alata za nadzor cjelokupnog sustava. Neki od poznati alata su: Nagios, Munin, Zabbix, Zenoss, Observium, Cacti, New Relic, MONyog i sl.
- *sustav za virtualizaciju* - na početku rada je spomenuto kako je virtualizacija jedan od osnovnih načela višestrukog iskorištenja resursa u ovom slučaju poslužitelja. Kako bi se mogla primijeniti virtualizacija potreban je jedan od sustava za njihov menadžment (engl. *provisioning, orchestrating*). Osim sustava za menadžment važno je znati i koje rješenje tj. platforma će se koristiti (KVM, WMware, Hyper-V i sl.).

Na slici 2.3. prikazana je arhitektura Web-farme sa svim komponentama koje su potrebne za njezin pravilan rad.



Slika 2.3. - Arhitektura Web-farme

Kako bi Web-farma pravilno radila potreban je veliki broj komponenti odnosno poslužitelja. Prethodno je već spomenuto da su poslužitelji najskuplja komponenta jednog podatkovnog centra zbog visokih kapitalnih troškova i visoke stope amortizacije. Razlog zašto su baš Web-poslužitelji i njihovi resursi predmet proučavanja je dugogodišnje iskustvo kao i svakodnevno suočavanje s ovim problemima u praksi. Utvrđeno je da je upravo kod Web-poslužitelja prisutna velika količina računalnih resursa koja se rijetko koristi, ali mora biti dodijeljena kako bi se osigurala dostupnost ovih važnih sustava. Dakle fokus ovog istraživanja nisu Web-farme i njihova arhitektura već nedovoljna iskoristivost postojećih računalnih resursa na primjeru Web-poslužitelja. Navedeni problem je ujedno i razlog zbog kojeg se u protekle dvije godine provelo nekoliko istraživanja na temu poboljšanja računalni resursa Web-poslužitelja [22][23]. U drugom dijelu rada su navedeni konkretni primjeri iz prakse koji potvrđuju nedovoljnu iskoristivost računalnih resursa Web-poslužitelja. Međutim, prvo je odrađena analiza postojećih rješenja i dosadašnjih istraživanja kako bi se utvrdilo da li postoji rješenje za navedeni problem, odnosno je li moguće postojeće resurse još više i bolje iskoristiti.

3 Pregled postojećih rješenja i dosadašnjih istraživanja

Kako bi odgovorili na istraživačka pitanja, izvršena je analiza literature i postojećih rješenja. Tijekom proučavanja znanstvene literature utvrđeno je da postoji veliki broj istraživanja iz navedenog područja koja predlažu mnogobrojne metode, koncepte, ali i pristupe rješenju ovog problema.

Većina istraživanja, koja se bave na temu kako bolje iskoristiti računalne resurse, koriste kombinaciju vlastitih algoritama i mehanizam za kontrolu rezervacije mrežnih resursa (engl. *Quality of Service, QoS*). Takvi mehanizmi omogućuju prioritizaciju resursa ili usluga između različitih aplikacija, korisnika ili tokova podataka. Međutim, navedeni prijedlozi rješenja se bave isključivo resursima procesora, a ne i radnom memorijom koja je bitna stavka računalnih resursa [24][25][26].

Jedan od pristupa je korištenje raznih mrežnih algoritama i arhitektura za raspodjelu prometa po mreži [27][28][29]. Ovakav pristup ne rješava problem u cijelosti, jer su resursi i dalje dodijeljeni određenim sustavima odnosno poslužiteljima i u slučaju porasta zahtjeva, problemu se pristupa na način da se zahtjevi preusmjeravaju na druge sustave umjesto da se postojeći sustavi više, odnosno bolje iskoriste [30].

Većina istraživanja na temu kako višestruko iskoristiti računalne resurse ima sasvim drugi cilj, a to je da se postigne veća dostupnost sustava na način da se obavlja migracija virtualnih poslužitelja s jednog fizičkog poslužitelja na drugi [31]. Ovakav pristup stvara povećanje indirektnih troškova (električna energija, veći broj poslužitelja, održavanje sustava i sl.), jer je glavni cilj da sustav radi, a ne i kako bolje iskoristiti postojeće resurse [32][33][34][35][36].

Sličan pristup predstavljaju kombinacije raznih metoda i algoritama koje omogućuju skaliranje sustava u svrhu osiguravanja dostupnosti istog. Primjer takvog pristupa je istraživanje u kojem se koristi *Dynamic Resource Allocation* algoritam koji omogućava raspodjelu opterećenja na više poslužitelja. Algoritam radi na način da opterećenje prosljeđuje na poslužitelje koji se slabije ili nikako ne koriste [37]. Nedostatak ovog pristupa je to što je za povećanje resursa potrebno imati već spremne poslužitelje koji kad se ne koriste nepotrebno troše resurse kao što su: IP adresa, električna energija, računalni resursi potrebni za operacijski sustav i sl. Drugim riječima, ukoliko se promjene kao što su dodavanje i oduzimanje resursa izvršavaju na već postojećim poslužiteljima onda se znatno manje nepotrebno troše resursi.

Skaliranje sustava je također moguće primjenom različitih mehanizama. Primjer jednog mehanizma je rezervacija računalnih resursa u obliku kontejnera koji se koriste prema potrebi, odnosno kada dođe do opterećenja poslužitelja [38]. Najveći nedostatak ovog pristupa je to što su računalni resursi rezervirani i ne mogu se koristiti u druge svrhe, odnosno mogu ih koristiti samo poslužitelji kojima su namijenjeni. U ovom slučaju problem nedovoljnog iskorištenja resursa se javlja kada ti poslužitelji nisu pod opterećenjem, a njihove resurse ne mogu koristiti drugi poslužitelji koji su možda tada potrebniji.

Resurse je također moguće skalirati pomoću sustava za raspodjelu zahtjeva. Ovaj pristup omogućava da se zahtjevi preusmjere na više odnosno manje poslužitelja prema određenim kriterijima [39]. Nedostatak ovog pristupa je to što su resursi dodijeljeni poslužiteljima koji su

u stanju mirovanja ili su ugašeni. Drugi nedostatak ovog pristupa je to što dodatni resursi nisu odmah dostupni sustavu za raspodjelu zahtjeva jer je potrebno određeno vrijeme da se poslužitelj pokrene. Dodatni resursi (u ovom slučaju poslužitelji) su najčešće potrebni u situacijama kada se dogodi nagli porast zahtjeva kada je ključno da su resursi čim prije dostupni kako ne bi došlo do potpunog zastoja cijelog sustava. U ovom slučaju se postavlja pitanje je li uopće isplativ ovakav način "uštede" resursa, jer on može rezultirati puno veće gubitke ukoliko dođe do zastoja cijelog sustava kao što su: financijski gubitci, ugled, vjerodostojnost i sl.

Računalne resurse je moguće skalirati vertikalno (izmjene resursa na postojećem poslužitelju) i horizontalno (dodavanje ili smanjivanje ukupnog broja poslužitelja). S obzirom da je u ovom istraživanju cilj poboljšati iskoristivost postojećih resursa riječ je o vertikalnom skaliranju. Postoje mnogobrojna istraživanja na temu skaliranja računalni resursa, a jedno od njih daje pregled postojećih rješenja koja se odnose na vertikalno i horizontalno skaliranje računalnih resursa [40]. Autori navode kako ne postoji vertikalno skaliranje resursa koje bi omogućilo smanjivanje računalni resursa bez potrebe za ponovnim pokretanjem poslužitelja. U radu su također navedene smjernice i prijedlozi na koji način riješiti nedostatke postojećih rješenja, ali je izostavljena sama realizacija odnosno implementacija predloženog koncepta.

Mnogo rješenja koja se odnose na skaliranje računalnih resursa ovise o svojoj primjeni tj. moguće ih je primijeniti samo za određene aplikacije. Primjer takvog rješenja HTTP/2 poslužitelj koji ima posebni mehanizam za skaliranje prometa prema više Web-poslužitelja [41]. Jedan od nedostataka ovog rješenja je njegova ograničena primjena (samo za Web-poslužitelje), a drugo je to što radi samo horizontalno skaliranje resursa, dakle prema potrebi koristi veći odnosno manji broj poslužitelja umjesto da na postojeće resurse na poslužiteljima bolje iskoristi. Drugi primjer rešenja za skaliranje računalnih resursa koja ovise o svojoj primjeni su *Hadoop* klasteri koji predstavljaju složena rješenja za obradu i analitiku podataka [42][43]. Oni se baziraju na rješenjima kao što su *OpenStack* i *Amazon Web Services (AWS)* koja će biti detaljnije opisana u nastavku rada. Navedena rješenja imaju mnogobrojne nedostatke jer se baziraju na dodavanju novih instanci poslužitelja koji najčešće imaju predefinirane specifikacije (računalne resurse) sto rezultira da se najčešće doda više resursa nego li je to stvarno potrebno.

Veliki broj rješenja za skaliranje računalnih resursa ima različite mehanizme koji omogućavaju prediktivno skaliranje resursa [44][45][46]. Nedostatak ovih rješenja je to što se odluke donose na temelju podataka koji su prikupljeni od prije i ne uključuju sve nove potencijalne scenarije kao što su izvanredne i neočekivane situacije koje najčešće zahtijevaju veliku količinu resursa. Ovaj pristup je prihvatljiv kod sustava koji uvijek imaju istu potrebu za resursima u određenim vremenskim periodima (npr. sezonske pojave). Danas se očekuje da

sustavi moraju biti spremni na veći broj scenarija, a ne samo na scenarije koji su se već dogodili i koje prediktivni sustavi mogu predvidjeti. Zbog navedenih nedostataka koje imaju postojeća rješenja novi model koji je predložen u ovom istraživanju ima komponentu koja se zove kontrola sustava i unutar nje su definirane dvije metode (*Load Average* i *Increment*) za raspodjelu resursa. Metode su naprednije od postojećih rješenja jer se ne temelje na prethodno prikupljenim podacima i nastoje uvijek poslužiteljima dodijeliti onoliko resursa koliko im je potrebno za njihov pravilan i konzistentan rad. Navedene metode su detaljnije opisane u poglavlju 7.3. koje se odnosi na dizajn i razdvoj novog modela.

Slična rješenja prethodno opisanim mehanizmima za prediktivno skaliranje resursa jesu rješenja koja koriste posebne alate kao što su *Autoscaling Performance Measurement Tool* [47]. Navedena rješenja se koriste u kombinaciji sa sustavom za raspodjelu zahtjeva prema *Amazon Web Services (AWS)* poslužiteljima. Kako *Amazon Web Services (AWS)* koristi već predefinirane poslužitelje dolazi do neučinkovitog iskorištenja postojećih računalnih resursa jer nije moguće precizno dodati onoliko resursa koliko je točno potrebno u tom trenutku već samo prema unaprijed zadanim specifikacijama. Drugi nedostatak ovog rješenja je to što se u trenutcima kada su potrebni dodatni resursi pokreće nova instanca odnosno poslužitelj što zahtjeva još više dodatnih resursa umjesto da se bolje iskoriste postojeći.

Problem neučinkovitog iskorištenja postojećih računalnih resursa se pokušava riješiti na više načina, jedan od njih su sustavi koji imaju tzv. "samosvjesne" (eng. *Self-Aware*) mehanizme za skaliranje resursa [48]. Međutim, takva rješenja su tek u konceptualnoj fazi odnosno u samim začetcima. Nedostatak kod ovog pristupa je njegova primjena u složenim heterogenim sustavima zbog različitih standarda koji danas postoje.

Osim prethodno navedenih rješenja postoje i veliki broj patenata koje su razvile mnogobrojne tvrtke koje skaliranjem nastoje postići optimizaciju računalnih resursa i veću razinu dostupnosti sustava. Primjer takvog rješenja je sustav za isporuku većeg broja paralelnih poslužitelja [49]. Međutim, kao i kod većine drugih rješenja ovo ima nedostatak što povećava broj poslužitelja umjesto da se postojećih resursi još više i bolje iskoriste. Jedan od patenata za skaliranje je razvijen od strane tvrtke za virtualizaciju računalnih resursa koja se zove *VMware* [50]. Glavna svrha navedenog rješenja je da osigura veću razinu dostupnosti sustava ali ne i kako učinkovitije iskoristiti postojeće resurse. Nedostatci *VMware* virtualizacijske platforme su u nastavku opisanu i dijelu koji se odnosi na komercijalnih rješenja. Slično prethodnom rješenju je i sustav koji u svrhu skaliranja koristi poslužitelja koji je uvijek u pripravnosti i koristi se samo kada dođe do opterećenja glavnog poslužitelja [51]. Ovaj pristup također rezultira da se neučinkovito iskorištavaju postojeći računalni resursi jer se navedeni poslužitelj

ne koristi uvijek, a za njegov rad su potrebni resursi što su: kapacitet na disku, mrežni resursi, procesor, memorija i sl.

Suprotno od skaliranja je pristup kojim se virtualni poslužitelji nastoje grupirati na što manji broj fizičkih poslužitelja, kako bi se ostatak (u tom trenutku slobodnih) fizičkih poslužitelja mogao ugasiti u svrhu manje potrošnje električne energije [52]. Ovaj pristup nije dobar, jer poslužitelji iako su ugašeni i dalje zauzimaju skupi prostor u ormarima (engl. *rack space*) koji se nalaze u podatkovnim centrima. Ovaj pristup pored velikih ograničenja koje ima, povlači i druga pitanja kao što su potencijalna nedostupnost cijelog sustava u slučajevima kada dođe do naglog povećanja zahtjeva. U tim slučajevima doći će do nagle potrebe za računalnim resursima koji u tom trenutku nisu dostupni jer je potrebno vrijeme da se fizički poslužitelj uključi u sustav.

Jedan od pristupa rješenju problema je i prioritizacija resursa procesora, tj. postoje mnogobrojni mehanizmi i algoritmi koji omogućavaju da pojedini virtualni poslužitelji dobiju veći prioritet nad resursima procesora fizičkog poslužitelja. Međutim, ovakvi pristupi ne povećavaju i broj jezgri procesora, već samo prioritet izvršavanja procesa [53][54]. Tijekom proučavanja postojećih znanstvenih istraživanja i literature utvrđeno je nekoliko problema, a oni su:

- *Djelomične analize* - većina istraživanja nisu razrađena do kraja, tj. nije prisutan niti jedan način evaluacije predloženih modela i rješenja, odnosno nije izvršeno ispitivanje već su u većini slučajeva samo navedene pretpostavke i koncepti koji bi trebali riješiti navedene probleme.
- *Primjenjivost rješenja* - predložena rješenja najčešće nisu primjenjiva u praksi, već samo u teoriji i razrađena su samo na konceptualnoj razini, a rješenja koja su primjenjiva u praksi nemaju široku uporabu zbog raznih tehnoloških ograničenja.
- *Ponovljivost ispitivanja* - u većini istraživanja okruženje nad kojim su vršena ispitivanja i mjerenja su slabo ili nikako opisana, a testni uzorci su dosta specifični, odnosno nije se moguće nadovezati na postojeće rezultate kako bi se postojeća istraživanja dodatno proširila.
- *Dostupnost rješenja* - ako i postoje određena rješenja i prijedlozi kako riješiti ovaj istraživački problem, ona su najčešće ostvarena kroz izmjenu ili proširenje postojećih modela i alata koji su komercijalni ili nisu svima dostupni što otežava njihovu provjeru i primjenu odnosno daljnja poboljšanja.

Iz područja stručne literature i prakse danas postoji nekoliko rješenja za raspodjelu računalnih resursa. Zajedničko za sva rješenja je to što se temelje na nekoj od virtualizacijskih

platformi (KVM, VMware, Hyper-V, itd.), jer kao što je već prethodno navedeno virtualizacija je preduvjet da bi se resursi mogli dijeliti.

Danas postoje i mnogobrojne platforme koje objedinjuju navedena rješenja i tehnologije, a među njima su najpoznatija *OpenStack* i *Eucalyptus*. One predstavljaju besplatne platforme otvorenog koda koje omogućavaju lakšu administraciju i raspodjelu računalnih resursa u računarstvu u oblaku [55][56][57]. Navedene platforme kao takve nisu neovisno rješenje, već predstavljaju skup mnogobrojnih tehnologija koje se nadovezuju na postojeća rješenja uključujući i navedene virtualizacijske platforme. Nedostatci i ograničenja za raspodjelu računalnih resursa koji su prisutni na samim virtualizacijskim platformama se prenose na višu razinu, odnosno na rješenja kao što su *OpenStack* i *Eucalyptus*. U nastavku rada navedeni su primjeri takvih rješenja.

- **Komercijala rješenja:**
    - *VMware* - predstavlja jednu od najpoznatijih virtualizacijskih platformi koja ima nekoliko načina raspodjele računalnih resursa. Najčešći način je automatsko pokretanje novih instanci poslužitelja na način da se prate resursi postojećih poslužitelja. Ovakvo rješenje zavisi o zalihama računalnih resursa, koji se ne koriste, ali se pritom moraju rezervirati za ovakve scenarije, ako se želi postići visoka razina dostupnosti sustava. Drugim riječima, u trenutcima kada sustav nije pod opterećenjem ti resursi nisu iskorišteni, što rezultira neučinkovitu iskoristivost ukupnih resursa [58][59]. Ova virtualizacijska platforma također omogućuje dodavanje resursa na postojeće poslužitelje, ali uz dva preduvjeta: da je to definirano prije samog pokretanja poslužitelja i da to omogućava operacijski sustav. Radnu memoriju je moguće dodijeliti i oduzeti na način da se definiraju gornje i donje granice resursa. U ovom slučaju se radna memorija oduzima odnosno dodaje u ovisnosti o opterećenju sustava. Što se tiče CPU, moguće je samo dodavanje resursa i to samo za određene operacijske sustave, dok oduzimanje resursa procesora nije moguće, jer kao takvo nije razvijeno kao mogućnost VMware platforme. Razlog zbog kojeg ovo još nije razvijeno je to što gotovi svi operacijski sustavi ne podržavaju mogućnost oduzimanja broja jezgri nad sustavom koji je u radnom stanju [60][61][62]. Drugim riječima, ponovo je prisutna neučinkovita iskoristivost sustava ako opterećenje sustava prođe, jer jednom kad su resursi procesora dodijeljeni (ako je to uopće moguće za taj operacijski sustav) njih više nije moguće smanjiti bez da se uradi ponovno pokretanje poslužitelja.

- *Hyper-V* - predstavlja Microsoftovu virtualizacijsku platformu, koja radi na sličnom principu kao i VMware, odnosno u trenutcima opterećenja sustava stvara nove instance poslužitelja i to sve dok ima zaliha tj. slobodnih računalnih resursa [63][64]. Također je moguće definirati gornje i donje granice unutar kojih je moguće dodavanje odnosno oduzimanje radne memorije. Manipulacija resursa procesora je riješena na način da se poslužiteljima dodjeljuje prioritizacija nad resursima procesora fizičkog poslužitelja (npr. ako su prisutna dva poslužitelja i prvi zahtjeva 400% resursa procesora a drugi 100%, to znači da će se prvo izvršavati četiri zahtjeva od prvog poslužitelja, a tek onda od drugog) [65][66][67]. Ovo rješenje nije dovoljno učinkovito, jer se sve svodi na prioritizaciju broja dretvi (jezgri) kod poslužitelja, drugim riječima nije moguće dodati ili oduzeti CPU resurse (povećati ili smanjiti kapacitet), već samo djelovati na njegovu prioritizaciju prilikom izvršavanja.
- *Ostala rješenja* - pored navedenih virtualizacijskih platformi, postoje mnogobrojna rješenja kao što su računarstvo u oblaku, koje djelomično rješavaju ovaj problem. Najpoznatiji su:
  - *Amazon Web Services (AWS)* - predstavlja jedno od Amazonovih rješenja za računarstvo u oblaku, gdje se optimizacija računalnih resursa obavlja na način da se automatski ili ručno pokreću i gase nove instance poslužitelja ovisno o opterećenju sustava, što je potrebno unaprijed definirati [68][69][70]. Kod ovakvog rješenja prisutna su dva nedostatka. Prvi nedostatak je to što su profili poslužitelja unaprijed definirani i nije moguće imati linearno povećanje ili smanjenje resursa već isključivo prema unaprijed definiranim specifikacijama. Drugim riječima, sustav nije fleksibilan, što zahtjeva da korisnici često koriste profile poslužitelja, odnosno poslužitelje s predefiniranim specifikacijama. Takav pristup najčešće rezultira da se koristi više računalnih resursa iako za to nema potrebe, jer ne postoje profili koji su optimalni za njihove potrebe. Takav pristup u konačnici rezultira neučinkovitu iskoristivost resursa. Skaliranje sustava predstavlja drugi nedostatak zbog kojeg ovo rješenje ima neučinkovitu iskoristivost računalnih resursa [71][72]. U tim situacijama dolazi do uključivanja i isključivanje novih instanci poslužitelja što rezultira sljedećim nedostacima:

- Izdvojeni resursi za svaki novi poslužitelj - svaki operacijski sustav da bi mogao funkcionirati zahtjeva određene računalne resurse (CPU, memorije, disk, itd.) za sebe, umjesto da se ti resursi dodijele poslužitelju kao takvom.
- Nepotrebno trošenje ostalih resursa - svaka nova instanca poslužitelja da bi bila funkcionalna zahtjeva dodatne resurse kao što su: IP-adrese, licence za operacijski sustav, dodatno troši IOPS (engl. *Input/Output Operations Per Second*) diska, što rezultira većim CPU opterećenjem fizičkog poslužitelja, jer instrukcije duže čekaju na izvršenje, itd.
- Veći troškovi održavanja - veći broj poslužitelja zahtjeva veći nadzor i kontrolu od strane stručnog osoblja, ali i dodatno opterećenje za alate za podršku koji su pritom najčešće ograničeni licencama (ovisno o broju poslužitelja), itd.
- Složenost sustava - veći broj poslužitelja povećava i složenost sustava, što u konačnici može rezultirati dužim vremenom otklanjanja kvara.
- Itd.

- *DigitalOcean* - predstavlja jednu od poznatijih tvrtki koja se bavi računarstvom u oblaku, drugim riječima predstavljaju alternativu za Amazon. Međutim, njihovo rješenje za razliku od Amazonovog ima još veća ograničenja jer ne omogućava automatsko skaliranje resursa podizanjem ili spuštanjem novih poslužitelja, već samo izmjenu postojećih na način da je potrebno ugasiti poslužitelja kako bi se resursi mogli izmijeniti [73].
- *Azure* - predstavlja Microsoftovo rješenja za računarstvo u oblaku, gdje krajnji korisnik sam bira poslužitelje po već unaprijed zadanim profilima. Ograničenja i nedostatci su slična kao kod *Amazon Web Services* i *Digital Oceana* [74][75].
- *IBM PowerVM* - predstavlja IBM-ovu virtualizacijsku platformu koja omogućava raspodjelu računalnih resursa. Rješenje se temelji na dinamičkom logičkom particioniranju (engl. *Dynamic Logical Partitioning, DLPAR*) resursa kao što su CPU i memorija bez potrebe za ponovnim pokretanjem operacijskog sustava [76][77]. Ovo se odnosi na mali broj operacijskih sustava koji ovo podržavaju (AIX,

i5/OS i sl.). Glavni nedostatak ovog rješenja je njegova jako ograničena primjena, odnosno može se primijeniti samo na IBM mikroprocesorima (POWER4 i novije generacije) [78][79].

- *Xen VMM* - predstavlja virtualizacijsku platformu koja koristi *Credit Scheduler*. On predstavlja mehanizam za prioritizaciju resursa procesora, ali ne i za njihovo povećanje i smanjenje samog broja procesora [80].

- **Nekomercijalna rješenja:**
  - *Docker* - za razliku od prethodnih rješenja, ovo je besplatno i otvorenog je koda. Docker predstavlja sustav koji jamči resurse poslužiteljima na način da obavlja rezervaciju računalnih resursa skalabilno po svim poslužiteljima u obliku kontejnera, koje kasnije koristi po potrebi [81][82]. Dakle, prisutno je naprednije korištenje postojećih računalnih resursa kako bi se postigla visoka razina dostupnosti sustava [83][84]. Međutim, resursi se i dalje ne iskorištavaju učinkovito, jer su u ovom slučaju dodijeljeni i rezervirani nekom poslužitelju i ne mogu se prenositi na druge poslužitelje kada se ne koriste.
  - *Ostala rješenja* - tijekom istraživanja ostalih nekomercijalnih rješenja, utvrđeno je da postoje mnogobrojni pristupi u rješavanju ovog problema na način da se kombiniraju razne skripte i programi. U usporedbi s komercijalnim rješenjima ova su još više ograničena što rezultira još slabijom iskoristivosti računalnih resursa [85].

Tablica 3.1. predstavlja usporedbu osobina postojećih komercijalnih i nekomercijalnih rješenja s novim predloženim rješenjem koje je nazvano VM Manager. Rješenje koje je prema mogućnostima izmjene računalnih resursa (CPU i memorije) "najbliže" novom predloženom rješenju je IBM PowerVM. Međutim, za razliku od novog modela koji se predlaže u ovom radu IBM-ovo rješenje je primjenjivo samo za određeni dio IBM hardvera (mikroprocesora) i samo na određenim operacijskim sustavima (AIX, i5/OS i sl.). Navedena ograničenja su bila motivacija da se razvije novi model koji nebi ovisio o hardverskom ili softverskom okruženju.

Tablica 3.1. - Usporedba osobina postojećih komercijalnih i nekomercijalnih rješenja s novim predloženim rješenjem

naziv rješenja	**vrsta licence**	**izmjena resursa bez potrebe za ponovnim pokretanjem sustava**			
		povećanje broja jezgri	**smanjivanje broja jezgri**	**povećavanje memorije**	**smanjivanje memorije**
VMware	zatvoreni kod (vlasnička komercijalna licenca)	da	ne	da	da
Hyper-V	zatvoreni kod (vlasnička komercijalna licenca)	ne	ne	da	da

Amazon Web Services (AWS)	zatvoreni kod (vlasnička komercijalna licenca)	ne	ne	ne	ne
DigitalOcean	zatvoreni kod (vlasnička komercijalna licenca)	ne	ne	ne	ne
Azure	zatvoreni kod za platformu ali otvoren za klijente (SDK)	ne	ne	ne	ne
IBM PowerVM	zatvoreni kod (vlasnička komercijalna licenca)	da	da	da	da
Xen VMM	GNU GPLv2 +	ne	ne	ne	ne
Docker	Freemium, Apache License 2.0	ne	ne	ne	ne
VM Manager	otvoreni kod (GNU GPL v3)	da	da	da	da

Navedena rješenja, bilo ona komercijalna ili nekomercijalna, ne rješavaju istraživački problem u potpunosti. Razlog tome je što navedena rješenja ne omogućavaju višestruku iskoristivost postojećih resursa koji su već dodijeljeni nekom poslužitelju već najčešće dodjeljuju nove resurse koji su rezervirani za ovakve izvanredne situacije [86][87][88]. Resursi se dodjeljuju na način da se postojeći resursi često previše povećaju ili se stvaraju dodatne replike poslužitelja koji su pod opterećenjem. Jedan od glavnih razloga zbog čega ne postoji rješenje koje bi višestruko iskoristilo postojeće računalne resurse je to što većina virtualizacijskih platformi predstavlja komercijalno rješenje čijim je proizvođačima (VMware, Hyper-V, sl.) u interesu da se koristi što veći broj poslužitelja odnosno više programski licenci koje se najčešće naplaćuju po broju poslužitelja. Kod nekomercijalnih rješenja nema licenci, ali su rješenja nedovoljno učinkovita ili ne rješavaju problem u potpunosti.

Analizom postojećih rješenja i njihovih ograničenja utvrđeno je da ne postoji rješenje koje bi omogućilo učinkovitije iskorištenje postojećih računalnih resursa (CPU i memorija). Kako bi se riješio navedeni problem prvo je izgrađen novi model koji je primjenjiv na više sustava, odnosno njegovo ostvarivanje ne ovisiti o tehnologiji i programskom jeziku. Novi model omogućava automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa. Validacija modela je izvršena pomoću aplikacije koja je razvijena na temelju modela koja je primijenjena na Web-poslužiteljima gdje su računalni resursi najčešće nedovoljno iskorišteni, a neophodni kako bi se postigla visoka razina dostupnosti sustava.

Prilikom izgradnje novog modela problemu je pristupljeno formalno odnosno metodološki. U nastavku istraživanja su opisane metode uz pomoć kojih je izgrađen novi model, ali prije toga navedeni su glavni motivi ovog istraživanja.

4 Svrha i motivacija za istraživanje

Danas se informacijska tehnologija i načini poslovanja izuzetno brzo mijenjaju i samo najuspješniji mogu opstati na tržištu. Kako bi se to postiglo potrebna su stalna ulaganja u rast i unapređenje sustava. Već je spomenuto da takva ulaganja rezultiraju visoke kapitalne i operativne troškove. Zbog toga se uvijek nastoji što više i bolje iskoristiti postojeća infrastruktura odnosno teži se ka postizanjem boljih rezultata sa što manje resursa.

Utvrđeno je da postoje sustavi čiji se resursi ne koriste dovoljno, a kao takvi moraju biti osigurani zbog važnosti sustava. Svakodnevno suočavanje s ovim problemima u praksi, ali i dugogodišnje iskustvo iz ovog područja bili su glavna motivacija da se pronađe učinkovitije rješenje za navedeni problem.

Istraživanje je započeto proučavanjem dosadašnjih znanstvenih istraživanja, ali i rješenja iz prakse. Nakon detaljnog proučavanja literature, ali i postojećih rješenja od kojih su neki i ostvareni i evaluirani, utvrđeno je da ne postoji dovoljno učinkovito rješenje koje omogućava bolju iskoristivost postojećih računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja.

Kako bi se to postiglo potrebno je imati mehanizme koji bi omogućili raspodjelu postojećih resursa na način da se obavi dodjela i oduzimanje resursa uz zadržavanje konzistentnosti sustava koja predstavlja neprekidan i ispravan rad sustava [89].

Takav pristup bi omogućio višestruko iskorištenje postojećih resursa što bi indirektno utjecalo na smanjenje troškova. Kapitalni troškovi bi bili manji, jer bi veće iskorištenje postojećih računalnih resursa značilo da je moguće postići bolje rezultate s manjim brojem poslužitelja. Što se tiče operativnih troškova, manji broj poslužitelja rezultira manje troškove kao što su: održavanje, električna energija, hlađenje prostora, zamjena neispravnih dijelova i sl.

Prethodno je spomenuto da je za dijeljenje resursa jednog fizičkog poslužitelja potrebna virtualizacija koja omogućava veću iskoristivost računalnih resursa. Postojeća rješenja koja se temelje na virtualizacijskim platformama nemaju mogućnost napredne dodjele i oduzimanja postojećih računalnih resursa koja je ključna u izvanrednim situacijama kako bi se postigla veća dostupnost sustava. Navedeni razlozi i nepostojanost učinkovitog rješenja za ovaj problem bila su glavna motivacija ovog istraživanja.

5 Ciljevi, istraživačka pitanja i hipoteze

Nakon što je opisana problematika i motivacija ovog istraživanja definirani su ciljevi, a oni su:

C1. Izgraditi novi model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa na primjeru Web-poslužitelja bez potrebe za ponovnim pokretanjem poslužitelja uz zadržavanje konzistentnosti rada.

C2. Povećati iskoristivost postojećih računalnih resursa.

C3. Istražiti i pronaći slučajeve i ograničenja kod kojih predloženi model pokazuje bolje rezultate s obzirom na postojeća rješenja.

Sukladno ovim ciljevima postavljaju se sljedeća istraživačka pitanja, a ona su:

- Kako napraviti automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa na primjeru Web-poslužitelja bez potrebe za ponovnim pokretanjem poslužitelja uz zadržavanje konzistentnosti rada?
- Na koji način pomoću predloženog modela povećati iskoristivost postojećih računalnih resursa?
- U kojim slučajevima i uz koja ograničenja predloženi model radi bolje od postojećih rješenja?

Iz istraživačkih pitanja proizlaze sljedeće hipoteze:

H1. Primjena predloženog modela nad virtualiziranim sustavima omogućava dodavanje/oduzimanje postojećih računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja.

H2. Primjena predloženog modela ne narušava konzistentan rad poslužitelja.

H3. Primjenom predloženog modela nad virtualiziranim sustavima postiže se bolja iskoristivost postojećih računalnih resursa (CPU i memorija) s obzirom na prvobitno stanje.

6 Znanstveni i društveni doprinos rada

U ovom poglavlju su navedeni doprinosi ovog istraživanja. U pogledu znanstvenog doprinosa, ovo istraživanje rezultira s:

- Novim modelom za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa, posebice procesora i radne memorije, bez potrebe za ponovnim pokretanjem poslužitelja.
- Postupak vrednovanja modela simulacijom rada stvarnih sustava.
- *Učinkovitije iskorištenje postojećih resursa* - model omogućava veću iskoristivost postojećih računalnih resursa jer je moguće dodavati i oduzimati resurse (CPU i memorija) bez da je potrebno ponovno pokretanje poslužitelja. Drugim riječima, sustav uvijek troši onoliko resursa koliko mu je potrebno za pravilan rad.
- *Veća dostupnost sustava* - sustav ima brži odziv, odnosno omogućava brže povećavanje resursa ako dođe do opterećenja sustava. Razlog tome je što nije potrebno ponovno pokretanje poslužitelja kako bi se izmijenili resursi već se sve odvija dok sustav radi.
- *Napredna dodjela resursa* - kod postojećih rješenja resursi se dodjeljuju linearno do maksimalne dozvoljene granice. Novi model može na temelju brzine zauzeća slobodnih resursa procijeniti koju količinu resursa je potrebno dodijeliti kako bi sustav radio optimalno. Ovaj način dodjele resursa omogućuje da poslužitelj lakše podnese

izvanredne situacije u kojima dođe do naglog povećanje zahtjeva odnosno potrebe za resursima.

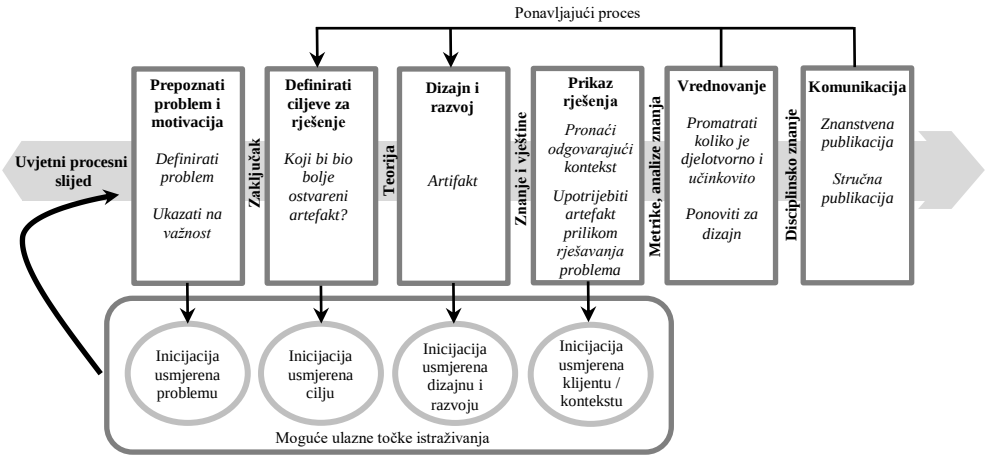
U svrhu društvenog doprinosa cijelo rješenje je otvorenog koda što je ujedno i jedan od glavnih ciljeva ovog istraživanja. Cjelokupno testno okruženje nad kojim je provedeno istraživanje je detaljno opisano kako bi se primjena i provjera modela mogla što jednostavnije ponoviti u svrhu daljnji unapređenja i istraživanja na ovu temu. Za razliku od postojećih rješenja novi model koji se predlaže u ovom istraživanju ima sljedeći društveni doprinos:

- *Rješenje je otvorenog koda* - danas postoje mnogobrojna komercijalna rješenja koja ne samo da ne rješavaju problem u potpunosti već su jako skupa i vrlo zahtjevna tj. potrebno je puno računalnih resursa za njihov pravilni rad. Novo rješenje je otvorenog koda, što omogućuje veću primjenu i daljnja unapređenja od strane ICT tvrtki i znanstvenih zajednica. Aplikacija je izdana pod GNU GPL v3 licencom što znači da je otvorenog koda i njezin izvorni kôd je dostupan je na sljedećoj Web-adresi: https://github.com/dialagic/vmmanager/wiki/VMmanager
- *Veća financijska isplativost* - višestruko iskorištenje postojećih resursa omogućava manji broj potrebnih poslužitelja kako virtualnih tako i fizičkih. Manji broj poslužitelja rezultira: jednostavniju infrastrukturu, manje troškove održavanja i zamjene hardvera, manji mjesečni troškovi režija podatkovnog centra (struja i klimatizacija) i sl. Sve navedene stavke u konačnici rezultiraju manji OPEX.
- *Jednostavnije rješenje* - novo rješenje se odnosi isključivo na problem nedovoljnog iskorištenja postojećih računalnih resursa za razliku od postojećih rješenja koja su najčešće skup rješenja koja su vrlo složena i nisu specijalizirana za navedeni problem. Za njihovu pravilnu uporabu i konfiguraciju najčešće su potrebni eksperti s višegodišnjim iskustvom iz navedenog područja ili je potrebno osigurati dodatno obrazovanje i certificiranje osoblja. Takvi složeni sustavi i platforme najčešće dodatno troše resurse sustava kako bi normalno funkcioniralo za razliku od novog rješenja koje nema ograničenja kao što su minimalni potrebni računalni resursi za rad.
- *Očuvanje okoliša* - iako je ova stavka na posljednjem mjestu nije manje vrijedna. Novim rješenjem moguće je postići bolje rezultate s manjim brojem poslužitelja što indirektno djeluje na očuvanje okoliša jer je manja potrošnja električne energije koja najčešće nije iz obnovljivih resursa.

Kao što je vidljivo iz priloženog, novi model bi trebao riješiti mnogobrojne nedostatke postojećih rješenja jer omogućava automatsku i poboljšanu raspodjelu postojećih računalnih resursa.

7 Model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa

Kao što je već u uvodu spomenuto u ovom istraživanju se primijenjena istraživačka paradigma znanost o dizajnu. Ona se koristi u informacijskim tehnologijama i nudi specifične smjernice za procjenu i iteraciju unutar istraživačkih projekata. Na slici 7.1. predstavljen je procesni model istraživačke paradigme znanosti o dizajnu [90].



Slika 7.1. - Procesni model istraživačke paradigme znanosti o dizajnu [90]

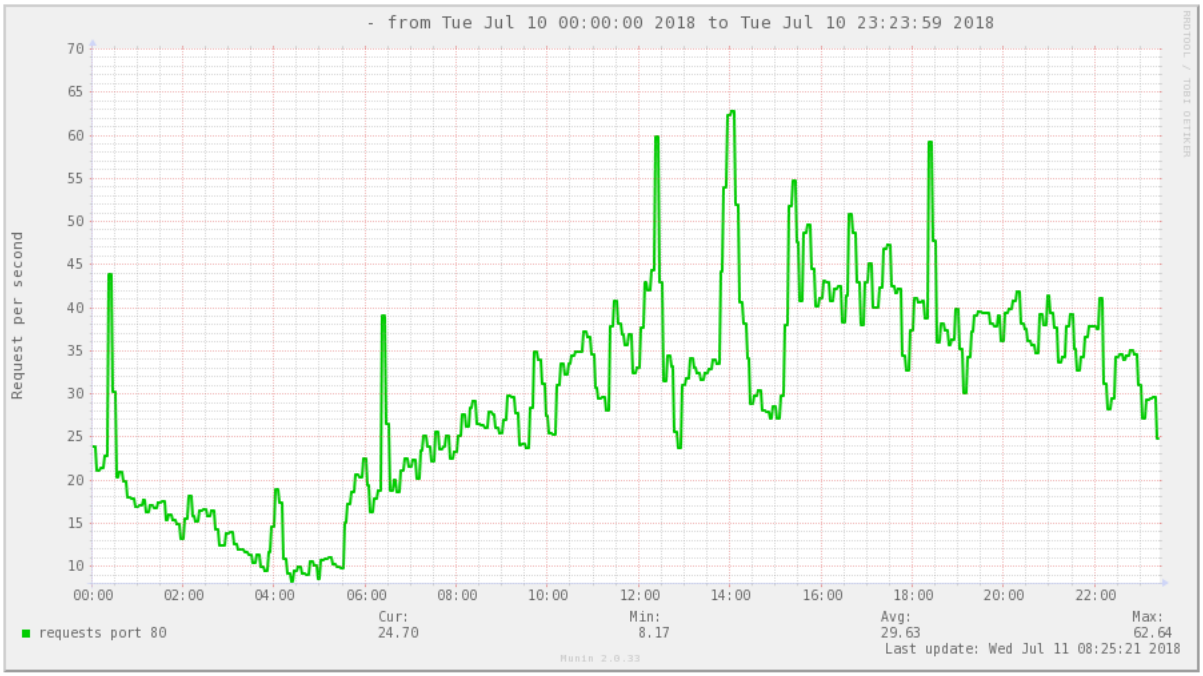
Metodologija se sastoji od šest slijednih procesnih koraka, a oni su: prepoznavanje problema i motivacija, definiranje ciljeva, dizajn i razvoj, prikaz rješenja, vrednovanje i komunikacija [90]. Cijeli proces je uzrokovan jednim od četiri razloga: problem, cilj, dizajn i razvoj ili klijent. U ovom istraživanju glavni inicijator je problem, odnosno nedovoljna iskoristivost računalnih resursa, a artefakt kojim se taj problem želi riješiti predstavlja novi model, koji bi omogućio automatiziranu i poboljšanu iskoristivost postojećih računalnih

resursa. U nastavku rada su predstavljeni svi koraci procesnog modela istraživačke paradigme znanosti o dizajnu, koji detaljnije opisuju ovo istraživanje.

Inicijacija usmjerena problemu

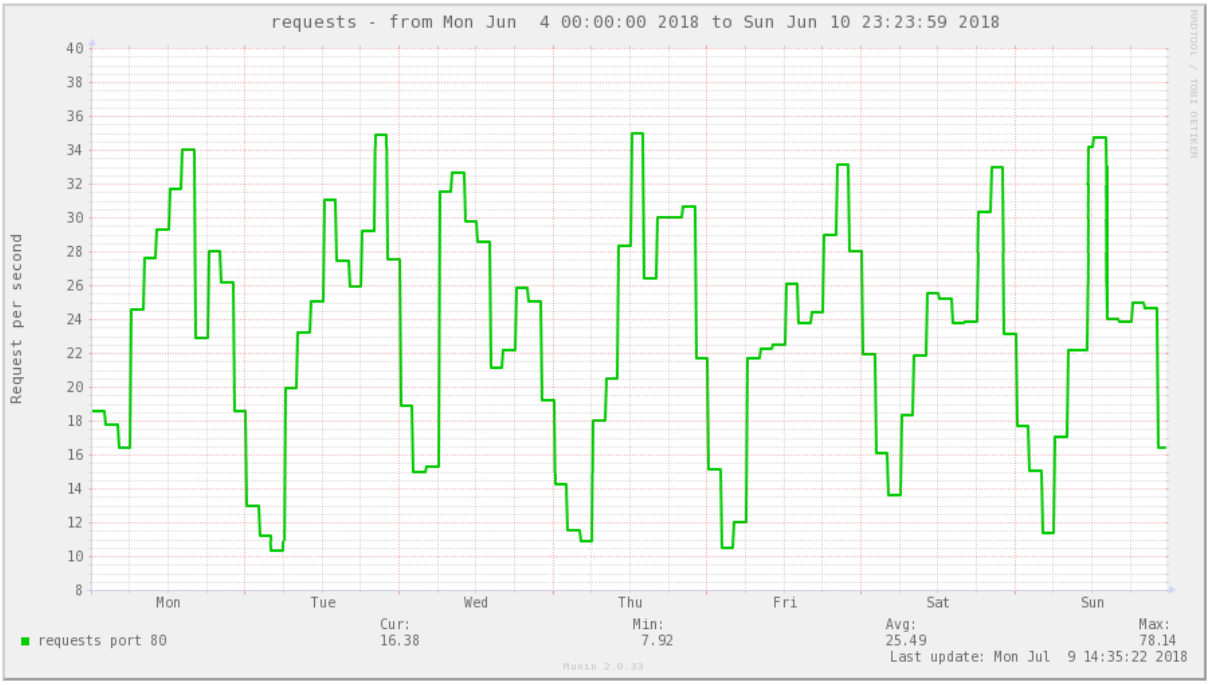
Kao što je već spomenuto postoje mnogobrojne analize i istraživanja koja potvrđuju da su poslužitelji najskuplja komponenta računarstva u oblaku [4][6][7][8]. Razlog tome su veliki kapitalni i operativni troškovi poslužitelja, ali i visoka razina amortizacije. Upravo se zbog toga računalni resursi žele što učinkovitije iskoristiti. Međutim, postoje konkretni primjeri iz prakse (Web-poslužitelji), koji potvrđuju da postoje sustavi čiji resursi (CPU i memorija) nisu dovoljno iskorišteni, ali ih moraju imati zbog zahtjeva za visokom razinom dostupnosti sustava. Kako bi se problematika ovog istraživanja potvrdila izvršeno je preliminarno istraživanje u kojem su korišteni podaci nekoliko IT poduzeća koja pružaju usluge Web-udomljavanja. Tvrtke su osigurale pristup svim relevantnim podacima i pokazateljima koji se odnose na Web-farmu, počevši od infrastrukture pa do broja posjeta, odnosno korisničkih zahtjeva u određenom vremenu prema pojedinim Web-stranicama. Broj korisničkih zahtjeva predstavlja važan parametar stvarnih sustava jer njihovom obradom dolazi do opterećenja računalnih resursa koji su predmet ovog istraživanja. Korisničke zahtjeve je potrebno sagledati kroz različite vremenske periode kako bi utvrdili reprezentativni uzorak koji je potreban za validaciju novog modela. Vrijednosti iz stvarnih sustava su prikupljene pomoću alata za nadzor sustava (*engl.* monitoring tools) koji se zove *Munin*.

Važno je napomenuti da skoro svi alati za nadzor sustava prikupljaju informacije svakih pet minuta i uzimaju u obzir vrijednost koju su u tom trenutku dohvatili od poslužitelja. Drugim riječima, unutar pet minuta moguće je da se dese još i veće odnosno manje promjene u vrijednosti, ali da one ne budu zabilježene od strane alata za nadzor sustava jer u tom trenutku nije vršila provjera poslužitelja. Vrijednost koja određuje koliko često će se vršiti provjera poslužitelja od strane alata za nadzor sustava je najčešće unaprijed definirana i nije promjenjiva. Razlog zbog kojeg je ovaj parametar najčešće nepromjenjiv i nije vremenski kraći je to što je glavna svrha ovih alata praćenje opterećenja sustava kroz neki period u svrhu promatranja i analiza. Alati kod kojih je ovaj parametar najčešće izražen u sekundama su alati koji služe za alarmiranje, odnosno slanje obavijesti u slučajevima kada neki sustav ne radi dobro ili da je nedostupan. U ovim slučajevima provjere se vrše češće kako bi se obavijest dobila čim prije. Slika 7.2. predstavlja broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog dana (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin*.



Slika 7.2. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog dana (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin*

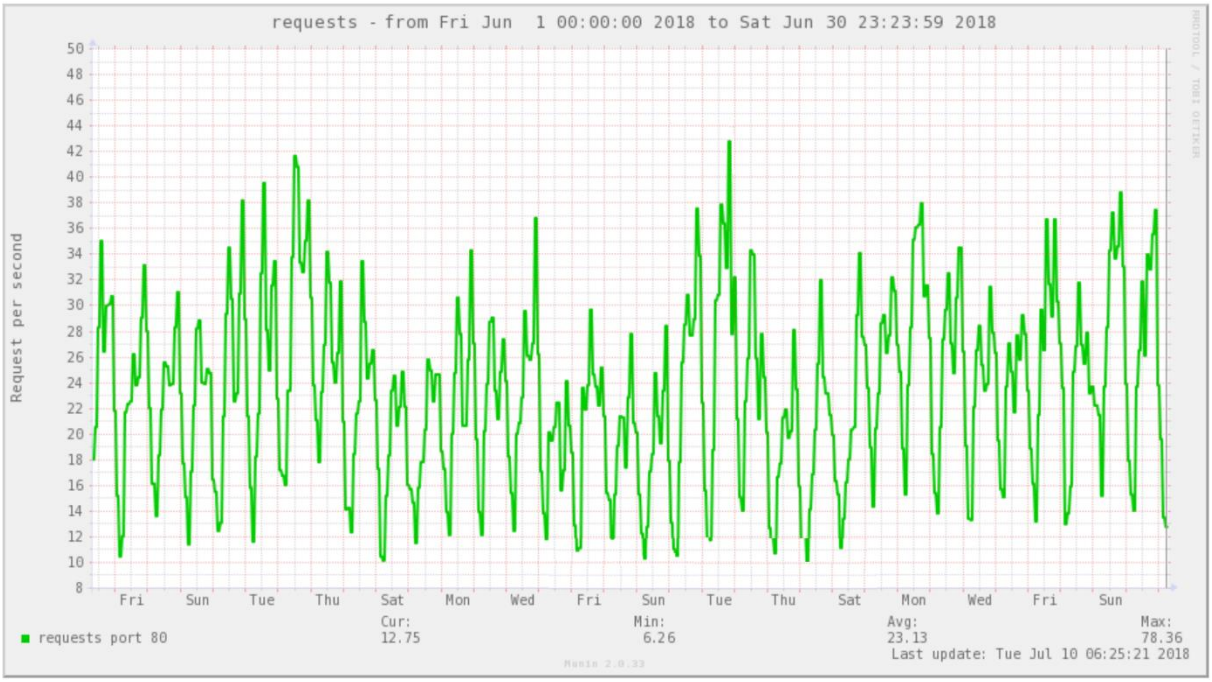
Iz prethodne slike se vidi da je prisutna razlika u broju korisničkih zahtjeva u sekundi tijekom 24 sata, odnosno tijekom dana njihov broj je znatno veći zbog čega je potrebno provjeriti i duži vremenski period kako bi se provjerilo da li su prisutna još veća odstupanja. Slika 7.3. predstavlja broj korisničkih zahtjeva u sekundi u vremenskom periodu od jednog tjedna za Web-farmu iz stvarnog sustava u alatu *Munin*.



Slika 7.3. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog tjedna (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin*

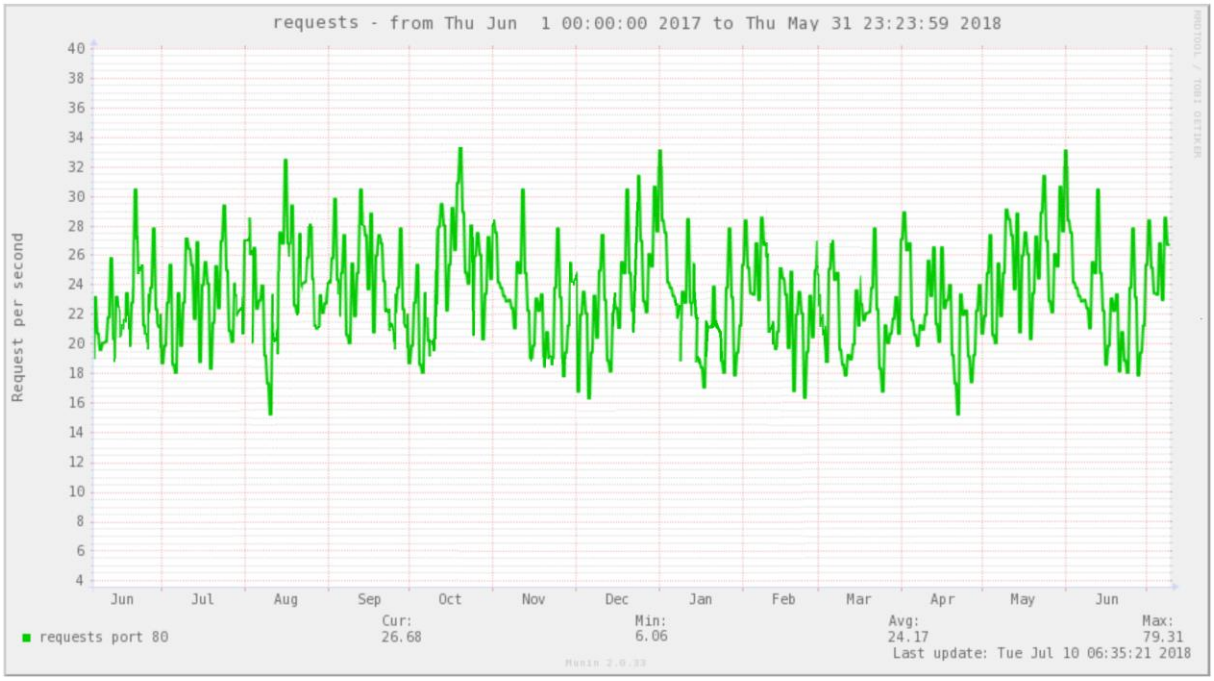
Iz prethodne slike je vidljivo da je i dalje prisutno veće odstupanje u broju korisničkih zahtjeva u sekundi između dnevnih i noćnih sati. Međutim, gledajući cijele dane odstupanja su

manja. Slika 7.4. predstavlja broj korisničkih zahtjeva u sekundi u vremenskom periodu od jednog mjeseca za Web-farmu iz stvarnog sustava u alatu *Munin*.



Slika 7.4. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jednog mjeseca (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin*

Iz prethodne slike se vidi da su i dalje najveća odstupanja u broju korisnički zahtjeva u sekundi tijekom 24 sata. Preostalo je provjeriti broj korisničkih zahtjeva u sekundi u vremenskom periodu od jedne godine (Slika 7.5).



Slika 7.5. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu od jedne godine (apscisa) za Web-farmu iz stvarnog sustava u alatu *Munin*

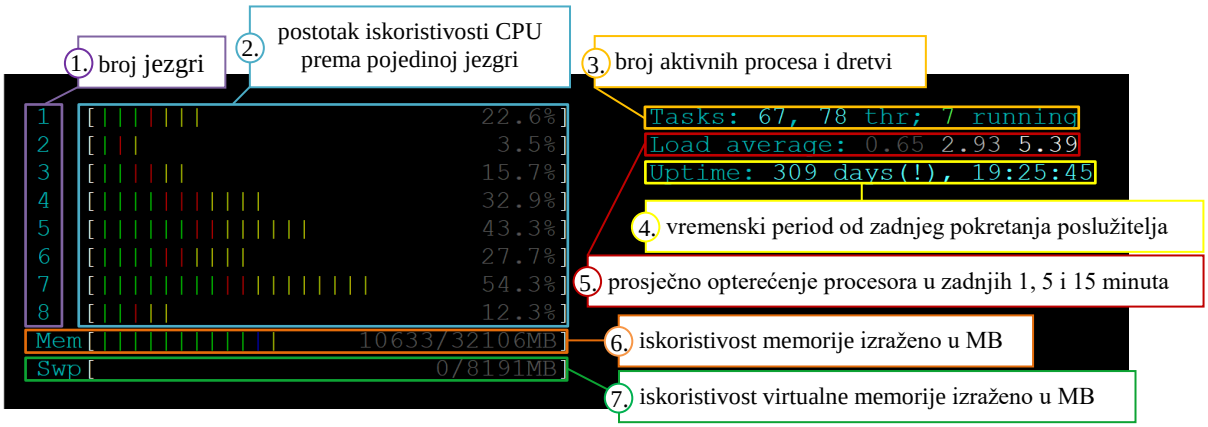
Iz prethodne slike je vidljivo da nema većih odstupanja tijekom godinu dana, odnosno nema sezonski i sličnih pojava. Tablica 7.1. predstavlja broj korisnički zahtjeva u sekundi prema različitim vremenskim periodima za jednu Web-farmu.

Tablica 7.1. - Broj korisničkih zahtjeva u sekundi iz stvarnog sustava prema različitim vremenskim periodima za jednu Web-farmu

vremensko razdoblje	minimalno	prosječno	maksimalno
dan	8.17	29.63	62.64
tjedan	7.92	25.49	78.14
mjesec	6.26	23.13	78.36
godina	6.06	24.17	79.31

Iz navedenih rezultata utvrđeno je da su oscilacije najveće tijekom jednog dana, odnosno da je razlika između minimalne i maksimalne vrijednosti najveća. Proučavanjem ostalih vremenski razdoblja (tjedan, mjesec i godina) utvrđeno je da nema većih odstupanja od onih koje su prisutne unutar 24 sata, odnosno može se reći da su razlike zanemarive. Iz navedenog se može zaključiti broj korisničkih zahtjeva za period od 24 sata predstavlja reprezentativan uzorak pomoću kojeg su kasnije definirati parametri za eksperimente koji su korišteni prilikom validacije novog modela.

Već je spomenuto da je rezultat obrada korisničkih zahtjeva opterećenje računalnih resursa. Prije nego što se navedeni rezultati prezentiraju potrebo je objasniti način njihovog mjerenja. Slika 7.6. predstavlja grafički prikaz Linux sučeljen nakon pokretanja naredbe *htop*.



Slika 7.6. - Grafički prikaz dijela Linux sučelja nakon pokretanja naredbe *htop*

Naredba omogućava interaktivni prikaz opterećenja resursa u stvarnom vremenu za procesor i memoriju. Predstavlja poboljšanu inačicu *top* naredbe koja je osnovni dio svakog Linux operacijskog sustava i služi za upravitelje njegovih procesa (engl. *task manager*). Na slici 7.6. su prikazane glavne komponente naredbe *htop*, a one su:

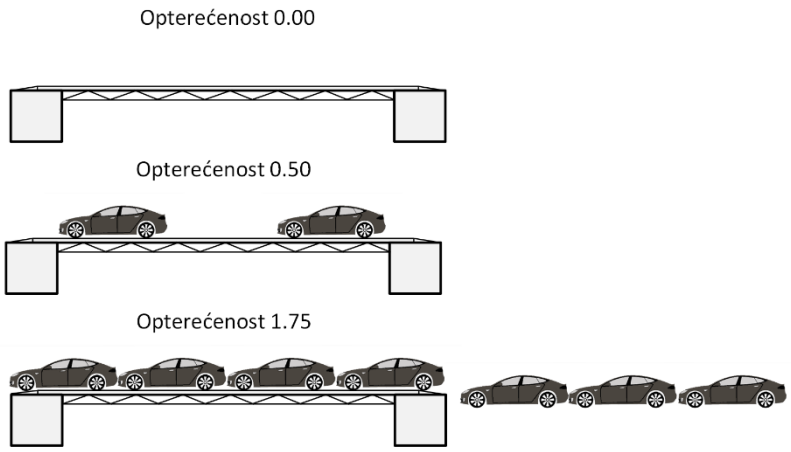
1. broj jezgri kojima operacijski sustav raspolaže,

2. postotak iskoristivosti CPU prema pojedinoj jezgri (prava boja - dretve niskog prioriteta, zelena boja - dretve normalnog prioriteta (korisnik), crvena boja - kernel dretve, žuta - dretve za virtualizaciju),
3. broj aktivnih procesa i dretvi (u navedenom primjeru ima 67 procesa i 78 dretvi od kojih se 7 trenutno izvršava),
4. vremenski period od zadnjeg pokretanja poslužitelja (odnosno koliko dugo operacijski sustav radi bez prekida),
5. *load average* mjera prikazuje prosječan broj procesa koji je koristio ili tražio korištenje CPU-a u zadnjih 1, 5 i 15 minuta izraženo u obliku decimalnog broja. Na osnovu te mjere se može očitati stvarno prosječno opterećenje procesora primjenom sljedeće formule:

$$stvarno\ prosječno\ opterećenje = \frac{load\ average}{broj\ jezgri} \times 100$$

   Npr. vrijednost koju *load average* pokazuje je 2.65, a broj raspoloživih jezgri je pet. U ovom slučaju stvarno prosječno opterećenje CPU-a iznosi 53%,
6. iskoristivost memorije izraženo u MB (zelena boja - iskorištena memorija, plava boja - međuspremnik, žuta boja - predmemorija) i
7. iskoristivost virtualne memorije izraženo u MB (koristi se u slučajevima kada je fizička memorija u potpunosti iskorištena, onda se koristi virtualna memorija koja pohranjuje podatke u memoriju na disku) [91].

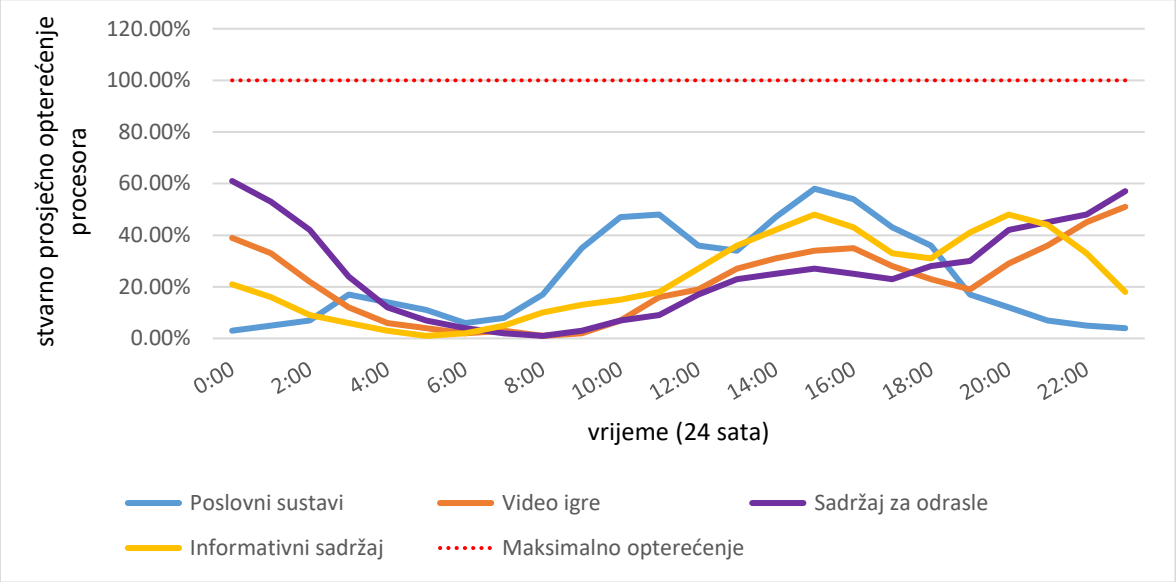
Važno je napomenuti da je prosječno opterećenje procesora prema naredbi *htop* može biti veće od 100% i mjeri se u tri različita vremenska perioda, a ona su: 1, 5 i 15 minuta. Prosječno opterećenje procesora ima važnu ulogu jer se uz pomoć njega može utvrditi je li poslužitelju potreban veći broj jezgri ili ne. Drugim riječima, on predstavlja indikator kolika bi mogla biti iskoristivost CPU prema pojedinoj jezgri. Npr. ako jedan poslužitelj ima jednu procesorsku jezgru i ako je prosječno opterećenje takvog poslužitelja 1.75, to znači da je došlo do preopterećenja u iznosu od 75%, odnosno da 0.75 pripravnih procesa mora čekati na izvođenje. Ovaj slučaj je metaforički objašnjen slikom 7.7., gdje ulogu procesora ima most, dok vozila predstavljaju opterećenje.



Slika 7.7. - Grafički prikaz opterećenja mosta

U posljednjem slučaju sa slike 7.7. je vidljivo da određen broj vozila čeka na oslobođenje mosta, što zapravo predstavlja pripravne procese koji moraju čekati zbog preopterećenosti procesora. Riječ je zapravo o slučaju kada je vrijednost prosječnog opterećenje procesora veća od samog broja jezgri kojim poslužitelj raspolaže. To je indikator da je sustav preopterećen i da je potrebno više resursa (broja jezgri) jer je došlo do zagušenja sustava. U nastavku rada kada se spominje pojam opterećenja iznad 100% onda se to odnosi na vrijednosti dobivene naredbom *htop*.

Broj korisničkih zahtjeva koji su dobiveni u preliminarnom istraživanju (Tablica 7.1.) rezultiraju određena prosječna opterećenja procesora koja su prikana na slici 7.8. U navedenom istraživanju korištena su četiri različita Web-klastera od kojih je svaki imao od 50 do 100 Web-stranica. Web-klasteri se u ovom slučaju razlikuju prema vrsti poslovanja, odnosno sadržaja na: poslovne, video igre, sadržaj za odrasle i informativne. Slika 7.8. prikazuje stvarno prosječno opterećenja procesora za četiri Web-klastera tijekom 24 sata prema naredbi *htop*. Koristi se period od 24 sata jer je prethodno utvrđeno da je on reprezentativan s obzirom da opterećenje računalnih resursa ovisi o broju korisničkih zahtjeva.

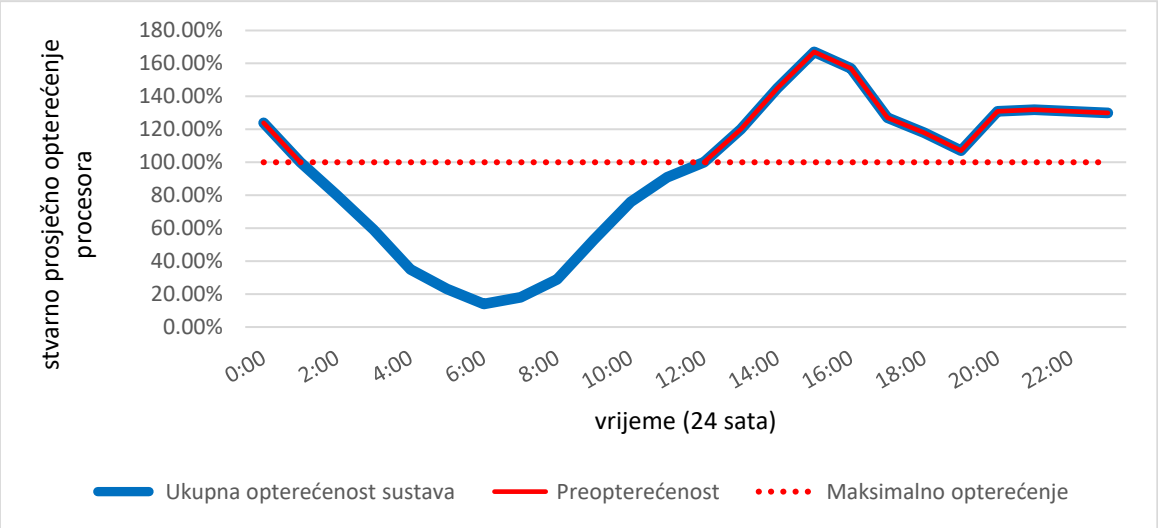


Slika 7.8. - Prikaz stvarnog prosječnog opterećenja procesora za četiri Web-klastera tijekom 24 sata prema naredbi *htop*

Iz prethodne slike moguće je zaključiti da niti jedan Web-klaster nije pod stvarnim opterećenjem većim od 65%, odnosno da njegovi računalni resursi nisu u potpunosti iskorišteni. Razlog tome je što se želi osigurati kapacitet odnosno resursi za iznenadna opterećenja sustava (engl. *peaks*). Na prvi pogled idealno stanje bi bilo kada bi stvarno prosječno opterećenje iznosilo 100%. To bi značilo da je iskoristivost svake jezgre procesora 100%, ali to je u praksi vrlo teško postići i najčešće nije dobro za poslovni sustav. To bi značilo da su svi resursi maksimalno iskorišteni i da nema zahtjeva koji čekaju u redu na izvršavanje da se resursi oslobode. Drugim riječima, opterećenje iznad 100% predstavlja stanje u kojem je sustav preopterećen, odnosno zahtjevi čekaju u redu na izvršavanje da se resursi oslobode. U ovakvim situacijama vrlo vjerojatno će doći do zagušenja sustava što najčešće rezultira da sustav stane raditi, što u današnjem poslovanju nije prihvatljivo.

U ovom slučaju opterećenost sustava ovisi o broju korisničkih zahtjeva, što predstavlja vanjsku varijablu sustavu koja je nepredvidljiva. Zbog toga je potrebno ostvariti sigurnosne mehanizme koji bi spriječili preopterećenost sustava većim iskorištenjem postojećih resursa, ali nikako iznad 100%. Na prethodnoj slici se također vidi da sva četiri Web-klastera imaju različit broj posjeta tijekom 24 sata, odnosno neki zahtijevaju veću količinu resursa tijekom dana (informativne i poslovne), a drugi tijekom noći (video igre i sadržaj za odrasle). Drugim riječima, opterećenja resursa Web-klastera tijekom 24 sata nisu jednaka, a pritom je potrebno osigurati da svaki klaster ima rezervirane resurse za situacije kada dođe do iznenadnog povećanog broja zahtjeva. Upravo zbog toga se postavlja pitanje kako je moguće djelovati na resurse koji su u stanju mirovanja, te kako njih raspodijeliti odnosno prebaciti u dio sustava gdje su potrebniji.

Navedena četiri Web-klastera imaju jednake performanse, odnosno svaki Web-klaster sastoji se od tri Web-poslužitelja s istim specifikacijama. Ako se na trenutak zanemare svi prethodno navedeni razlozi zbog kojih je potreban veći broj Web-klastera i ako se grafički zbirno prikaže prosječno opterećenje resursa za sve četiri Web-klastera tijekom 24 sata (Slika 7.9.) vidljivo je da su prisutne još veće nepravilnosti u iskoristivosti resursa. Dio resursa nikad neće biti iskorišten, a pritom je prisutan vremenski period kada je sustavu potrebno iznad 100% resursa. To bi značilo da bi svi ostali zahtjevi nakon 100% čekali u redovima na izvršenje što bi rezultiralo lošijim performansama cijelog sustava.



Slika 7.9. - Simulacija ukupnog stvarnog prosječnog opterećenja procesora prema za četiri Web-klastera tijekom 24 sata prema naredbi *htop*

Dakle, problem nije u povećanju resursa ili grupiranju Web-stranica neovisno o vrsti sadržaja, već kako postojeće slobodne resurse bolje iskoristiti. Navedeni problem niti danas nije u potpunosti riješen.

U ovom istraživanju se pod pojmom računalni resursi misli na procesor i memoriju. Međutim, u preliminarnom istraživanju na primjeru Web-poslužitelja je utvrđeno da se resursi kao što je procesor (broj jezgri) znatno slabije iskorištavaju za razliku od memorije jer imaju veće oscilacije u opterećenju. Upravo je zbog toga procesor naveden kao primjer nedovoljne iskoristivosti postojećih računalnih resursa koje poslužitelj mora imati na raspolaganju zbog prethodno navedenih razloga.

7.1 Utvrđivanje problema i motivacija

U ovom koraku istraživačke paradigme znanost o dizajnu se definiraju glavni problemi i motivacija za ovo istraživanje. U prethodnim poglavljima ovog rada nalazi se detaljniji opis problema koji su razlog ovog istraživanja. Jedan od glavnih problema ovog istraživanja je nedovoljna iskoristivost postojećih računalnih resursa, koja indirektno utječe na povećanje kapitalnih i operativnih troškova.

Tijekom proučavanja postojećih rješenja utvrđeno je da ne postoji dovoljno učinkovito rješenje koje bi omogućilo veću iskoristivost postojećih računalnih resursa (CPU i memorija), a da pritom nije potrebno uraditi ponovno pokretanje poslužitelja kako bi se to postiglo. Upravo je to bila dodatna motivacija da se ovo istraživanje provede.

7.2 Cilj rješenja

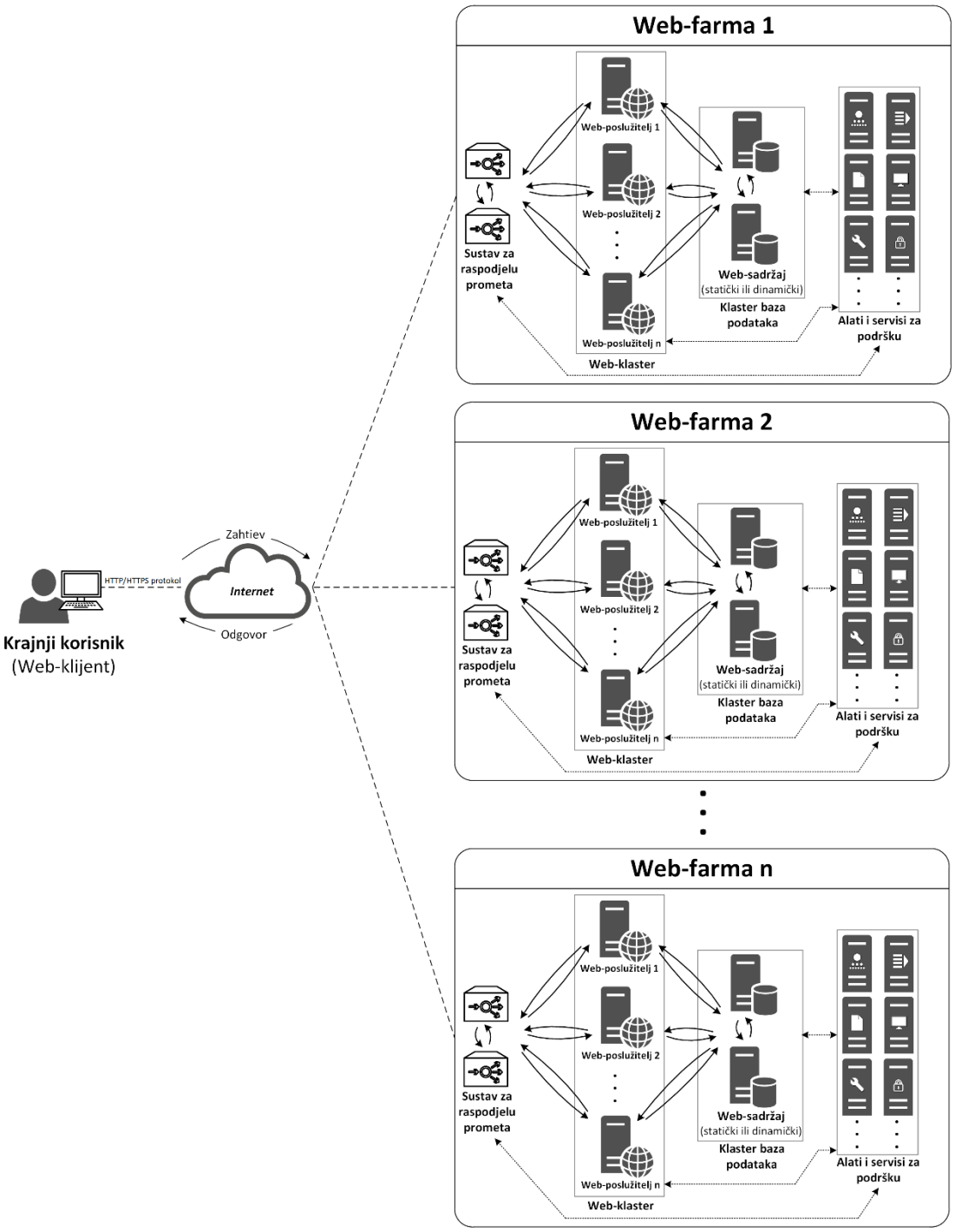
Nakon što je opisana problematika i motivacija istraživanja definirani su ciljevi. Glavni ciljevi ovog istraživanja su također opisani u jednom od prethodnih poglavlja ovog rada (poglavlje broj 5). Postavljena su ukupno tri cilja (C1, C2 i C3) koji su postignuti rješavanjem sljedećih koraka:

- Analiza literature i postojećih rješenja te evaluacija alata ako je potrebno.
- Pronalazak odgovarajuće metode.
- Izgradnja modela i prijedlog novog rješenja.
- Definiranje ograničenja i identifikacija uskih grla sustava.
- Analiza ponašanja stvarnih sustava (broj korisničkih zahtjeva i opterećenje resursa u određenom vremenu).
- Instalacija i konfiguracija testnog okruženja.
- Instalacija i konfiguracija alata koji su potrebni za validaciju modela.
- Izgradnja aplikacije na temelju modela.
- Definiranje eksperimenta i metrike pomoću kojih se provjeriti novi model.
- Ispitivanje glavnih komponenti novog modela, odnosno provjera prve hipoteze (mogućnost dodavanja i oduzimanje računalnih resursa bez potrebe za ponovnim pokretanjem poslužitelja).
- Provjera je li primjena predloženog modela narušila konzistentan rad poslužitelja, odnosno provjera druge hipoteze.
- Provesti eksperimente s i bez primjene novog modela.
- Usporediti rezultate, odnosno dokazati da se postigla veća iskoristivost postojećih računalnih resursa (provjera treće hipoteze).
- Istražiti i pronaći slučajeve i ograničenja kod kojih predloženi model pokazuje bolje rezultate s obzirom na postojeća rješenja.

7.3 Dizajn i razvoj

Nakon što su definirani ciljevi ovog istraživanja slijedi prijedlog i izgradnja novog konceptualnog modela za raspodjelu postojećih računalnih resursa bez potrebe za ponovnim pokretanjem poslužitelja. Resursi koji se mogu mijenjati novim modelom su CPU odnosno broj

jezgri i radna memorija. Novi model je koncipiran na način da ne ovisi o virtualizacijskoj platformi i programskom kodu u kojem je ostvaren. U nastavku rada je prikazana arhitektura Web-farmi (Slika 7.10.), jer je već spomenuto kako je validacija novog modela izvršena na primjeru Web-poslužitelja.



Slika 7.10. - Arhitektura Web-farmi

Na slici 7.10. je prikazana arhitektura Web-farmi koja je detaljnije opisana na slici 2.3. Prikaz arhitekture je bitan kako bi se bolje razumjele veze između komponenata novog modela i sustava nad kojim je primijenjeno novo rješenje. U istraživanju se koristi Web-farma s dva Web-klastera kako bi se provjerila skalabilnost rješenja. Cilj je postići automatiziranu i

poboljšanu raspodjelu računalni resursa Web-poslužitelja unutar jednog i više Web-klastera, odnosno više fizičkih poslužitelja.

Tijekom istraživanja i dizajna novog modela koriste se mnogobrojne metode i tehnike, a neke od njih su:

- Tehnike modeliranja: UML (*Unified Modeling Language*).
- Dijagramske tehnike: dijagram uzročno-posljedičnih veza.
- Strukturna analiza procesa: dijagrami dekompozicije, dijagrami toka podataka i blok dijagram.
- Programiranje: Pseudo jezik i skriptni jezik (BASH i PHP).
- Metode: uspoređivanje, eksperiment, evaluacija/validacija, analiza sadržaja.

Dizajn i razvoj predstavljaju jedan od najvažnijih koraka istraživačke paradigme znanost o dizajnu jer se u ovom koraku dizajnira odnosno gradi novi artefakt što u ovom slučaju predstavlja model za automatiziranu i poboljšanu raspodjelu postojećih računalnih resursa. Postoje određeni preduvjeti koji su važni kako bi model mogao automatizirano i poboljšano raspoređivati postojeće resurse, a oni su:

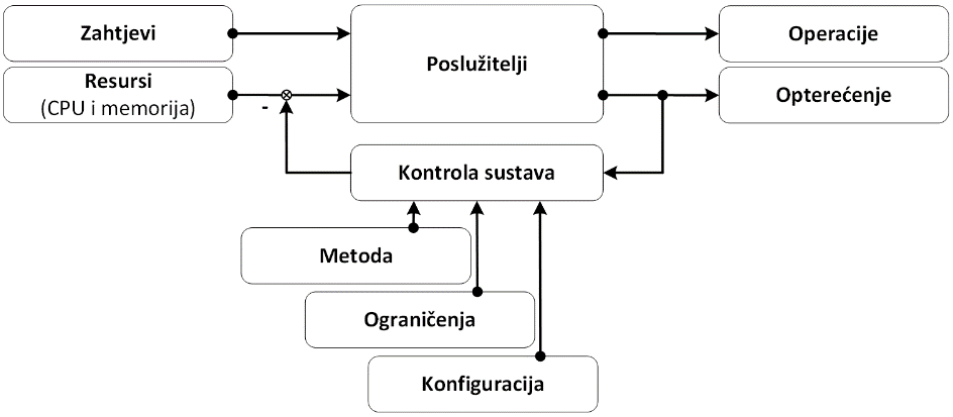
1. *Stalna praćenja i mjerenja opterećenosti poslužitelja* - odnosno računalnih resursa (CPU i memorija), ali i provjere stanja resursa samog fizičkog poslužitelja.
2. *Mogućnost oduzimanja i dodavanja računalnih resursa bez potrebe za ponovnim pokretanjem poslužitelja* - kako bi se postigla poboljšana iskoristivost postojećih računalnih resursa jer je važno da poslužitelj ima dovoljno računalnih resursa za svoj pravilni rad. Tijekom izmjene računalnih resursa potrebno je paziti na dostupnost sustava tj. ne smije se desiti prekid u radu sustava odnosno ne smije se narušiti konzistentan rad poslužitelja.
3. *Napredna raspodjela resursa* - mogućnost dodavanja veće količine resursa u kritičnim situacijama ako je to potrebno. Sustav bi trebao naprednim metodama prepoznati kojom brzinom se resursi troše i prema tome odlučiti za koliko je potrebno povećati resurse, ali i oduzeti ako ih poslužitelji više ne treba.

U nastavku rada slijedi detaljna razrada novog modela počevši od konceptualne razine pa do ostvarene razine uz pomoć koje je razvijeno programsko rješenje uz pomoć kojeg je izvršena validacija ovog modela.

7.3.1 Konceptualni model

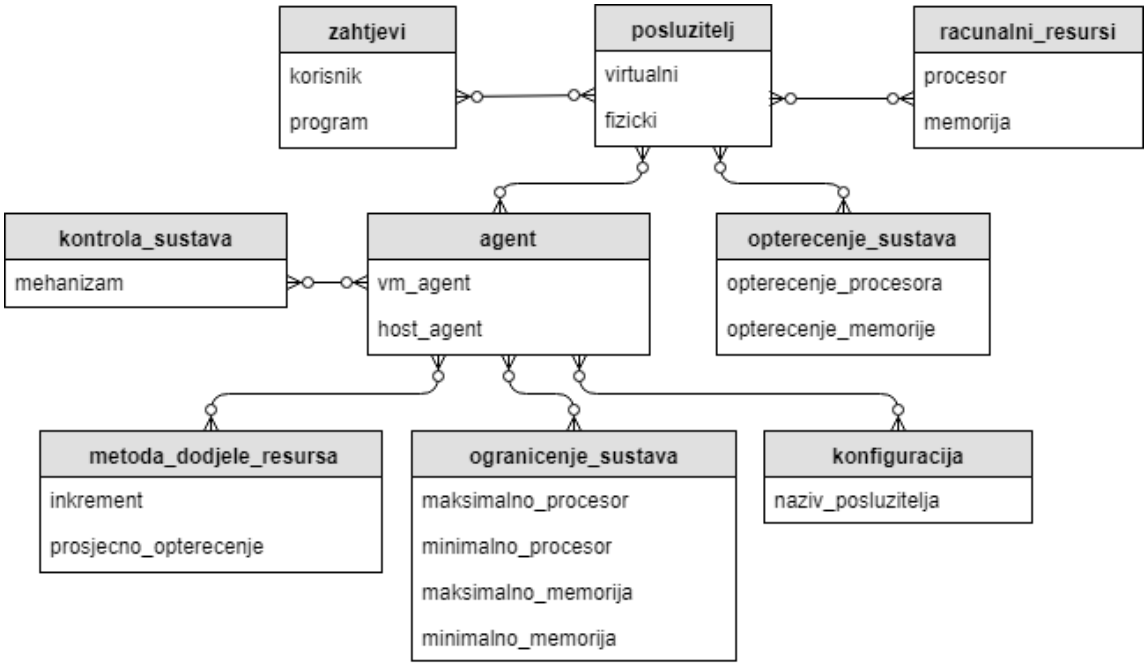
Prije izgradnje modela za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa potrebno je prikazati kako sustav radi na sustavnoj razini odnosno napraviti metamodel

sustava automatskog upravljanja (Slika 7.11.). Generalizacija sustava je potrebna kako bi se omogućila što šira primjena modela neovisna o virtualizacijskoj platformi i programskom jeziku.



Slika 7.11. - Metamodel sustava automatskog upravljanja

Sustav se sastoji od dvije ulazne komponente, a one su: zahtjevi (aplikacija ili korisnički zahtjevi) i resursi (CPU i memorija). Središnji dio sustava predstavljaju poslužitelji koji uz pomoć resursa pretvaraju zahtjeve u izlazne komponente: operacije (rezultat obrade zahtjeva) i opterećenje koje nastaje prilikom obrade zahtjeva. Glavna komponenta ovog sustava za automatsko upravljanje je kontrola sustava koja predstavlja negativnu povratnu vezu u obliku zatvorene petlje koja obavlja stalnu provjeru opterećenja sustava i na temelju posebnih parametara (konfiguracija, ograničenja i metoda) obavlja raspodjelu resursa. Vodeća veličina u ovom slučaju predstavlja opterećenje sustava. Sljedeći korak u izradi modela predstavlja razrada modela na višoj razini na kojoj su prikazane glavne komponente sustava sa svojim parametrima i njihovim međusobnim vezama (Slika 7.12.).



Slika 7.12. - ERA model novog modela

Sve komponente modela (entiteti) i njihove veze s prethodne slike su u nastavku rada detaljno opisani, a svi mehanizmi detaljno razrađeni u obliku pseudo jezika kako bi se postigla generalizacija rješenja. Ovaj pristup omogućava primjenu modela neovisno o virtualizacijskoj platformi i programskom kodu. U nastavku rada su detaljnije opisane komponente (entiteti) ovog modela, a one su:

- ***Zahtjevi*** - predstavljaju osnovni dio ovakvih modela i mogu biti generirani od strane aplikacija (npr. različiti servisi) i korisničkih zahtjeva. Razlika između ove dvije grupe zahtjeva je to što su aplikativni najčešće predvidljivi i moguće ih je izračunati pa i u nekim slučajevima definirati kad da se stvaraju. Korisnički zahtjevi najčešće predstavljaju krajnjeg korisnika i oni su najčešće nepredvidljivi. S obzirom da je validacija modela na primjeru Web-poslužitelja ovdje se promatraju HTTP/HTTPS zahtjevi.
- ***Poslužitelj*** - također predstavljaju osnovnu i centralnu komponentu sustava jer upravo oni omogućavaju da se spomenuti sustavi obrade. Postoje dvije vrste poslužitelja, a oni su virtualni i fizički. Na fizičkim poslužiteljima se nalaze virtualni Web-poslužitelji, ali i svi ostali koji su bitni za validaciju modela kao što su sustavi za raspodjelu HTTP/HTTPS zahtjeva i baze podataka.
- ***Računalni resursi*** - predstavljaju komponentu koja se odnosi na obje vrste poslužitelja tj. virtualne i fizičke. U ovom slučaju resurse predstavljaju CPU (broj jezgri) i radna memorija.
- ***Opterećenje sustava*** - odnosno opterećenje spomenutih resursa nastaje kao rezultat obrade zahtjeva koji se za CPU mjeri u postotcima, a za memoriju u KB (kilobajt) zbog preciznijeg mjerenja temeljem kojeg se donosi odluka o raspodjeli resursa.

U nastavku rada slijedi detaljniji opis i razrada ostalih komponenti čija međusobna veza i rad omogućavaju da navedeni model pruža bolje rezultate od postojećih rješenja.

7.3.1.1 Konfiguracija

Konfiguracija predstavlja polaznu komponentu novog modela jer su u njoj definirane važne stavke kao što su:

- *globalni parametri* - predstavljaju parametre koje koriste sve komponente sustava, a oni su:
  - naziv poslužitelja,
  - minimalna dozvoljena količina resursa procesora (broj jezgri),
  - maksimalna dozvoljena količina resursa procesora (broj jezgri),

- trenutna količina resursa procesora (broj jezgri),
- minimalna definirana količina memorije (izraženo u GB (gigabajt)),
- maksimalna definirana količina memorije (izraženo u GB (gigabajt)),
- trenutna količina memorije (izraženo u KB (kilobajt) zbog preciznijeg mjerenja na temelju koje se donosi odluka o promjeni količine memorije),
- profili dodjele resursa procesora,
- profili dodjele memorije,
- ukupan broj dodijeljenih jezgri virtualnim poslužiteljima (ovaj parametar ovisi o broju jezgri fizičkog poslužitelja i služi kao dodatna kontrola kako se ne bih dodijelilo više jezgri nego što ih ima sam fizički poslužitelj na raspolaganju),
- ukupna memorija koja je dodijeljena virtualnim poslužiteljima izražena u KB (kilobajt). Ona je također izražena u KB kako bi se odluke o promjeni količine memorije mogle donositi što preciznije. Ovaj parametar također ovisi o memoriji fizičkog poslužitelja i služi kao dodatna kontrola kako se ne bih dodijelilo više memorije nego što ih ima sam fizički poslužitelj na raspolaganju,
- ukupna memorija koja je dodijeljena virtualnim poslužiteljima izražena u GB (gigabajt). Ovdje vrijede ista pravila kao za prethodno navedeni parametar.
- prosječno opterećenje procesora virtualnog poslužitelja koji se u tom trenutku provjerava i
- zauzeće memorije za virtualnog poslužitelja koji se u tom trenutku provjerava.

- *početni parametri* - sustav uvijek provjerava da li su prisutni posebno definirani parametri za ograničenja resursa, ako nisu onda se koriste parametri iz glavne konfiguracije koji vrijede za sve poslužitelje, a oni su:
  - minimalna početna vrijednost resursa procesora (broj jezgri),
  - maksimalna početna vrijednost resursa procesora (broj jezgri),
  - minimalna početna vrijednost raspoložive memorije (izraženo u GB (gigabajt)),
  - maksimalna početna vrijednost raspoložive memorije (izraženo u GB (gigabajt)),
  - početni profili resursa procesora (broj jezgri) i
  - početni profili memorije (izraženo u GB (gigabajt)).

Prilikom navođenja i objašnjavanja svih stavki modela, više puta se primjenjuje pseudo jezik. Na taj način omogućeno je lakše razumijevanje, ali i mogućnost prilagodbe i korištenje modela u bilo kojem drugom programskom jeziku. Slika 7.13. predstavlja pseudo jezik koji se odnosi na početne parametre.

```
1.  izlaz („Unos početnih parametara: “)
2.  ulaz (novi_cpu_min)
3.  ulaz (novi_cpu_max)
4.  ulaz (novi_ram_min)
5.  ulaz (novi_ram_max)
6.  ulaz (novi_cpu_balance_logic)
7.  ulaz (novi_mem_balance_logic)
8.  ako je (učitan novi_cpu_min) onda
9.      cpu_min := novi_cpu_min
10. inače
11.     cpu_min := predodređeni_cpu_min
12. ako je (učitan novi_cpu_max) onda
13.     cpu_max := novi_cpu_max
14. inače
15.     cpu_max := predodređeni_cpu_max
16. ako je (učitan novi_ram_min) onda
17.     ram_min := novi_ram_min
18. inače
19.     ram_min := predodređeni_ram_min
20. ako je (učitan novi_ram_max) onda
21.     ram_max := novi_ram_max
22. inače
23.     ram_max := predodređeni_ram_max
24. ako je (učitan novi_cpu_balance_logic) onda
25.     cpu_balance_logic := novi_cpu_balance_logic
26. inače
27.     cpu_balance_logic := predodređeni_cpu_balance_logic
28. ako je (učitan novi_mem_balance_logic) onda
29.     mem_balance_logic := novi_mem_balance_logic
30. inače
31.     mem_balance_logic := predodređeni_mem_balance_logic
32. izlaz („Ispis konačnih početnih parametara: “)
33. izlaz (novi_cpu_min)
34. izlaz (novi_cpu_max)
35. izlaz (novi_ram_min)
36. izlaz (novi_ram_max)
37. izlaz (novi_cpu_balance_logic)
38. izlaz (novi_mem_balance_logic)
```

Slika 7.13. - Pseudo jezik za početne parametre

Već kod objašnjavanja prethodno navedenog postupka primijenjen je pseudo jezik (Slika 7.13.). Kod unosa podataka koristi se naredba „unos“, dok se za ispis podataka na zaslon koristi naredba „izlaz“. Ovaj postupak se svodi na jednostavnu provjeru, gdje se selekcijama „ako je - inače“ provjerava da li će biti korištene posebne vrijednosti za početne parametre ili već unaprijed zadane vrijednosti. Ako

se žele koristiti posebne vrijednosti za pojedine početne parametre, korisnik onda mora na početku unijeti nove vrijednosti za te parametre.

- *parametri ograničenja* - ovi parametri služe za definiranje donjih i gornjih granica resursa, odnosno koliko je minimalno i maksimalno dozvoljeno stanje resursa, a oni su:
    - minimalno dozvoljeno opterećenje resursa procesora (izraženo u %),
    - maksimalno dozvoljeno opterećenje resursa procesora (izraženo u %),
    - minimalno dozvoljeno zauzeće memorije (izraženo u %),
    - maksimalno dozvoljeno zauzeće memorije (izraženo u %),
    - maksimalno resursa procesora (broj jezgri) i
    - maksimalno memorije (izraženo u GB (gigabajt)).
- *postavke za zapise (sustav za bilježenje promjena)* - definirani su parametri kao što su lokacija i format.
- *postavke za razmjenu informacija* - ovdje su definirani parametri za komponentu agent koji joj omogućavaju razmjenu informacija o stanju resursa između fizičkog poslužitelja i njihovih virtualnih poslužitelja.

Kako bi se pojednostavila uporaba modela, parametri koje definira administrator sustava, a odnose se na memoriju izraženi su u GB (gigabajt). Svi navedeni parametri koji se nalaze u konfiguraciji mogu imati proizvoljne vrijednosti, odnosno mogu se mijenjati ovisno o potrebama poslužitelja. U nastavku slijedi par primjera profila za memoriju (ista pravila vrijede i za CPU):

- 2:1 - ako je dodijeljena memorija na poslužitelju veća od 2 GB onda povećaj ili smanji za 1 GB,
- 4:2 - ako je dodijeljena memorija na poslužitelju veća od 4 GB onda povećaj ili smanji za 2 GB,
- 16:4 - ako je dodijeljena memorija na poslužitelju veća od 16 GB onda povećaj ili smanji za 4 GB.

Profili predstavljaju predefinirana pravila dodavanja odnosno oduzimanja resursa poslužitelja. Kako bi navedene primjere profila lakše razumjeli navedena su dva scenarija:

- U trenutku provjere resursa poslužitelj je imao 6 GB memorije i ako je potrebno dodati još resursa, povećat će za 2 GB.
- Ako je poslužitelj imao 18 GB memorije, a potrebno je smanjiti resurse onda će to biti za 4 GB.

Osim generalnih parametara ograničenja kao što su donja i gornja granica za CPU i memoriju, ovo rješenje ima i dodanu mjeru koja se zove minimalna i maksimalna slobodna memorija koja omogućava još veću iskoristivost postojećih računalnih resursa. Uz pomoć nje se dodatno kontrolira razmak između donje i gornje granice, ali i osiguravaju resursi za normalan rad poslužitelja. Kako bi se ovo bolje razumjelo naveden je konkretan primjer u kojem su definirani parametri ograničenja donja granica (40%) i gornja granica (80%):

- U slučaju da virtualni poslužitelj ima 2 GB memorije to bi značilo da su njegove granice 0.8 GB odnosno 1.6 GB, drugim riječima poslužitelj ima 0.8 GB resursa koji su dodijeljeni njemu za pravilni rad. U ovom scenariju nema puno resursa koji se nepravilno koriste jer su granice blizu jedna druge.
- Međutim, ako virtualni poslužitelj ima dodijeljeno više resursa, npr. 10 GB, onda su njegove granice 4 GB odnosno 8 GB. To znači da ima minimalno na raspolaganju 4 GB (od 0 do 4 GB). U ovakvim scenarijima ti minimalni resursi najčešće nisu dovoljno iskorišteni jer je za pravilni rad poslužitelja najčešće potrebno puno manje (npr. za rad samog operacijskog sustava).

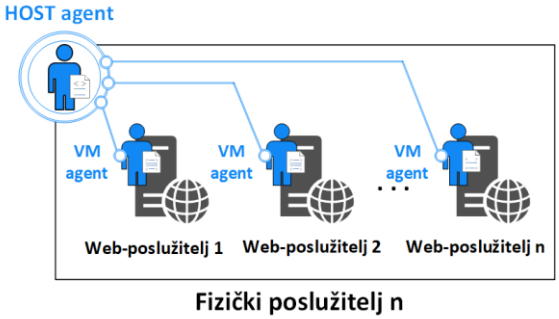
Kako bi se izbjegao navedeni scenarij ostvarene su dodatne mjere kao što su minimalna i maksimalna slobodna memorija koje osiguravaju da poslužitelj, odnosno njegov operacijski sustav ima dovoljno resursa za svoj rad neovisno o opterećenju sustava. Upravo ovo precizno definiranje minimalnih odnosno maksimalnih resursa koji su potrebni za rad poslužitelja omogućuju da se postojeći resursi još više iskoriste.

7.3.1.2 Agent

Komponenta agent predstavlja program koji ima poseban zadatak i ulogu. Model se sastoji od dvije vrste agenata, a oni su:

1. ***HOST agenti*** - se nalaze na fizičkim poslužiteljima i njihov glavni zadatak je provjera i raspodjela računalnih resursa cijelog fizičkog poslužitelja.
2. ***VM agenti*** - se nalaze na virtualnim poslužiteljima i njihov glavni zadatak predstavlja slanje izvještaja o stanju vlastitih računalnih resursa.

Slika 7.14. predstavlja raspored agenata na poslužiteljima. Plava linija predstavlja komunikaciju između agenata.



Slika 7.14. - Raspodjela agenata na poslužiteljima

Ovakav način komunikacije omogućava da HOST agent ima informaciju o opterećenju resursa svojih virtualnih poslužitelja, ali i ukupno stanje resursa kojima on raspolaže. Upravo ova razmjena informacija rezultira posljednjim korakom, koji predstavlja raspodjelu računalnih resursa između poslužitelja. Cijeli proces je grupiran u dvije faze:

1. ***Inicijalizacija*** - proces započinje provjerom koliko je ukupno resursa dodijeljeno virtualnim poslužiteljima i na temelju toga se računa koliko fizički poslužitelj ima resursa na raspolaganju. Nakon toga slijedi očitavanje glavne konfiguracije, ograničenja virtualnih poslužitelja i metode dodjele resursa. Ovo je iterativni proces koji se odvija za svakog poslužitelja posebno. Pseudo jezikom na slici 7.15. prikazuje jedan od mogućih načina izračuna ukupnog broja jezgri i memorije fizičkog poslužitelja koji su dodijeljeni virtualnim poslužiteljima. Resursi svih virtualnih poslužitelja se zbrajaju, što rezultira dobivanjem ukupne količine dodijeljenih resursa. Kako bi se dobila vrijednost ukupno dodijeljenih resursa korištena je petlja. Važno je napomenuti da se petlja odvija sve dok nisu učitani resursi svih virtualnih poslužitelja.

```
1. ukupni_dodijeljeni_cpu = 0
2. ukupni_dodijeljeni_ram = 0
3. broj_poslužitelja := ukupan broj virtualnih poslužitelja
4. izlaz („Izračunavanje ukupne količine dodijeljenih Resursa procesora i
   memorije…“)
5. dok je (broj_poslužitelja != 0) činiti {
6.     ukupni_dodijeljeni_cpu := ukupni_dodijeljeni_cpu + cpu_poslužitelja
7.     ukupni_dodijeljeni_ram := ukupni_dodijeljeni_ram + ram_poslužitelja
8.     broj_poslužitelja  := broj_poslužitelja – 1 }
9. izlaz („Ukupna količina dodijeljenih Resursa procesora:“)
10. izlaz (ukupni_dodijeljeni_cpu)
11. izlaz („Ukupna količina dodijeljene memorije:“)
```

Slika 7.15. - Postupak izračuna ukupnog broja jezgri i memorije fizičkog poslužitelja koji su dodijeljeni virtualnim poslužiteljima

2. ***Raspodjela resursa*** - postupak provjere i raspodjele resursa za virtualnog poslužitelja nakon čega slijedi provjera konzistentnosti rada poslužitelja. Postupak provjere resursa virtualnog poslužitelja je relativno jednostavan i sastoji se od usporedbe trenutne iskorištenosti i dopuštene količine iskorištenosti resursa. Riječ je samo o načinu

otkrivanja potrebe za povećanjem, odnosno smanjenjem resursa virtualnog poslužitelja. Posebna pažnja je kasnije posvećena mogućim načinima konačne raspodjele resursa. Postupak provjere resursa virtualnog poslužitelja prikazan je sljedećim pseudo jezikom (Slika 7.16.).

```
1.  izlaz („Trenutna konfiguracija virtualnog poslužitelja:“)
2.  izlaz („CPU: “, cpu_poslužitelja)
3.  izlaz („RAM: “, ram_poslužitelja)
4.  izlaz („Izračunavanje trenutne iskorištenosti resursa: „)
5.  iskorištenost_cpu_poslužitelja := trenutna iskorištenost Resursa procesora
6.  iskorištenost_ram_poslužitelja := trenutna iskorištenost memorije
7.  izlaz („CPU iskorištenost: “, iskorištenost_cpu_poslužitelja)
8.  izlaz („RAM iskorištenost: “, iskorištenost_ram_poslužitelja)
9.  ulaz (max_dopuštena_iskorištenost_cpu)
10. ulaz (min_dopuštena_iskorištenost_cpu)
11. ulaz (max_dopuštena_iskorištenost_ram)
12. ulaz (min_dopuštena_iskorištenost_ram)
13. izlaz („Provjeravanje dopuštenih iskorištenosti resursa…“)
14. ako je (iskorištenost_cpu_poslužitelja > max_dopuštena_iskorištenost_cpu)
15.     onda
16.         cpu_promjena := više
17. inače
18.     ako je (iskorištenost_cpu_poslužitelja < min_dopuštena_iskorištenost_cpu)
19.     onda
20.         cpu_promjena := manje
21.     inače
22.         izlaz („Iskorištenost Resursa procesora unutar optimalnog raspona!“)
23. ako je (iskorištenost_ram_poslužitelja > max_dopuštena_iskorištenost_ram)
24.     onda
25.         ram_promjena := više
26.     inače
27.         ako je (iskorištenost_ram_poslužitelja < min_dopuštena_iskorištenost_ram)
28.     onda
29.         ram_promjena := manje
30.             inače
31.     izlaz („Iskorištenost memorije unutar optimalnog raspona!“)
```

Slika 7.16. - Postupak za provjeru resursa virtualnog poslužitelja

Nakon što se obavi drugi korak slijedi provjera da li ima još poslužitelja. Ako ih ima, postupak se ponavlja od početka, u suprotnom se završava i čeka unaprijed proizvoljno definirani vremenski interval kad će se ponoviti cijeli postupak. Pomoću pseudojezika su napisane *Bash Shell* skripte za HOST i VM agente čiji izborni kôdovi se nalazi na kraju ovog rada u prilozima.

7.3.1.3 Ograničenja sustava

Kako bi novi model bio primjenjiv i kako bi ispravno radio, potrebno je osigurati određene preduvjete, a oni su:

1. Virtualizacijska platforma je nužna kako da bi se resursi mogli dijeliti. U ovom slučaju model je validiran na KVM virtualizacijskoj platformi koja je otvorenog koda za razliku od drugih rješenja kao što su VMware i Hyper-V.
2. Operacijski sustav koji podržava oduzimanje i dodavanje računalnih resursa (CPU i memorije) bez potrebe za ponovnim pokretanjem sustava, odnosno dok sustav radi. Primjer operacijskog sustava koji ovo podržava je Linux (kernel 2.6. ili novije inačice [92]). Primjer operacijskog sustava koji za sada ne podržava novo rješenje je Microsoft Windows, jer nije moguće oduzimanje navedenih resursa bez potrebe za ponovnim pokretanjem sustava [93].

Osim preduvjeta koje je potrebno osigurati, važno je ukazati i na određena ograničenja novog rješenja, a oni su:

- *Infrastruktura* - s obzirom da je infrastruktura jednog podatkovnog centra vrlo širok pojam i da je vrlo teško obuhvatiti sve njene komponente u ovom istraživanju glavni fokus je na poslužitelje kao takve.
- *Računalni resursi* - važno je napomenuti da su računalni resursi (CPU i memorija) generalizirani, jer cilj je da model ne ovisi o tehnologiji ili specifikacijama (vrsta proizvođača, model, generacija i sl.). Također je važno napomenuti da ograničenja resursa ovise o količini resursa kojima raspolaže fizički poslužitelj. Diskovni prostor kao računalni resurs u ovom modelu nije uzet u obzir jer za taj problem postoje mnogobrojna rješenja kao centralni sustav za pohranu podataka. Takvi sustavi imaju mnogobrojna programska rješenja koja se brinu o prioritizaciji i raspodjeli diskovnog prostora.

7.3.1.4 Metoda dodjele resursa

Ova komponenta se sastoji od dvije metode odnosno mehanizma pomoću kojih se dodjeljuju i oduzimaju računalni (CPU i memorija) resursi, a oni su:

1. **LA** (Load Average) - napredna dodjela resursa koja je moguća samo za CPU jer se oni promatraju kao prosječno opterećenje što omogućava precizniji i brzi rast resursa. Drugim riječima ako je prosječno opterećenje 500% onda je potrebno dodati pet jezgri kako bi sustav mogao normalno raditi. Broj za koji povećavamo broj jezgri (multiplikator) je također varijabilan što omogućuje još bolje dodjeljivanje resursa. Npr. ako je multiplikator 1.2 i desi se opterećenje od 500% onda će biti dodijeljeno šest jezgri ($1.2 \times 5 = 6$). Ovo omogućava dodjeljivanje dodatnih resursa kao rezerva u slučaju kada je sustav pod neočekivanim

opterećenjem. Osim multiplikatora moguće je birati više načina zaokruživanja rezultata (broja potrebnih jezgri), npr.:

- *HALF UP* - standardno zaokruživanje (npr. 0.5 se zaokružuje na veću vrijednost tj. 1),
- *UP* - zaokruživanje na višu vrijednost (npr. 0.1 je 1) i
- *DOWN* - zaokruživanje na nižu vrijednost (npr. 1.7 je 1).

Ovaj mehanizam dodjele i oduzimanja računalnih resursa zbog lakšeg razumijevanja je predstavljen pomoću pseudo jezika (Slika 7.17.). VM_LA_LOGIC predstavlja varijablu koja sadrži izabranu metodu dodjele, odnosno oduzimanja resursa. Ako je riječ o „LA“ metodi, tada korisnik bira način zaokruživanja vrijednosti. Kao što je već navedeno, riječ je o tri moguća načina zaokruživanja dobivene vrijednosti. Izbor načina zaokruživanja ovisi o samom korisniku i provjerava se pomoću selekcija. Korisnik još pri samom početku određuje koju vrstu zaokruživanja i vrijednost multiplikatora želi. Važno je napomenuti da dobivena vrijednost ne smije prelaziti maksimalnu ili minimalnu dopuštenu količinu resursa procesora. U tom slučaju za novu vrijednost resursa procesora uzima se maksimalna, odnosno minimalna dozvoljena količina.

```
1. ulaz (VM_LA_LOGIC)
2. ulaz (VM_LA_ROUNDING)
3. ulaz (VM_LA_MODIFICATOR)
4. ako je (VM_LA_LOGIC = 'LA') onda
5.     ako je (VM_LA_ROUNDING = „HALF_UP“) onda
6.         cpu_nova_vrijednost := cijela vrijednost (cpu_opterećenost*VM_LA_MODIFICATOR)
7.     ako je (VM_LA_ROUNDING = „UP“) onda
8.         cpu_nova_vrijednost := zaokruži na višu vrijednost (cpu_opterećenost*
   VM_LA_MODIFICATOR)
9.     ako je (VM_LA_ROUNDING = „DOWN“) onda
10.        cpu_nova_vrijednost := zaokruži na nižu vrijednost (cpu_opterećenost*
   VM_LA_MODIFICATOR)
11.        ako je (cpu_nova_vrijednost = 0) onda
12.        cpu_nova_vrijednost := 1
13. ako je (cpu_nova_vrijednost < cpu_min) onda
14. cpu_nova_vrijednost := cpu_min
15. ako je (cpu_nova_vrijednost > cpu_max) onda
16. cpu_nova_vrijednost := cpu_max
17. izlaz („Nova količina Resursa procesora: “)
18. izlaz (cpu_nova_vrijednost)
```

Slika 7.17. - Postupak za LA (Load Average) mehanizam

2. **INCREMENT** - predefinirana pravila uz pomoć kojih se definira način dodjeljivanja odnosno oduzimanja resursa (CPU i memorija) za sve poslužitelje.

Sustav je dizajniran tako da prvo provjerava glavnu konfiguraciju u kojoj su sljedeće vrijednosti:

- *vm* - naziv virtualnog poslužitelja
- *cpu_min* - minimalna dozvoljena vrijednost za CPU (broj jezgri)
- *cpu_max* - maksimalna dozvoljena vrijednost za CPU (broj jezgri)
- *mem_min* - minimalna dozvoljena vrijednost za memoriju (izražena u GB)
- *mem_max* - maksimalna dozvoljena vrijednost za memoriju (izražena u GB)
- *cpu_balance_logic* - profil rasta, odnosno smanjivanja resursa procesora (X:Y, gdje X predstavlja vrijednost opterećenosti sustava, a Y faktor rasta, odnosno smanjenja resursa)
- *mem_balance_logic* - profil rasta, odnosno smanjivanja memorije (izračunava se isto kao prethodna vrijednost).

Najvažnije vrijednosti za ovaj mehanizam dodjele i oduzimanja računalnih resursa jesu profili rasta, odnosno smanjivanja resursa procesora i memorije. Oni zapravo predstavljaju niz omjera koji odgovaraju različitim resursima kojim virtualni poslužitelji mogu raspolagati. Kako bi se utvrdio koji je omjer pogodan za trenutnu konfiguraciju virtualnog poslužitelja, koristi se petlja koja provjerava sve omjere profila. Omjer je pogodan ako je njegov prvi parametar jednak ili manji od trenutnih resursa procesora ili memorije virtualnog poslužitelja. U tom slučaju drugi parametar omjera određuje veličinu dodijeljenog, odnosno oduzetog resursa. Resursi se povećavaju ako je operator jednak aritmetičkom operatoru za zbrajanje, a smanjuju ako je jednak aritmetičkom operatoru za oduzimanje. Ovaj postupak sveden samo na povećanje, odnosno oduzimanja memorije virtualnog poslužitelja, prikazan je pomoću sljedećeg pseudo jezika (Slika 7.18.).

```
1.  dodijeljeni _resurs := 1
2.  ako je (ram_promjena != 0) onda
3.      ako je (ram_promjena = više) onda
4.          operator := '+'
5.      inače  operator := '-'
6.      izlaz („Izračunavanje nove količine memorije…“)
7.      ram_poslužitelja = trenutna radna memorija virtualnog poslužitelja
8.      za (svaki omjer iz mem_balance_logic) činiti
9.          limit_profil := prvi parametar omjera
10.         inkrement_profil := drugi parametar omjera
11.         ako je (ram_poslužitelja >= limit_profil) onda
12.             dodijeljeni_resurs = inkrement_profil
13.         inače   prekid petlje
14.     ako je (operator = '+') onda
15.         ram_nova_vrijednost := ram_poslužitelja + dodijeljeni_resurs
16.     ako je (operator = '-') onda
17.         ram_nova_vrijednost := ram_poslužitelja - dodijeljeni_resurs
18.     ako je (ram_nova_vrijednost < min_vrijednost_memorije) onda
19.         ram_nova_vrijednost := min_vrijednost_memorije
20.     inače
21.         ako je (ram_nova_vrijednost > max_vrijednost_memorije) onda
22.             ram_nova_vrijednost := max_vrijednost_memorije
23.     izlaz („Nova količina radne memorije poslužitelja: „)
24.     izlaz (ram_nova_vrijednost)
```

Slika 7.18. - Postupak za mehanizam INCREMENT

Ako za CPU nije definiran LA način dodjele resursa onda se izračun obavi po INCREMENT principu, što je također prikazano navedenim pseudo jezikom (Slika 7.19.). Važno napomenuti da se kod ovih izračuna poštuju dopuštene minimalne i maksimalne vrijednosti koje su definirane u konfiguraciji.

```
1.  ako je  (VM_LA_LOGIC = 'INCREMENT') onda
2.      za (svaki omjer iz cpu_balance_logic) činiti
3.          limit_profil := prvi parametar omjera
4.          inkrement_profil := drugi parametar omjera
5.          ako je (cpu_poslužitelja >= granica_profil) onda
6.              dodijeljeni_resurs = inkrement_profil
7.          inače   prekid petlje
8.      cpu_nova_vrijednost =  cpu_poslužitelja + dodijeljeni_resurs
```

Slika 7.19. - Postupak za slučaj kada za CPU nije definiran LA način dodjele resursa

7.3.1.5 Kontrola sustava

Kako bi novi model besprijekorno radio ostvareno je nekoliko mehanizama za kontrolu rada sustava, a oni su:

- *Neprekidnost izvršavanja sustava* - Sustav se izvršava periodički i moguće je izmijeniti vrijeme ponovnog pokretanja sustava kako bi se novo rješenje moglo mijenjati ovisno o složenosti samog sustava. Prije ponovnog pokretanja postupka obavi se provjera postojećih procesa koji se izvršavaju. Ako je postupak još u tijeku, onda se iduće pokretanje odgađa dok se trenutno ne obavi. Postupak provjere i održavanja neprekidnosti sustava ostvaren je upotrebom različitih

datoteka. Naime, pri samom početku izvođenja sustava stvori se datoteka za zaključavanje, čije postojanje označava da se sustav trenutno izvršava i da se odgodi novo pokretanje sustava. To sprječava pojavu naglih prekida rada sustava, odnosno omogućava njegov kontinuiran rad. Kada su svi procesi okončani spomenuta datoteka se briše. Osim te datoteke koristi se još i datoteka u kojoj je zapisan broj preskakanja, odnosno broj odgođenih pokretanja sustava. Broj preskakanja se povećava svaki put kada petlja pronađe datoteku za zaključavanje. Ako broj preskakanja pređe dopuštenu vrijednost, dolazi do slanja upozorenja korisniku. Nakon što su svi procesi završeni i datoteka za zaključavanje izbrisana, vrijednost unutar datoteke s brojem preskakanja se postavlja na nulu. Cijeli postupak je prikazan na slici 7.20.

Slika 7.20. - Postupak provjere i održavanja neprekidnosti sustava

```
1.  ulaz (limit_broj_preskakanja)
2.  zaključavanje_datoteka := adresa datoteke za zaključavanje
3.  preskakanje_datoteka := adresa datoteke s brojem preskakanja
4.  ako je ("zaključaj") onda
5.      ako je (postoji datoteka na zaključavanje_datoteka) onda
6.          broj_preskakanja := sadržaj datoteke na preskakanje_datoteka
7.          izlaz („Skripta se već izvršava, preskače novo pokretanje (Broj preskakanja:“)
8.          izlaz broj_preskakanja)
9.          novi_broj_preskakanja:= broj_preskakanja + 1
10.         ako je (novi_broj_preskakanja >= limit_broja_preskakanja) onda
11.             izlaz („Broj preskakanja je veći od dopuštenog“)
12.         upiši vrijednost iz novi_broj_preskakanja u datoteku na preskakanje_datoteka
13.     inače
14.         izbriši sav sadržaj i upiši samo '0' u datoteku na preskakanje_datoteka
15.         stvori novu datoteku na zaključavanje_datoteka
16. ako je ("otključaj") onda
17.     izbriši datoteku na zaključavanje_datoteka
```

- *Nadzor resursa* - postoje dvije razine provjere i nadzora resursa:
    1. *Na razini fizičkog poslužitelja* - HOST agent prije dodjele resursa virtualnom poslužitelju uvijek prvo provjeri raspoložive resurse samog fizičkog poslužitelja kako bi cijeli sustav mogao pravilno raditi. Ovaj postupak se sastoji od računanja nove vrijednosti ukupno dodijeljenih resursa virtualnim poslužiteljima i njenim uspoređivanjem s raspoloživim resursima fizičkog poslužitelja. Nova vrijednost svih dodijeljenih resursa, bilo da je riječ o CPU ili memoriji, dobiva se uz pomoć prethodno izračunate vrijednosti svih dodijeljenih resursa, trenutne vrijednosti i izmijenjene vrijednosti resursa virtualnog poslužitelja. U slučaju da nova vrijednost svih dodijeljenih resursa iznosi više od raspoloživih resursa

fizičkog poslužitelja, dolazi do preskakanja dodjeljivanja novih resursa procesora, odnosno memorije. Preskakanje planirane promjene resursa izvršava se postavljanjem vrijednosti dodijeljenih resursa na nulu. Spomenuti postupak prikazan je na slici 7.21. pomoću pseudo jezika.

Slika 7.21. - Postupak za provjeru resursa fizičkog poslužitelja

```
1. ulaz (raspoloživi_cpu)
2. ulaz (raspoloživi_ram)
3. novi_ukupni_dodijeljeni_cpu := ukupni_dodijeljeni_cpu - cpu_poslužitelja + promjena_cpu
4. novi_ukupni_dodijeljeni_ram := ukupni_dodijeljeni_ram - ram_poslužitelja + promjena_ram
5. ako je (novi_ukupni_dodijeljeni_cpu > raspoloživi_cpu) onda
6.     izlaz („Ukupni dodijeljeni resursi procesora iznose više od raspoloživih Resursa procesora fizičkog poslužitelja!“)
7.     izlaz („Preskače se dodjeljivanje resursa procesora…“)
8.     novi_ukupni_dodijeljeni_cpu := 0
9. inače
10.    izlaz („Ukupni dodijeljeni resursi procesora ne iznose više od raspoloživih Resursa procesora fizičkog poslužitelja!“)
11.    cpu_poslužitelja := cpu_poslužitelja + promjena_cpu
12. ako je (novi_ukupni_dodijeljeni_ram > raspoloživi_ram) onda
13.    izlaz („Ukupna dodijeljena memorija iznosi više od raspoložive memorije fizičkog poslužitelja!“)
14.    izlaz („Preskače se dodjeljivanje memorije…“)
15.    novi_ukupni_dodijeljeni_ram := 0
16. inače
17.    izlaz („Ukupna dodijeljena memorija ne iznosi više od raspoložive memorije fizičkog poslužitelja“)
18.    ram_poslužitelja := ram_poslužitelja + promjena_ram
```

2. *Na razini virtualnog poslužitelja* - HOST agent se brine da virtualni poslužitelj ne ostane bez resursa uz pomoć parametara za minimalne dozvoljene vrijednosti (*cpu_min* i *mem_min*). Ovo je relativno jednostavan postupak provjere novih vrijednosti resursa virtualnog poslužitelja, tj. nove vrijednosti resursa procesora ili memorije virtualnog poslužitelja. Ovaj postupak je opisan na slici 7.22. gdje se upotrebom selekcija provjerava da li nove vrijednosti resursa prelaze minimalnu dozvoljenu vrijednost. U tom slučaju, kao nova vrijednost resursa koriste se učitane minimalne dozvoljene vrijednosti resursa.

```
1. ako je (cpu_nova_vrijednost < cpu_min) onda
2.     cpu_nova_vrijednost := cpu_min
3. izlaz („Nova količina Resursa procesora: “)
4. izlaz (cpu_nova_vrijednost)
5. ako je (ram_nova_vrijednost < ram_min) onda
6.     ram_nova_vrijednost := ram_min
7. izlaz („Nova količina radne memorije poslužitelja: „)
8. izlaz (ram_nova_vrijednost)
```

Slika 7.22. - Postupak za provjeru resursa virtualnog poslužitelja

- *Provjera konzistentnosti u radu poslužitelja* - predstavlja jedan od najvažnijih mehanizama novog modela jer provjerava da nije došlo do zastoja ili greške u radu glavnog servisa virtualnog poslužitelj čiji su resursi izmijenjeni. Ovo podrazumijeva da svi pristupi, dodavanja ili oduzimanja resursa ostave programski sustav u konzistentnom stanju. Konzistentnost sustava predstavlja neprekidan i ispravan rad sustava [89]. Riječ je o jednostavnoj provjeri odaziva virtualnog poslužitelja nakon svake promjene resursa. Na taj način se nastoji spriječiti svaka mogućnost nastanka prekida u radu poslužitelja, tj. nastoji se očuvati konzistentnost rada poslužitelja.
- *Mogućnost upozorenja* - predstavljaju kontrolne mehanizme koje šalju obavijesti (elektroničku poštu) ako je:
    1. došlo do pogreške ili nekog prekida u radu sustava i
    2. fizički poslužitelj nema slobodnih računalnih resursa (CPU i memorija) za dodjelu svojim virtualnim poslužiteljima.

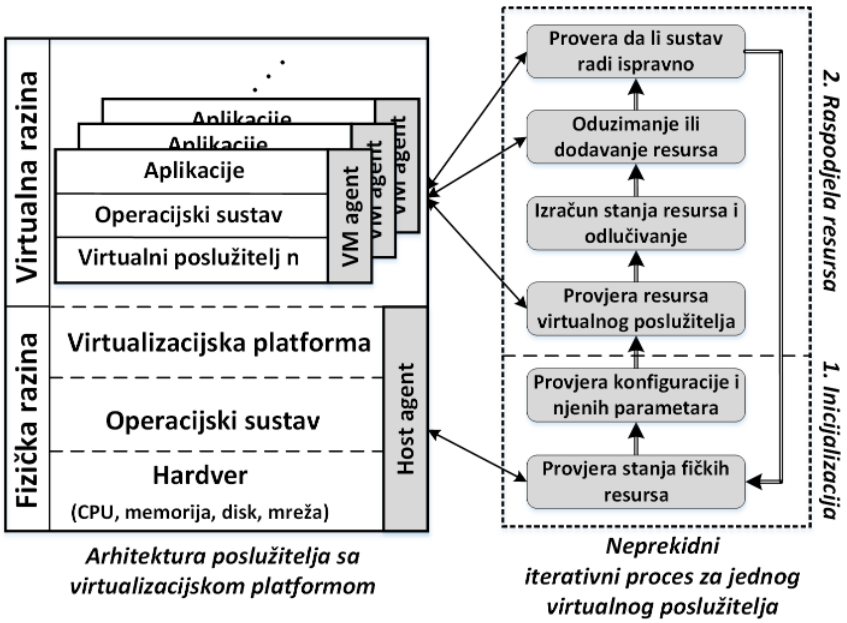
  Kao što je već spomenuto, kontrolni mehanizam putem elektroničke pošte šalje obavijest ako je došlo do pogreške, zastoja i nekog drugog mogućeg problema. Obavijesti se šalju kako bi se na vrijeme korisnik upozorio te poduzeo odgovarajuće mjere. Prilikom slanja obavijesti, odnosno elektroničke pošte, potrebne su određene informacije poput naziva virtualnog poslužitelja i adrese na koju obavijesti trebaju stizati. Jedan od mogućih slučajeva kad se šalje obavijest prikazan je na slici 7.23. pomoću pseudo jezika.

```
1.  poslužitelj_ime := ime virtualnog poslužitelja
2.  status_poslužitelja := odaziv virtualnog poslužitelja
3.  ako je (status_poslužitelja = radi) onda
4.      cpu_poslužitelja := trenutni resursi procesora poslužitelja
5.      ram_poslužitelja := trenutna memorija poslužitelja
6.  inače
7.      izlaz („Dobivanje informacija o poslužitelju neuspješno!“)
8.      pošalji e-mail („Dobivanje informacija o virutalnom
    poslužitelju“, poslužitelj_ime, „ neuspješno!“)
```

Slika 7.23. - Postupak za kontrolni mehanizam koji šalje obavijesti

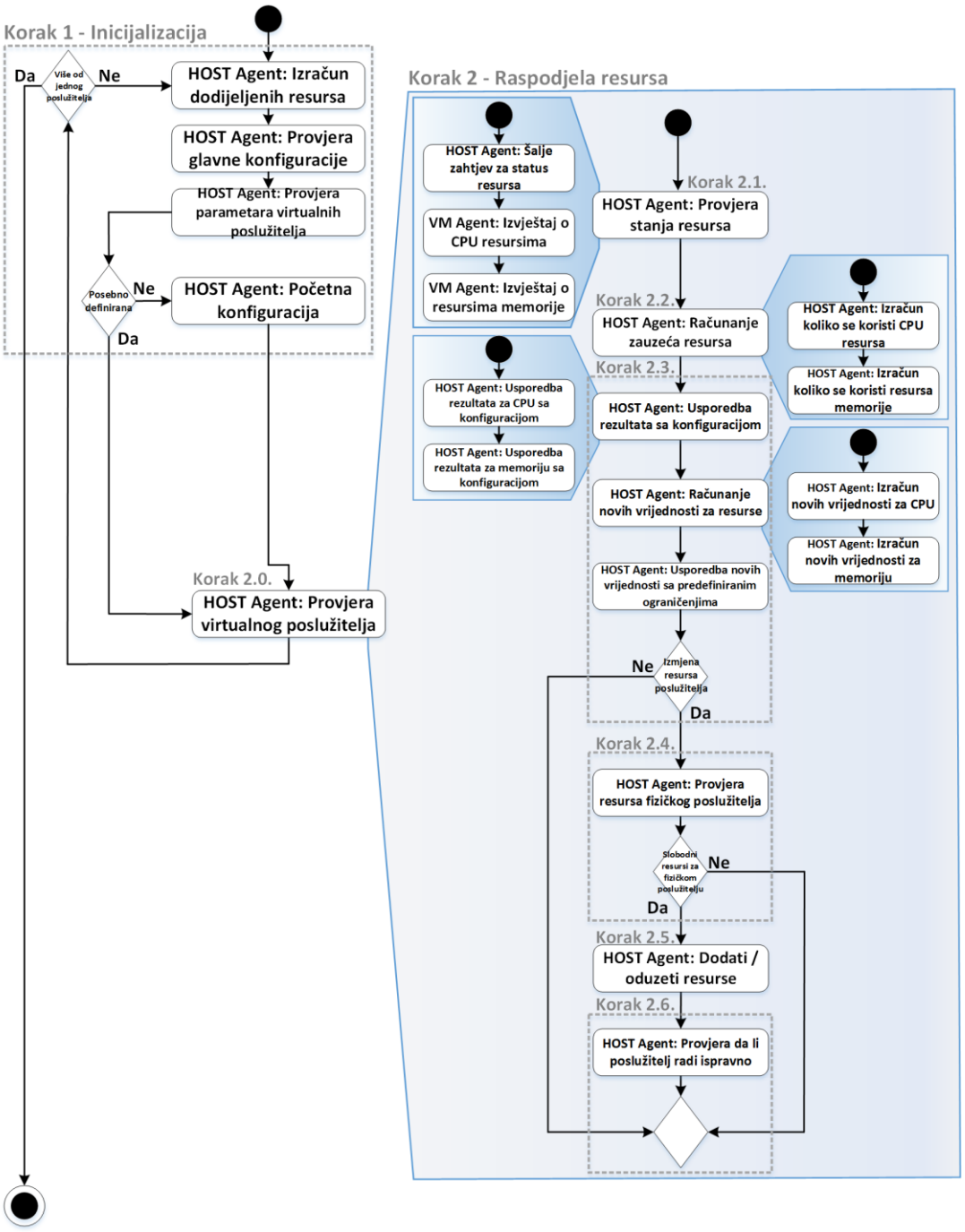
7.4 Prikaz rješenja

Ovaj korak istraživačke paradigme znanost o dizajnu predstavlja prikaz rješenja odnosno modela na konkretnom primjeru. Na slici 7.24. prikazana je slojevita arhitektura novog modela koja je kasnije formalno opisana pomoću dijagrama aktivnosti kako bi se bolje razumjelo ponašanje novog modela.



Slika 7.24. - Slojevita arhitektura novog modela

Na slici 7.24. je prikazan jedan fizički poslužitelj s virtualizacijskom platformom (isti princip vrijedi i za više fizičkih poslužitelja) na kojem se nalazi agent kao važna komponenta novog modela. Na slici 7.25. prikazan je dijagram aktivnosti predloženog modela.



Slika 7.25. - Dijagram aktivnosti novog modela

Cijeli proces s prethodne slike se izvršava prema unaprijed definiranim vremenskim iteracijama i sastoji se od dva glavna koraka:

1. ***Inicijalizacija*** - proces započinje provjerom koliko je ukupno resursa dodijeljeno virtualnim poslužiteljima odnosno koliko resursa ima na raspolaganju fizički poslužitelj. Nakon toga HOST agent provjerava glavnu konfiguracijsku datoteku i iz nje očitava:
    1. popis svih poslužitelja,
    2. zadane početne parametre za donju i gornju granicu CPU i memorije (ako su definirani) i

3. profile resursa za CPU i memoriju (ako su definirani).

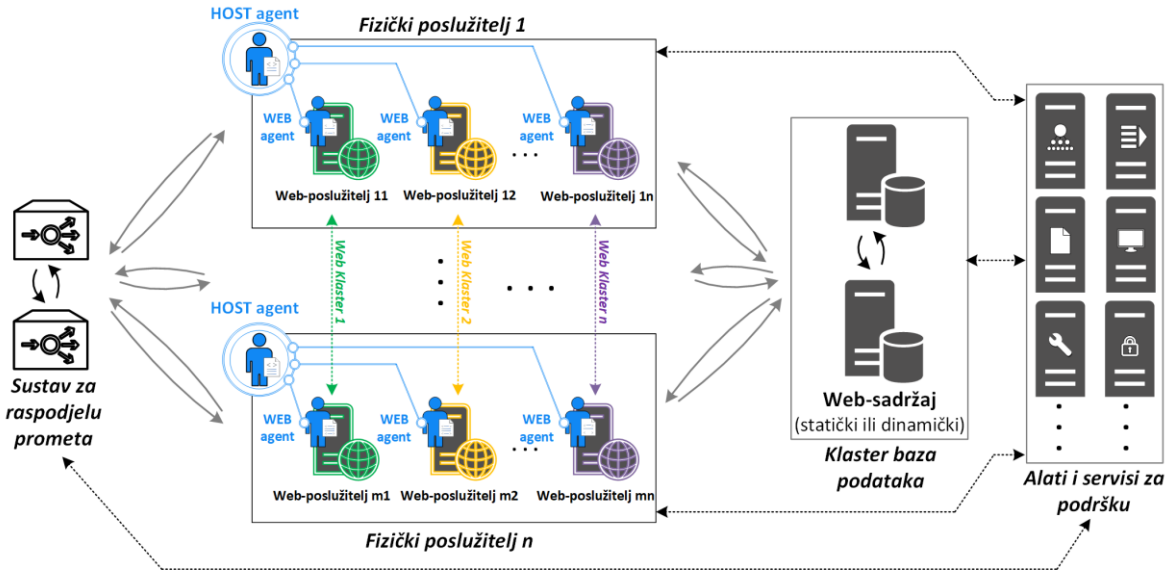
U ovom koraku HOST agent slijedno provjerava jedan po jedan poslužitelj na način da dohvati njegovo ime i pripadajuću konfiguraciju. HOST agent prvo provjerava da li poslužitelj ima posebno definirane parametre, ako ne, onda koristi početno zadane vrijednosti glavne konfiguracije.

2. ***Raspodjela resursa*** - nakon što se očitaju konfiguracijski parametri slijedi proces od šest koraka koji se odnosi samo na jednog virtualnog poslužitelja što jamči da proces neće utjecati i narušavati rad drugih poslužitelja. Ovo rezultira neprekidnim radom cijelog sustava. Cijeli proces se sastoji od sljedećih koraka:
    1. ***Provjera resursa na samom virtualnom poslužitelju*** - računalni resursi koji se provjeravaju su: memorija s nasumičnim pristupom (engl. *Random Access Memory, RAM*) koja je izražena u gigabajtima (GB) i procesor, odnosno središnje jedinice za obradu (engl. *Central Processing Unit, CPU*) koja je izražena u broju njegovih jezgri (engl. *cores*). Opterećenje procesora je izraženo u postotcima. U ovom koraku VM agent prvo provjeri stanje resursa, odnosno prosječno opterećenje za procesor i postotak slobodne memorije (neiskorištena memorija zajedno s predmemorijom). Nakon toga se pomoću *Server Daemona* (xinetd) navedene vrijednosti dalje proslijede na HTTP port (u ovom slučaju proizvoljni port broj 9299) HOST agentu kako bi ih on očitao i dalje obradio.
    2. ***Izračun slobodnih resursa na virtualnom poslužitelju*** - u ovom koraku HOST agent uspoređuje dohvaćene podatke o stanju resursa virtualnog poslužitelja s vrijednostima koji su dodijeljeni tom poslužitelju, te na temelju toga računa iskoristivost za procesor i memoriju.
    3. ***Usporedba dobivenih rezultata s početnom konfiguracijom i odlučivanje da li će se resursi oduzeti ili dodati*** - u ovom koraku se obavlja provjera i usporedba s donjom i gornjom granicom za procesor i memoriju koje su definirane u konfiguraciji. Na temelju toga se obavlja odlučivanje da li će se i koliko će se resursa oduzeti odnosno dodati.
    4. ***Provjera resursa fizičkog poslužitelja*** - prije oduzimanja ili dodjele resursa prvo se provjere resursi fizičkog poslužitelja kako ne bih došlo do opterećenja cijelog sustava. Ako su potrebni resursi, a fizički poslužitelj ih ima, onda se obavlja dodjela, u suprotnom se nastavlja s provjerom sljedećeg virtualnog poslužitelja (cijeli postupak kreće od početka).

5. ***Postupak dodavanja ili oduzimanja resursa*** - u ovom koraku se resursi dodaju ili oduzimaju ovisno o opterećenju virtualnog poslužitelja. Postupak oduzimanja odnosno dodavanja resursa se obavlja istovremeno (paralelno) za procesor i memoriju. Drugim riječima, mogući su scenariji da je potrebno procesor (odnosno broj jezgri) dodati, a memoriju oduzeti i obratno.
6. ***Provjera da li virtualni poslužitelj radi*** - Nakon što se obavi proces dodavanja ili oduzimanja resursa obavlja se provjera konzistentnosti u radu poslužitelja, odnosno je li došlo do zastoja ili greške u radu glavnog servisa virtualnog poslužitelj čiji su resursi izmijenjeni.

Nakon što se obavi drugi korak slijedi provjera glavne konfiguracije odnosno provjera da li ima još virtualnih poslužitelja. Ako ih ima, cijeli postupak se ponavlja od početka u suprotnom se završava i čeka unaprijed definirani vremenski interval kad će se ponovo pokrenuti. Rezultati su pokazali da cijeli prethodno navedeni proces po jednom virtualnom poslužitelju traje do tri sekunde (potvrđeno u poglavlju 7.5.7, npr Slika 7.54.).

Nakon detaljnog prikaza i opisa slijedi primjena odnosno validacija modela na konkretnom primjeru. U ovom slučaju to su Web-poslužitelji gdje je problem neučinkovitog iskorištenja postojećih računalnih resursa i prepoznat. Slika 7.26. predstavlja grafički prikaz cijelog rješenja odnosno primjena modela na *n* Web-poslužitelja odnosno *n* Web-klastera.



Slika 7.26. - Prikaz novog rješenja na primjeru Web-poslužitelja

Model je dizajniran na način da se ne narušavaju postojeći zahtjevi i arhitektura Web-farme. Drugim riječima i dalje je moguće skaliranje svih komponenti sustava ako je to potrebno.

Kao što je već prethodno spomenuto proces validacije je izvršen na KVM virtualizacijskoj platformi i poslužiteljima s Linux (CentOS) operacijskim sustavom. Kako bi

se izvršila provjera modela razvijena je aplikacija koja se sastoji od dvije skripte (glavna i pomoćna). Rješenje se sastoji svih komponenti modela koje su opisane u poglavlju 7.3.1. Glavna skripta (Host agent, Prilog 1.) se nalazi na svim fizičkim poslužiteljima, a pomoćna skripta na svim virtualnim poslužiteljima (VM agent, Prilog 2). Skripta na fizičkim poslužiteljima se pokreće prema proizvoljnom vremenskom intervalu koji je unaprijed definiran. U ovom slučaju za pokretanje glavne skripte koristi se *cron* servis koji je dio CentOS operacijskog sustava i služi za definiranje periodičnog izvršavanja određenih zadataka. Jednom kad se glavna skripta pokrene ona izvršava dva glavna koraka koji su opisani na slici 7.25. Prvi korak predstavlja inicijalizaciju, odnosno provjera koliko je ukupno resursa dodijeljeno virtualnim poslužiteljima odnosno koliko resursa ima na raspolaganju fizički poslužitelj. Nakon toga slijedi provjerava glavne konfiguracijske (vm_config.csv, Prilog 3) iz koje se očitava popis svih poslužitelja i njihovih pripadajući parametara koji su opisani u poglavlju 7.3. U drugom koraku slijedi proces raspodjele resursa koji se sastoji od šest pod koraka, a oni što su:

1. provjera resursa na samom virtualnom poslužitelju,
2. izračun slobodnih resursa na virtualnom poslužitelju,
3. usporedba dobivenih rezultata s početnom konfiguracijom i odlučivanje da li će se resursi oduzeti ili dodati,
4. provjera resursa fizičkog poslužitelja,
5. postupak dodavanja ili oduzimanja resursa i
6. provjera da li virtualni poslužitelj radi.

Tijekom ovog procesa glavna skripta dohvaća informacije o stanju resursa virtualnih poslužitelja koje provjerava pomoćna skripta koja se nalazi na njima. Komunikacija između glavne i pomoćne skripte se vrši pomoću Server Daemona (xinetd) koji informacije glavnoj skripti prosljeđuje na HTTP port.

7.5 Vrednovanje

Proces vrednovanja predstavlja jedan od najvažniji koraka prema istraživačkoj paradigmi znanosti o dizajnu. Prilikom vrednovanja novog modela korišten je Microsoftov priručnik za testiranje Web-aplikacija koji se sastoji od skupa smjernica i preporuka napisanih od strane stručnjaka iz navedenog područja [94]. U priručniku su opisane smjernice kako strukturno izvršiti cjelokupni proces testiranja počevši od osnovnih koraka kao što su prepoznavanje testnog uzorak pa do načina prikupljanja rezultata i pisanja izvještaja. Također se navode mnogobrojne preporuke kao što su: smjernice za prepoznavanje uskih grla i potencijalnih rizika u procesu testiranja, način utvrđivanja prema zadanim ciljevima, način dizajniranja testova i testnog okruženja, preporuke za odabir testnih alata i sl. Tablica 7.2. predstavlja popis

navedenih smjernica prema Microsoft priručniku za testiranje Web-aplikacija koje su podijeljene u sedam slijednih aktivnosti. Svaka aktivnost ima određene ulazne i izlazne elemente.

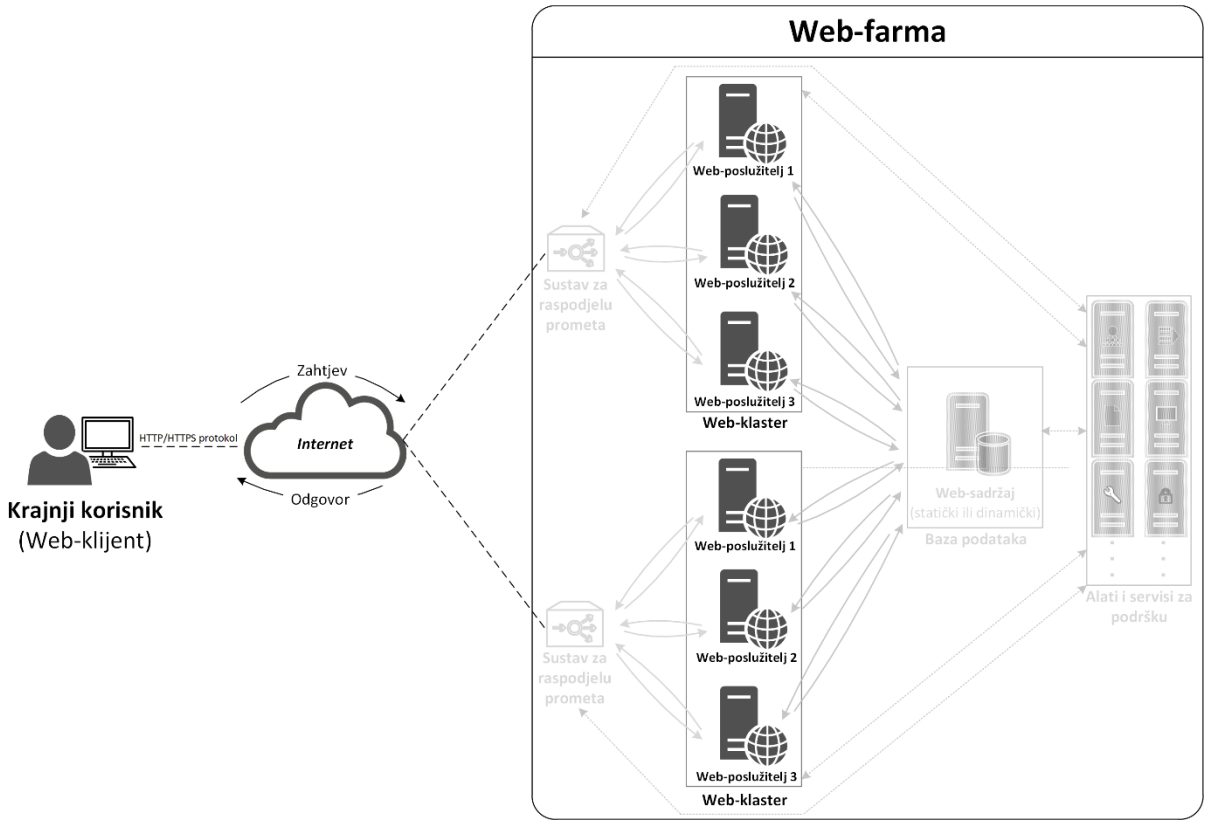
Tablica 7.2. - Sažetak osnovnih aktivnosti prema Microsoftov priručniku za testiranje Web-aplikacija [94]

aktivnost	ulazni elementi	izlazni elementi
Aktivnost 1. - Određivanje testnog okruženja	• Logička i fizička arhitektura produkcijskog okruženja • Logička i fizička arhitektura testnog okruženja • Dostupni alati	• Usporedba produkcijskog i testnog okruženja • Utjecaj na okruženje • Odluka da li su potrebni dodatni alati
Aktivnost 2. - Određivanje kriterija prihvaćenja	• Očekivanja klijenata • Rizici koje treba ublažiti • Poslovni zahtjevi • Ugovorne obveze	• Kriteriji uspješnosti testiranja • Ciljevi i zahtjevi izvedbe • Ključna područja istraživanja • Ključni pokazatelji uspješnosti • Ključni pokazatelji poslovanja
Aktivnost 3. - Planiranje i dizajn testova	• Dostupne značajke i / ili komponente aplikacije • Scenariji korištenja aplikacija • Jedinični testovi • Kriteriji prihvaćanja uspješnosti	• Konceptualna strategija • Preduvjeti za izvršenje testiranja • Potrebni alati i resursi • Modeli korištenja aplikacija koji se simuliraju • Testni podaci potrebni za provođenje testova • Testovi spremni za implementaciju
Aktivnost 4. - Konfiguracija testnog okruženja	• Konceptualna strategija • Dostupni alati • Dizajnirani testovi	• Konfigurirani alati za stvaranje opterećenja i alati za nadzor resursa • Okruženje spremno za testiranje performansi
Aktivnost 5. - Ostvarivanje testnog dizajna	• Konceptualna strategija • Dostupni alati / okruženja • Dostupne aplikacijske značajke i / ili komponente • Dizajnirani testovi	• Provjereni i izvršni testovi • Validirani nadzor resursa • Zbirka validiranih podataka
Aktivnost 6. - Provesti testiranje	• Plan izvršenja zadataka • Dostupni alati / okruženja • Dostupne aplikacijske značajke i / ili komponente • Provjereni i izvršni testovi	• Provjera dobivenih rezultata
Aktivnost 7. - Analizirati rezultate, izvješća i ponoviti testiranje	• Rezultati izvršenja zadatka • Kriterij prihvaćanja • Rizici i potencijalni problemi	• Analiza rezultata • Preporuke • Izvještaji

U nastavku rada slijedi detaljni opis svih sedam aktivnosti zajedno sa svojim ulaznim i izlaznim elementima koji su propisani od strane Microsoft priručnika.

7.5.1 Aktivnost 1. - Određivanje testnog okruženja

U prvoj aktivnosti potrebno je proučiti stvarni sustav odnosno produkcijsko okruženje zajedno sa svim njegovim elementima kao što su: hardver, mreža, alati, softver, vanjski čimbenici i sl. Nakon prikupljenih rezultata slijedi dizajniranje reprezentativnog testnog okruženje. Cijelo rješenje mora biti jasno opisano kako bi se ograničenja i potencijalni problemi prilikom testiranja prepoznali čim prije. Slika 7.27. prikazuje arhitekturu testnog okruženja nad kojem su provedeni eksperimenti odnosno validacija modela.



Slika 7.27. - Arhitektura testnog okruženja nad kojem su provedeni eksperimenti

Validacija modela je izvršena na primjeru Web-poslužitelja zbog toga su na slici 7.27. pojedini dijelovi Web-farme (sustavi za raspodjelu prometa, baza podataka, alati i servisi za podršku) označeni svijetlije.

Kao što je već spomenuto jedan od glavnih ciljeva ovog istraživanja je to da se koriste rješenja otvorenog tipa, ali i da se uradi detaljni opis cjelokupnog okruženje nad kojim su provedena ova istraživanja kako bi se ispitivanja mogla što jednostavnije ponoviti u svrhu daljnjeg istraživanja i unapređenja. Osim metoda i tehnika koje su navedene u poglavlju 7.3. u ovom istraživanju koriste se sljedeći operacijski sustav i programska rješenja:

- Operacijski sustav CentOS 6.7 (Linux) i virtualizacijska platforma Foreman Version 1.9.1 s Libvirt (virsh) 0.10.2
- Baza podataka: MySQL Ver 14.14 Distrib 5.6.12
- Sustav za raspodjelu prometa (HTTP/HTTPS): HAproxy version 1.5.4
- Web-poslužitelj: apache/2.2.15 i nginx/1.8.0
- Web-stranice: WordPress Version 4.3 (Content Management System)
- *Server Daemon*: xinetd Version 2.3.14 (za prikaz opterećenja poslužitelja)
- Servisi: lsyncd 2.1.5 (za sinkronizaciju podataka između Web-poslužitelja)
- Skripti jezici: pseudojezik, PHP 5.6.13 i Bash Shell Scripting

Nakon logičkog prikaza arhitekture i svih njenih komponenti slijedi fizički opis. Testno okruženje se sastoji od tri fizička poslužitelja (HP ProLiant DL360 G7) koja se zovu: host1, host2 i host3. Sva tri fizička poslužitelja imaju iste specifikacije, a one su:

- dva procesora (svaki procesor s četiri jezgre),
- 32 GB memorije (8 memorijskih modula po 4 GB) i
- dva lokalna čvrsta diska po 146 GB (konfigurirana u RAID 1).

Na fizičkim poslužitelja se nalazi ukupno devet virtualnih poslužitelja. Tablica 7.3. predstavlja popis virtualnih poslužitelja i njihovih početnih karakteristika.

Tablica 7.3. - Popis virtualnih poslužitelja i njihovih početnih karakteristika

naziv virtualnog poslužitelja	fizički poslužitelj na kojem se nalazi	CPU (broj jezgri)	memorija (GB)	lokalni čvrsti disk (GB)
dt-lb1	host1	1	1	10
dt-lb2	host1	1	1	10
dt-web1	host2	1	0.6	10
dt-web2	host2	1	0.6	10
dt-web3	host3	1	0.6	10
dt-web4	host3	1	0.6	10
dt-web5	host3	1	0.6	10
dt-web6	host2	1	0.6	10
dt-db1	host1	4	4	20

U ovom koraku je također važno prepoznati i popisati sve alate koji su se koristili u procesu validiranja novog modela. Eksperimenti su izvršeni uz pomoć dva alata sa simulaciju zahtjeva:

- *ApacheBench* (inačica 2.4.3.) - omogućava brze i jednostavne testove bez naprednih mogućnosti. Alat je korišten u prvim inačicama aplikacije kako bi se provjerile osnovne mogućnosti.
- *Apache JMeter* (inačica 2.11.) - omogućava napredna ispitivanja (paralelna) između više Web-stranica koje se nalaze na više Web-klastera. Alat je korišten prilikom provjere hipoteza.

Osim navedenih alata koji omogućavaju simulaciju zahtjeva, korišteni su i alati za nadzor sustava i evaluaciju rezultata:

- *Observium* (inačica 0.15.12.7231) - nadzor računalnih resursa i opterećenja servisa.
- *Munin* (inačica 2.0.25) - nadzor računalnih resursa i opterećenja servisa.
- *Apache OpenOffice* (inačica 3) - prikaz i usporedbu dobivenih rezultata.

Svi prethodno navedeni alati koji su korišteni u procesu vrednovanja novog modela se nalaze na posebnim poslužiteljima koji nisu dio testnog okruženje kako bi se postigli što

vjerodostojniji rezultati. Drugim riječima, cilj je bio da poslužitelji koji su dio testnog okruženja budu opterećeni samo korisničkim zahtjevima i aplikacijama potrebnim za njihovu obradu.

7.5.2 Aktivnost 2. - Određivanje kriterija prihvaćanja

Nakon proučavanja i određivanje svih elemenata testnog okruženja potrebno je definirati ciljeve odnosno kriterije prihvaćanja koji nastaju analizom: poslovnih zahtjeva, ugovornih obveza, očekivanja klijenata i sl. U praksi to su najčešće parametri kao što su: vrijeme odziva, propusnost, iskoristivost resursa i sl. Glavni parametar u ovom istraživanju predstavlja iskoristivost postojećih računalni resursa. Kako bi se zadani ciljevi postigli definirane su tri hipoteze čijim bi potvrđivanjem postigli kriterije prihvaćanja. Provjera hipoteza je izvršen u tri faze, a one su:

- ***Faza I*** - *validacija prve hipoteze (metoda eksperiment)*: Kako bi se provjerila prva hipoteza razvijena je aplikacija po uzoru na predloženi model. Nakon toga izvršeni su eksperimenti za simulaciju korisničkih zahtjeva kojima se opterećuju računalni resursi poslužitelja. Svrha ovog testa je provjera je li moguće dodavanje i oduzimanje računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja. Eksperiment je obavljen uz pomoć alata sa simulaciju zahtjeva kao što su: *ApacheBench* i *Apache JMeter*. Testovi su definirani na temelju već spomenutih preliminarnih istraživanja u kojima su korišteni podaci nekoliko IT poduzeća. Na temelju podatka kao što je broj simultanih korisnika u određenom vremenu definirani su testovi koji rezultiraju opterećenjem računalnih resursa (CPU i memorije). Testovi su podijeljeni u grupe kako bi da provjerile sljedeće tri mogućnosti:
    1. ***Test 1: Dodavanje i oduzimanje računalnih resursa*** - odnosno linearno povećanje i smanjenje zahtjeva tj. opterećenja bez potrebe za ponovnim pokretanjem poslužitelja. Test je koncipiran na način da se svake minute poveća broj simultanih (istovremenih) korisnika za jedan. Nakon 25 simultanih korisnika njihov broj se počne smanjivati za jedan svake minute dok ne dođe do nule. Test ukupno traje 50 minuta i u tom vremenu su vidljive promjene kao što su povećanje i smanjenje opterećenja računalnih resursa (CPU i memorija) Web-poslužitelja. Kako bi navedena količina korisničkih zahtjeva u ovom testu čim prije opteretila Web-poslužitelja potrebno je na početku testa imati što manje računalnih resursa potrebnih za rad. U ovom slučaju to je 1 CPU jezgra i 0.6 GB memorije.
    2. ***Test 2: Napredna raspodjela računalnih resursa*** - odnosno iznenadno povećanje i smanjenje velikog broja korisničkih zahtjeva bez potrebe za

ponovnim pokretanjem poslužitelja. Drugi test je za razliku od prvog dosta intenzivniji jer se u ovom slučaju broj simultanih korisnika naglo povećava tj. svake sekunde po jedan dok ne dođe do broja 100. U tom trenutku test se nastavlja izvršavati dodatnih 60 sekundi s 100 simultanih korisnika i onda se broj korisnika smanjuje za jedan svake sekunde dok ne dođe do nule. Test ukupno traje 260 sekundi. Kako bi se rezultati čim prije vidjeli, Web-poslužitelju je kao i u prvom testu na početku dodijeljen manji broj resursa (1 CPU jezgra i 0.6 GB memorije).

3. ***Test 3: Ispitivanje aplikacije (modela) nad više poslužitelja*** - provjera je li model skalabilan odnosno primjenjiv na više poslužitelja. U ovom slučaju provjera je izvršena na primjeru dva Web-klastera (jedan *nginx*, a drugi *apache*) od kojih svaki ima po tri Web-poslužitelja. Ovim se provjerava ne samo da je model primjenjiv na više Web-poslužitelja već i na više tehnologija (*apache i nginx*).

Danas gotovo svi operacijski sustavi imaju ostvarene mehanizme za provjeru koliko je sustav aktivan odnosno kad je izvršeno zadnje pokretanje sustava. Upravo se taj mehanizam koristi kako bi se provjerilo je li došlo do ponovnog pokretanja poslužitelja nakon raspodjele resursa.

- ***Faza II*** - *validacija druge hipoteze (metoda eksperiment)*: Primjena predloženog modela ne narušava konzistentnost u radu poslužitelja. U ovom slučaju potrebno je osigurati da nije došlo do zastoja ili greške u radu glavnog servisa (u ovom slučaju Web-poslužitelja). Kako bi se ovo postiglo novi model ima komponentu koja se zove kontrola sustava i ona se sastoji od nekoliko mehanizama koji su opisani u obliku pseudokoda što omogućuje njihovu primjenu neovisno o virtualizacijskoj platformi i programskom jeziku. U ovoj fazi je na temelju spomenutog pseudokoda ostvarena aplikaciju koja se koristiti za provjeru prve hipoteze. S obzirom da je validacija modela na primjeru Web-poslužitelja konzistentnost se provjerava pomoću HTTP status kodova. Odnosno nakon svake operacije nad resursima (pristup, dodavanje ili oduzimanje) obavlja se provjera Web-poslužitelja. Drugim riječima ako kontrola sustava od Web-poslužitelja dobije HTTP kod 200 onda konzistentnost sustava nije narušena, a ako dobije npr. kod 500 onda je došlo do greške u radu Web-poslužitelja. Kako bi se provjerila konzistentnost sustava ponovljeni su testovi iz prve faze jer oni u sebi imaju sve tri važne operacije nad resursima: pristup, dodavanje i oduzimanje.
- ***Faza III*** - *validacija treće hipoteze (metode eksperiment i uspoređivanje)*: Zadnja faza predstavlja provjeru treće hipoteze odnosno da li primjena novog modela povećava

iskoristivost postojećih računalnih resursa s obzirom na prvobitno rješenje. Postupak validacije se sastoji od sljedećih koraka:

1. ***Analiza ponašanja stvarnih sustava*** (broj korisničkih zahtjeva i opterećenje resursa u određenom vremenu) - koriste se podaci nekoliko IT tvrtki koje pružaju usluge Web-udomljavanja.
2. ***Instalacija i konfiguracija testnog okruženja*** - napravljeno reprezentativno testno okruženje (Web-farma) po uzoru na postojeća rješenja koja se koriste u spomenutim tvrtkama.
3. ***Instalacija i konfiguracija alata*** - za nadzor opterećenja računalnih resursa i sustava koristi se *Observium* i *Munin*, a za simulaciju korisničkih zahtjeva *ApacheBench* i *Apache JMeter*.
4. ***Definiranje eksperimenta*** - na temelju prvog koraka definirani reprezentativni eksperimenti.
5. ***Provedba eksperimenta*** - uz pomoć alata za simulaciju korisničkih zahtjeva obavljeni eksperimenti.
6. ***Usporedba rezultata*** - uspoređeni dobiveni rezultati alata za nadzor opterećenja računalnih resursa s rezultatima iz stvarnih sustava (postotak iskoristivosti CPU i memorije). Odnosno obavljeno uspoređivanje s i bez primjene novog modela i utvrđeno da li se postojeći računalni resursi učinkovito iskorištavaju.

7.5.3 Aktivnost 3. - Planiranje i dizajn testova

Nakon određivanje kriterija prihvaćanja slijedi planiranje i dizajn testova koji sastoji od nekoliko koraka, a oni su: prepoznavanje ključnih scenarija, utvrditi varijabilnost između reprezentativnih uzorka i načina simulacije, definirati testne podatke i odrediti metrike za njihovo prikupljanje.

Glavna problematika ovog istraživanja zajedno s ključnim scenarijima je prepoznata i opisana na početku sedmog poglavlja. U navedenom poglavlju su predstavljeni rezultati analize stvarnih sustava (Tablica 7.1.) temeljem kojih je definiran reprezentativni uzorak, ali i dizajnirani testovi koji su prikazani u narednim aktivnostima. Utvrđeno je da je glavna varijabla broj korisničkih zahtjeva jer upravo njihovom obradom dolazi do zauzeća računalnih resursa koji su predmet ovog istraživanja.

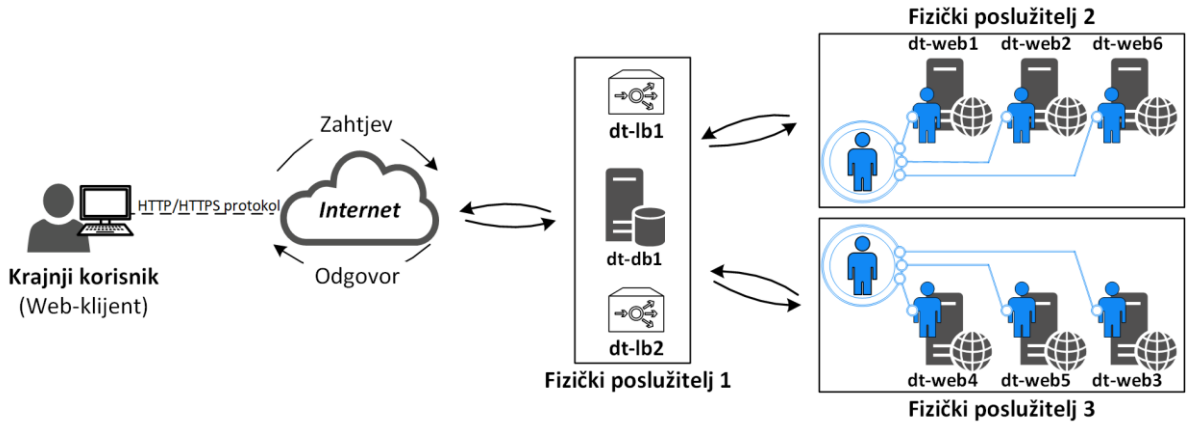
Simulaciju korisničkih zahtjeva je izvršena pomoću alata *ApacheBench* i *Apache JMeter*, a rezultati su prikupljani pomoću alata za nadzor opterećenja računalnih resursa kao što su *Observium* i *Munin*.

Jedno od IT poduzeća čiji su podaci korišteni u analizama stvarnih sustava ustupilo je potrebne resurse kako bi se ovo istraživanje provelo. Resursi su se koristili samo za potrebe istraživanja i bili su dostupni tijekom cijelog trajanja istraživanja. Navedeni resursi su iskorišteni za ostvarivanje testnog okruženja, ali i svih alata koji su bili potrebni tijekom ovog istraživanja.

7.5.4 Aktivnost 4. - Konfiguracija testnog okruženja

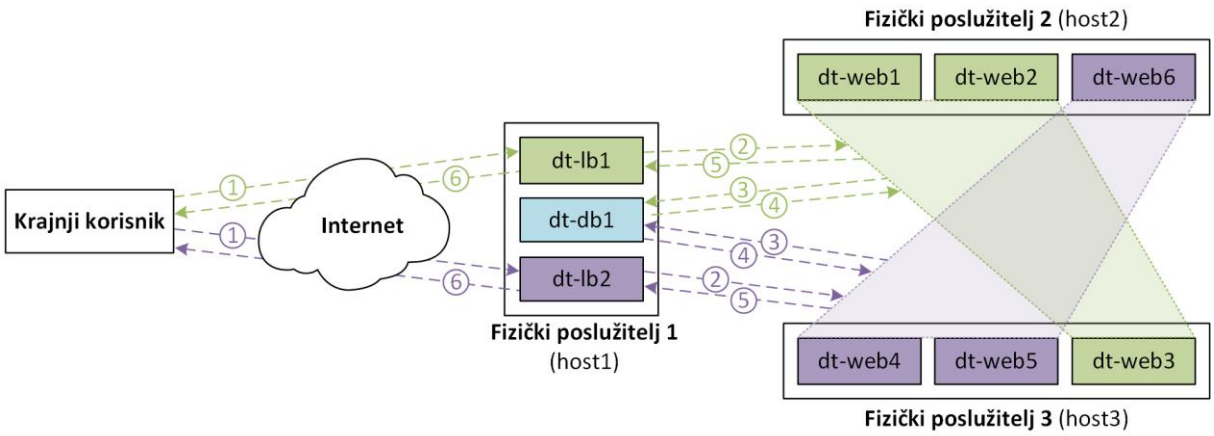
Ova aktivnost predstavlja ostvarivanje reprezentativnog testnog okruženje koje je definirano u prvoj aktivnosti. Osim testnog okruženja potrebno je pripremiti sve potrebne alate i resurse za izvršavanje kako bi se dizajnirani testovi mogli izvršiti.

Slika 7.28. predstavlja grafički prikaz cijelog testnog okruženja nad kojim je izvršeno ispitivanje i provjera novog modela za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa između Web-poslužitelja. Na slici se nalaze poslužitelji koji su popisani u tablici 7.3.



Slika 7.28. - Prikaz testnog okruženja nad kojim je ostvaren novi model

Cjelokupno okruženje je prikazano na slici 7.29., na kojoj se može vidjeti i raspored virtualnih poslužitelja ovisno o fizičkim poslužiteljima, ali i tijek zahtjeva koji rezultira opterećenjem resursa.



Slika 7.29. - Tijek podataka u testnom okruženju između krajnjeg korisnika i Web-stranice

U produkcijskim okruženjima Web-klasteri odnosno Web-poslužitelji su najčešće raspoređeni na više fizičkih poslužitelja kako bi se postigla veća razina dostupnosti sustava u slučaju da jedan od fizičkih poslužitelja postane nedostupan. Razlog zbog kojeg je isti koncept primijenjen i u ovom testnom okruženju gdje nema potrebe za visokom razinom dostupnosti je to što se želi provjeriti da li ima utjecaja na rad Web-klastera koji se nalazi na više fizičkih poslužitelja.

Na slici 7.29. označen je cijeli proces komunikacije koji se odvijanja između krajnjeg korisnika i Web-stranice koja se nalazi na jednom od Web-klastera. Proces je isti za oba Web-klastera i sastoji se od sljedećih koraka:

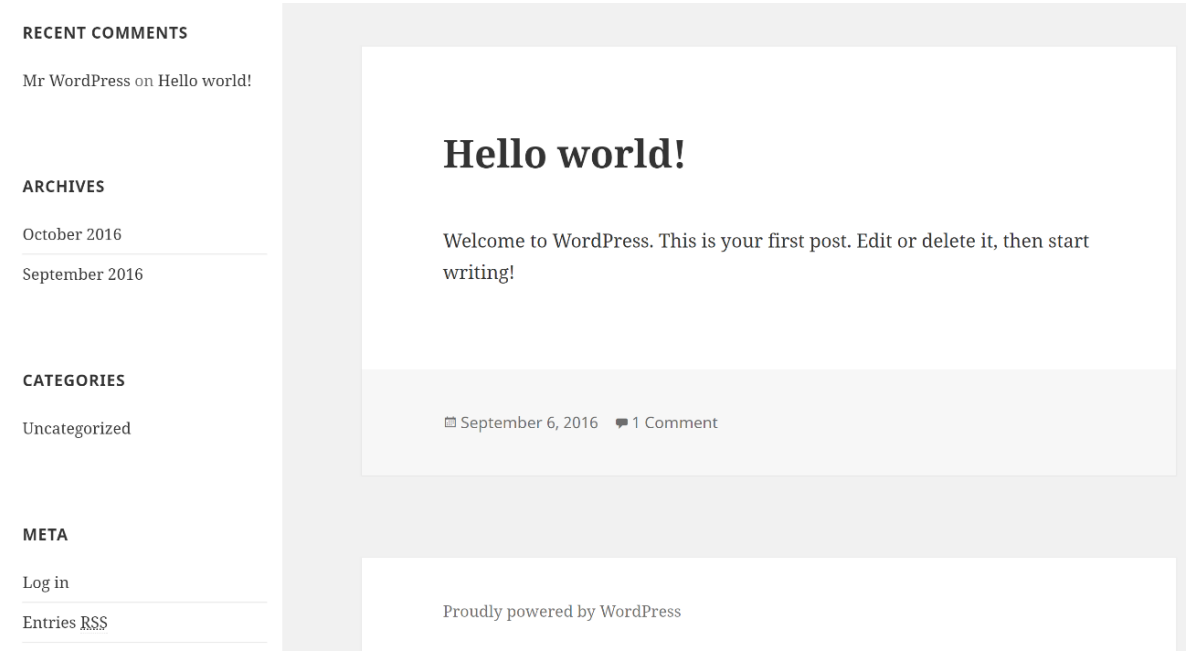
- *Korak 1.* - krajnji korisnik pomoću Internet preglednika izvršava HTTP/HTTPS upit na jednu od Web-stranica koji putem Internet poslužitelja uz pomoć registra i poslužitelja za domene dolazi do sustava za raspodjelu HTTP/HTTPS zahtjeva.
- *Korak 2.* - sustav za raspodjelu zahtjeva ovisno o algoritmu koji se koristi dalje zahtjeve prosljeđuje na Web-klaster odnosno na jedan od Web-poslužitelja na kojim se nalazi Web-stranica. U ovom slučaju je riječ o sustavu HAproxy i mogući algoritmi su: roundrobin, static-rr, leastconn, first, source, uri, rdp-cookie, url_param, hdr i sl. [95].
- *Korak 3. i 4.* - u ovom koraku Web-poslužitelj obrađuje zahtjeve te po potrebi obavlja dohvat podataka iz baze podataka.
- *Korak 5.* - nakon što se zahtjev obradi on biva proslijeđen nazad kroz sustav za raspodjelu prometa prema krajnjem korisniku.
- *Korak 6.* - isporuka zahtjeva krajnjem korisniku.

Kao što se vidi iz prethodne slike prisutna su dva Web-klastera. Svaki Web-klaster se sastoji od tri Web-poslužitelja s pripadajućim sustavnom za raspodjelu HTTP/HTTPS zahtjeva koji nisu redundantni jer je riječ o testnom okruženju. Osim njih nije redundantna niti baza podataka, jer ni ona nije glavni predmet ovog istraživanja.

Kako bi se potvrdilo da novi model ne ovisi o tehnologiji korištena su dva različita Web-poslužitelja. Prvi Web-klaster je *nginx* (dt-web1, dt-web2 i dt-web3) a drugi *apache* Web-klaster (dt-web4, dt-web5 i dt-web6).

Kako bi se ispitivanje novog modela moglo provjeriti potrebne su i Web-stranice. U ovom slučaju korišteno je 10 WordPress (inačica 4.3) instalacija po jednom Web-klasteru. WordPress predstavlja Web-sustav za upravljanje sadržajem i svaka instalacija se sastoji od više Web-stranica. Slika 7.30. predstavlja početnu Web-stranicu jedne WordPress instalacije koja je korištene u testu. Početna Web-stranica WordPress instalacije se sastoji od glavnog izbornika i

koji se nalazi s lijeve strane i Web-članaka s pripadajućim komentarima s desne strane. Glavni izbornik se sastoji od niza Web-poveznica na ostale Web-stranice koje pripadaju istoj WordPress instalaciji.



Slika 7.30. - Početni prikaz WordPress Web-stranice

Testovi su dizajnirani na način da se uvijek pozivala početna Web-stranica svake WordPress instalacije. Tablica 7.4. predstavlja popis Web-klastera i pripadajućih Web-domena (odnosno WordPress instalacija).

Tablica 7.4. - Popis Web-klastera i pripadajući Web-domena

naziv Web-klaster	vrsta Web-poslužitelja	Web-domena
apache-cluster	Apache	website1-apache-cluster.int.ch
		website2-apache-cluster.int.ch
		website3-apache-cluster.int.ch
		website4-apache-cluster.int.ch
		website5-apache-cluster.int.ch
		website6-apache-cluster.int.ch
		website7-apache-cluster.int.ch
		website8-apache-cluster.int.ch
		website9-apache-cluster.int.ch
		website10-apache-cluster.int.ch

nginx-cluster	Nginx	website1-nginx-cluster.int.ch
		website2-nginx-cluster.int.ch
		website3-nginx-cluster.int.ch
		website4-nginx-cluster.int.ch
		website5-nginx-cluster.int.ch
		website6-nginx-cluster.int.ch
		website7-nginx-cluster.int.ch
		website8-nginx-cluster.int.ch
		website9-nginx-cluster.int.ch
		website10-nginx-cluster.int.ch

Razlog zbog kojeg početne konfiguracije Web-poslužitelja i WordPress Web-stranica nisu dodatno mijenjane je to što se želi omogućiti što jednostavnija ponovljivost ispitivanja. Zbog toga je i cjelokupno testno rješenje detaljno opisano.

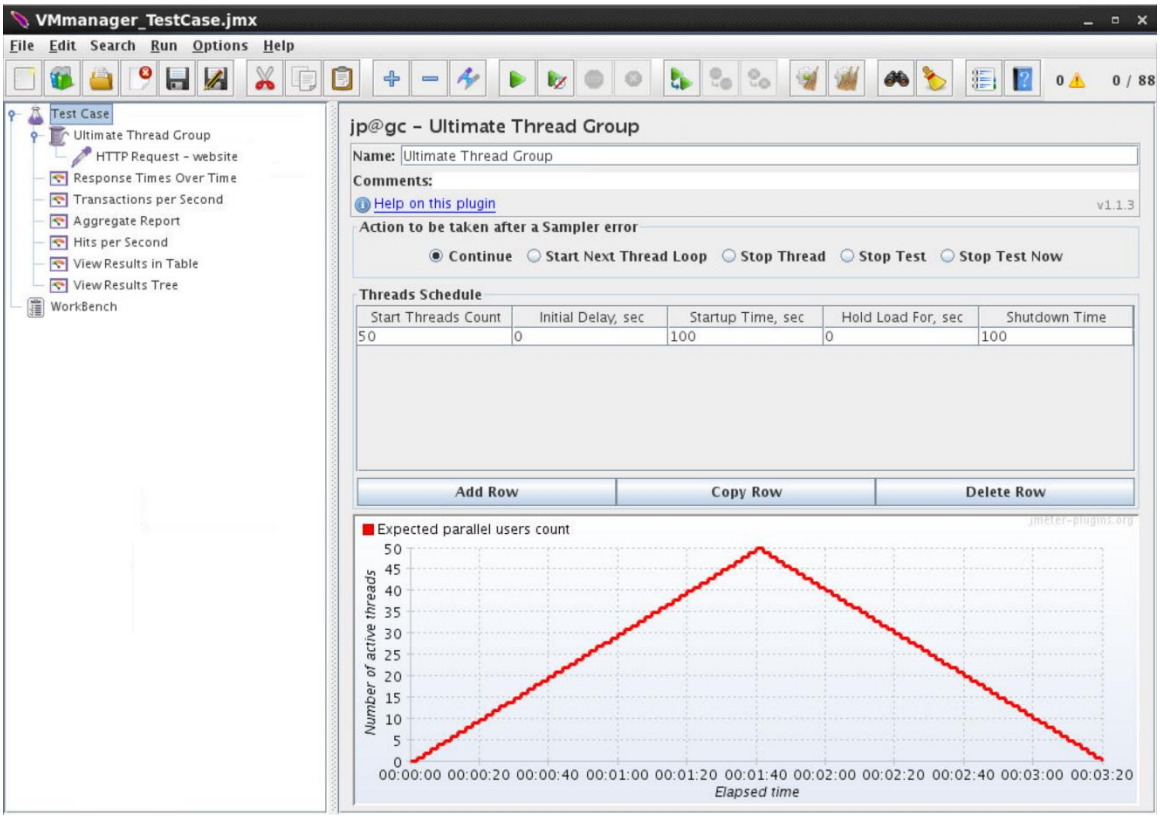
U ovoj aktivnosti je također potrebno osigurati da postoje instrumenti za nadzor resursa testnog okruženja. Već je prethodno spomenuto da će se koristiti alati *Observium* i *Munin*. Svi alati koji su korišteni u ovom istraživanju su otvorenog kako bi se omogućila lakša ponovljivost testiranja u svrhu daljnjih analiza i unapređenja modela.

7.5.5 Aktivnost 5. - Ostvarivanje testnog dizajna

Nakon konfiguracije testnog okruženja potrebno je razviti testove u skladu s definiranim testnim dizajnom. Svrha ove aktivnosti je da osigura da se testovi pravilno izvršavaju. Važno je napomenuti da se svi resursi koji su konfigurirani u prošloj aktivnosti koriste isključivo u svrhu ovog istraživanja. Fizički i virtualni poslužitelji su koristili isključivo programe i servise koji su opisani u prvoj aktivnosti, odnosno nije bilo drugih aplikacija koje bi dodatno opteretile računalne resurse. Takav pristup omogućava da dobiveni rezultati budu još više reprezentativniji.

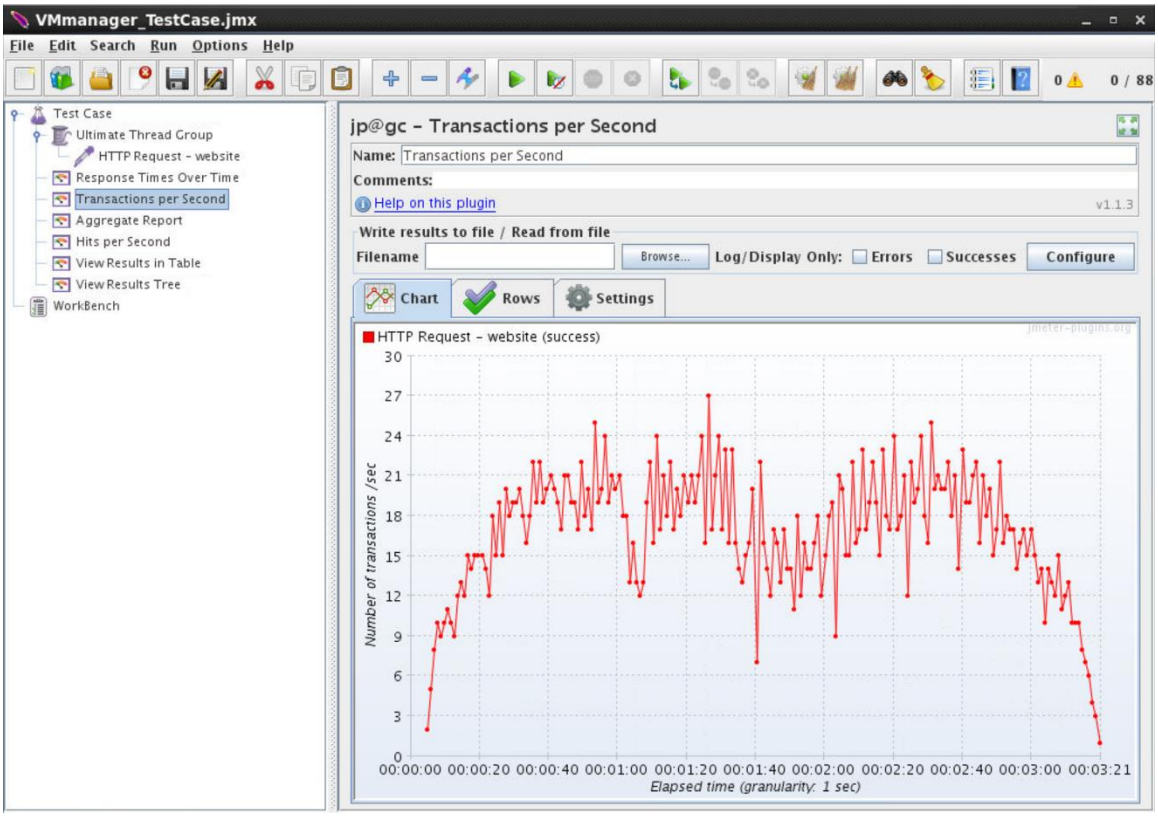
Ovo tvrdnja je potvrđena pomoću naredbe *htop* koja je spomenuta na početku sedmog poglavlja jer omogućava ispis svih aktivnih procesa i stanja računalnih resursa. Računalni resursi prije, tijekom i poslije testova su također praćeni pomoću alata za nadzor računalnih resursa (*Observium* i *Munin*), što je i prikazano u sljedećim aktivnostima. Tijekom cijelog istraživanja fizički i virtualni poslužitelji nisu ponovno pokretani niti gašeni.

U ovoj aktivnosti je važno provjeriti ispravnost testnog okruženja i alata koji su ostvareni u prijašnjim aktivnostima. Provjera se vrši pomoću glavne varijable koja je u ovom slučaju broj korisničkih zahtjeva. Kako bi izvršili simulaciju korisničkih zahtjeva korišteni su alati *ApacheBench* i *Apache JMeter*. Slika 7.31. predstavlja Prikaz postavki eksperimenta u alatu *Apache JMeter*.



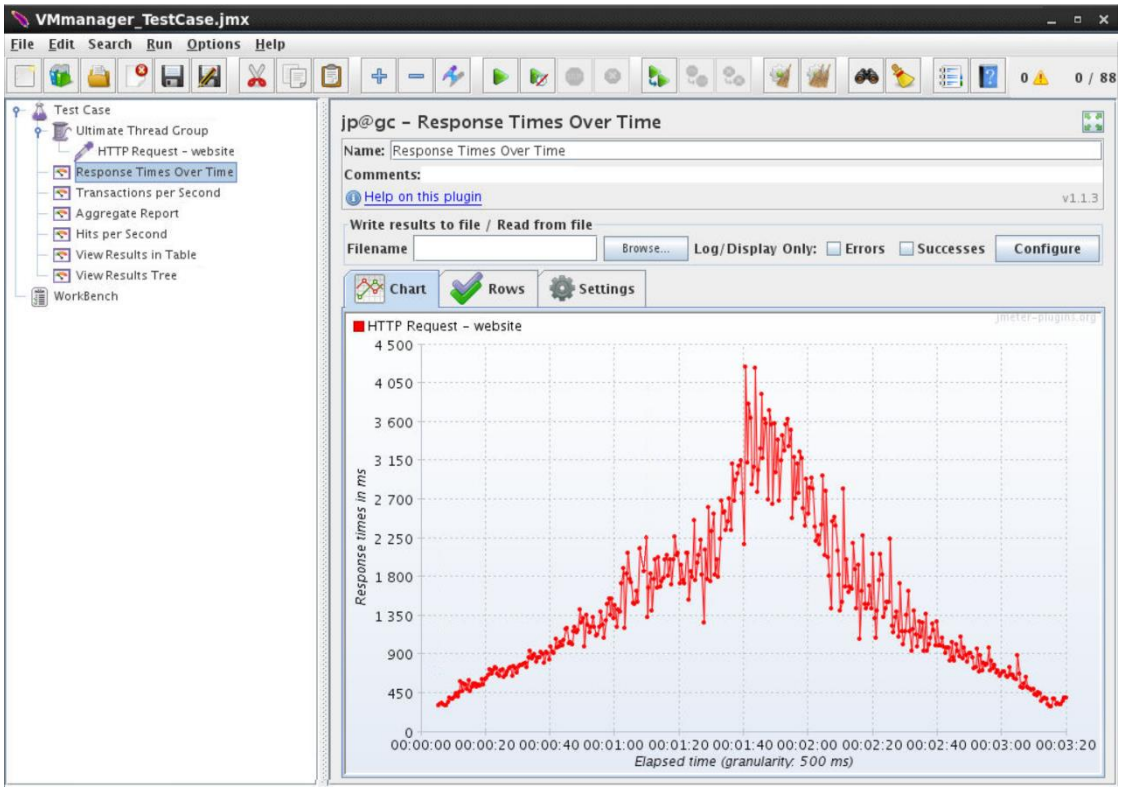
Slika 7.31. - Prikaz postavki eksperimenta u alatu *Apache JMeter*

Cilj ovog eksperimenta je provjeriti ispravnost testnog okruženja, odnosno da li broj korisničkih zahtjeva utječe na računalne resurse kao što je to slučaj u stvarnim sustavima. Ostali parametri u ovom testiranju nisu bitni (npr. vrijeme trajanja testa, vrijeme obrade zahtjeva, propusnost, ukupni broj korisničkih zahtjeva i sl.). Eksperiment ukupno traje 200 sekundi. Prvih 100 sekundi broj simultanih korisnika linearno raste do broja 50, a zatim linearno pada do nule idućih 100 sekundi. Slika 7.32. predstavlja grafički prikaz broja zahtjeva u sekundi za izvršeni eksperiment u alatu *Apache JMeter*. Za vizualizaciju je korišten graf *Transactions Per Seconds*.



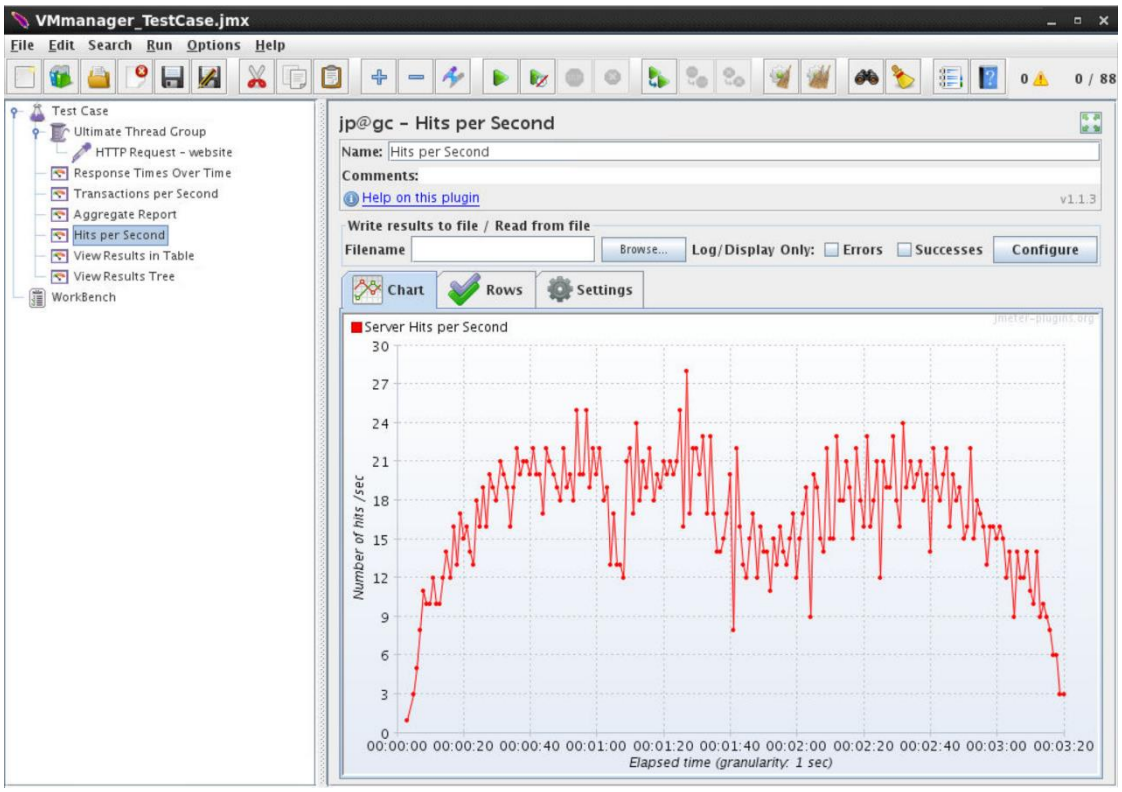
Slika 7.32. - Broj zahtjeva u sekundi za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta)

Iz prethodne slike se vidi kako broj korisničkih zahtjeva u vremenu prati parametre koji su definirani u testu (Slika 7.31.). Kako bi bili sigurni da je testno okruženje ispravno na sjedećim slikama će biti prikazani i ostali grafički prikazi iz alata. Slika 7.33. predstavlja grafički prikaz vremena odziva za izvršeni eksperiment u alatu *Apache JMeter*. Za vizualizaciju je korišten graf *Response Times Over Time*.



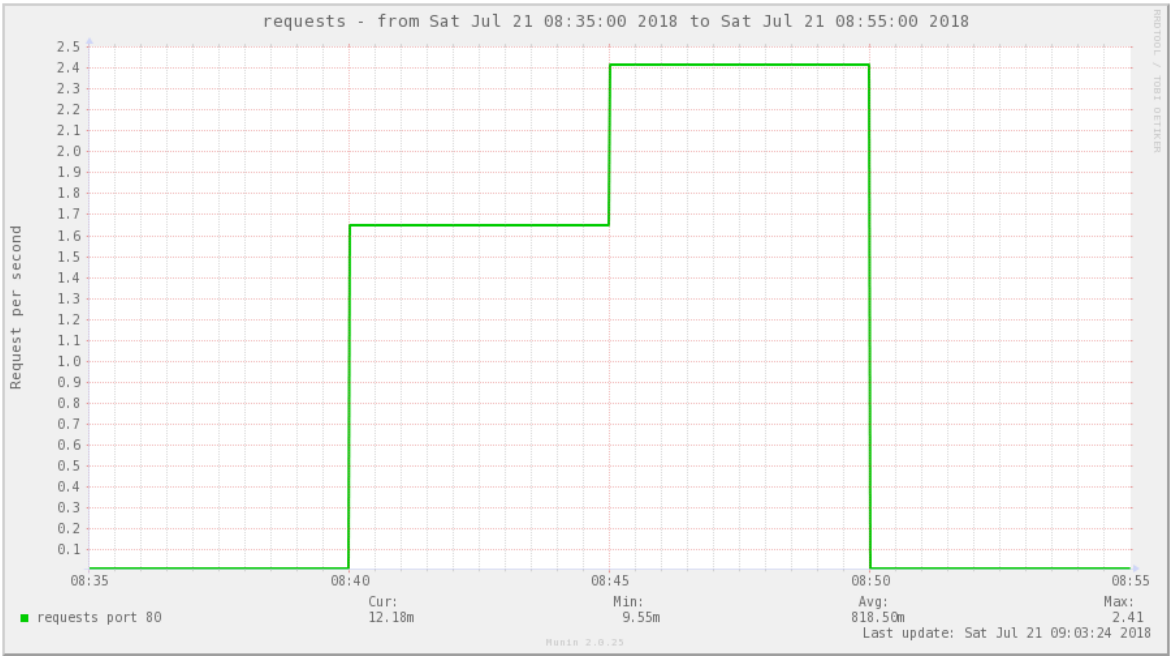
Slika 7.33. - Vrijeme odziva za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja vrijeme odziva izraženo u milisekundama, a apscisa vrijeme trajanja eksperimenta)

Slika 7.34. predstavlja grafički prikaz broj učitavanja u sekundi za izvršeni eksperiment u alatu *Apache JMeter*. U ovom slučaju za vizualizaciju je korišten graf *Hits per Second*.



Slika 7.34. - Broj učitavanja u sekundi za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj učitavanja zahtjeva u sekundi, a apscisa vrijeme)

Iz prethodne dvije slike se može vidjeti kako svi rezultati odgovaraju zadanim parametrima testa. Rezultati kao što su vrijeme odziva i broj učitavanja u sekundi se više neće prikazivati u nastavku istraživanja jer nisu glavni fokus ovog istraživanja. Kako bi bili u potpunosti sigurni da je testno okruženje ispravno potrebno je provjeriti i rezultate dobivene pomoću alata za nadzor računalnih resursa i opterećenja servisa. Slika 7.35. predstavlja broj korisničkih zahtjeva u sekundi u vremenu za izvršeni eksperiment u alatu *Munin*.



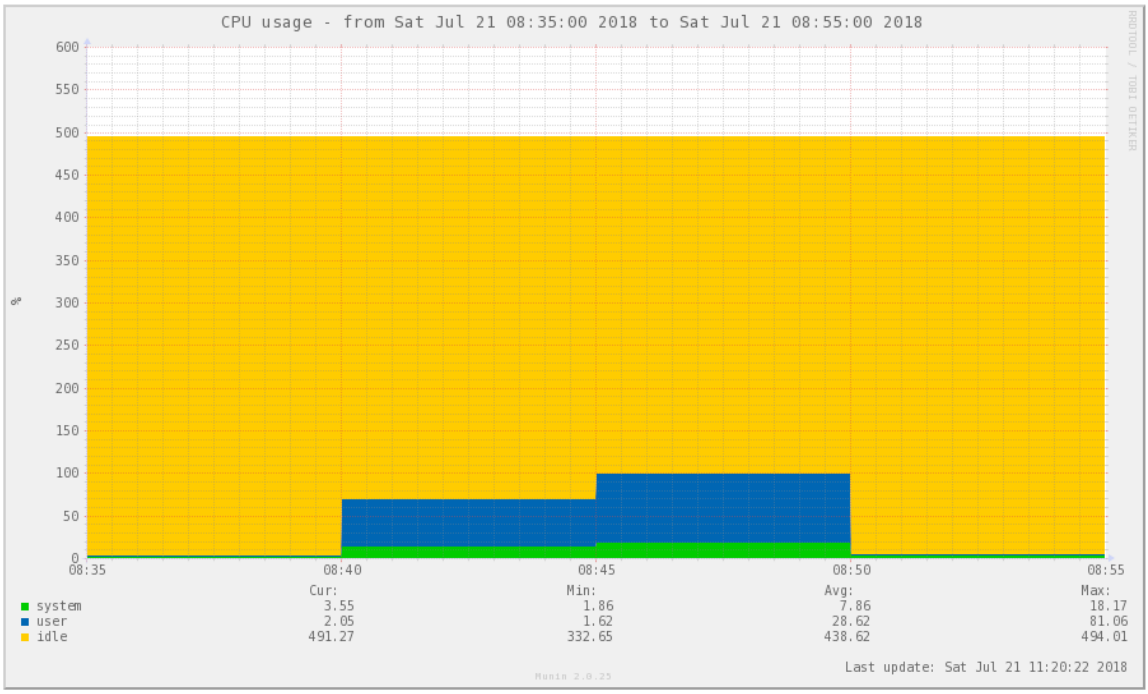
Slika 7.35. - Broj korisničkih zahtjeva u sekundi (ordinata) u vremenskom periodu (apscisa) za izvršeni eksperiment u alatu *Munin*

Na prethodnoj slici se vidi da je došlo do porasta broja korisničkih zahtjeva u sekundi za period kada je test izvršen. Broj korisničkih zahtjeva na slici je manji nego što je to definirano u eksperimentu zbog dva razloga, a oni su:

1. Alat za nadzor računalnih resursa (*Munin*) uzima vrijednost koju je dobio samo u trenutku kada je provjerio poslužitelja (svakih pet minuta), što je i objašnjeno u na početku poglavlja sedam.
2. Slika 7.35. predstavlja jednu trećinu zahtjeva jer je prikazan samo jedan Web-poslužitelj od tri koliko ih ima u Web-klasteru, odnosno predstavljeni su samo zahtjevi koje je on obradio.

Tijekom ovog istraživanja alat *Munin* je uvijek bio pokrenut, jer je riječ o alatu čija je glavna uloga neprekidni nadzor računalnih resursa i opterećenja sustava, za razliku od *Apache JMeter* alata koji se najčešće koristio samo tijekom testiranja. Upravo se zbog toga na svim rezultatima (slikama) od *Munin* alata na apscisi vide stvarno vrijeme (odnosno 24 sata) kako bi se što preciznije znalo kad su neke promjene nastale.

Kako bi bili sigurni da testno okruženje ispravno moramo provjeriti i računalne resurse (CPU i memorija) Web-poslužitelja za period kada je eksperiment izvršen. Slika 7.36. predstavlja iskoristivost procesora u vremenu za izvršeni eksperiment u alatu *Munin*.

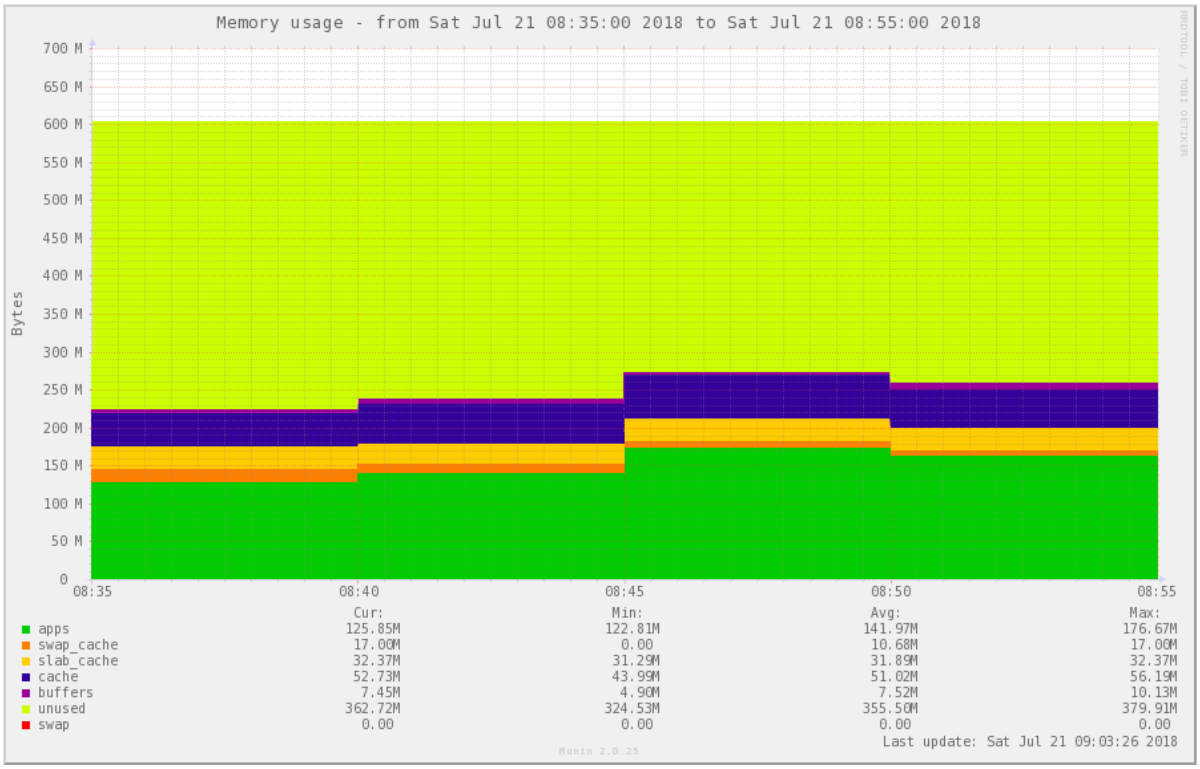


Slika 7.36. - Iskoristivost resursa procesora za izvršeni eksperiment (ordinata predstavlja postotak opterećenja procesora po broju jezgri gdje 100 predstavlja jednu jezgru, a apscisa vrijeme) u alatu *Munin*

Na slici 7.36. se nalazi nekoliko boja koje predstavljaju različito stanje resursa: zelena (resursi koji su zauzeti od strane operacijskog sustava), plava (resursi koje koriste programi i servisi pokrenuti od strane korisnika) i žuta (slobodni resursi).

Iz prethodne slike se vidi da Web-poslužitelj ima ukupno pet jezgri koje su označene s 500 jer se ujedno i mjeri postotak ukupnog opterećenja procesora prema broju jezgri. Također se vidi da je tijekom testiranja došlo do rasta opterećenja procesora.

Prilikom provođenja ovog testa Web-poslužitelj je imao ukupno 0.6 GB memorije. Slika 7.37. predstavlja iskoristivost memorije za izvršeni eksperiment u alatu *Munin*. Važno je napomenuti da je u preliminarnom istraživanju realnih sustava utvrđeno da su promjene u opterećenju memorije znatno manje nego što je to slučaj kao kod procesora gdje je opterećenje znatno lakše postići. To ne znači da ne postoje Web-stranice koje zahtijevaju više memorije, već samo da u ovom preliminarnom istraživanju to nije bio slučaj.



Slika 7.37. - Iskoristivost memorije za izvršeni eksperiment (ordinata predstavlja memoriju izraženu u MB, a apscisa vrijeme) u alatu *Munin*

Na slici 7.37. se nalazi nekoliko boja koje predstavljaju različito zauzeće memorije: zelena (programi i servisi), narančasta/žuta/plava (predmemorija), ljubičasta (međuspremnik), žuto zelena (slobodna memorija) i crvena (virtualna memorija).

Iz prethodne slike se vidi kako je došlo do povećanja opterećenja memorije prilikom provođenja testa. Temeljem rezultata dviju prethodnih slika možemo zaključiti da je testno okruženje ispravno dizajnirano jer je dokazano da broj korisničkih zahtjeva utječe na opterećenje računalnih resursa (CPU i memorija). Nakon provjere ispravnosti testnog kruženja slijedi provođenje testova koji su definirani u prethodnim aktivnostima.

7.5.6 Aktivnost 6. - Provesti testiranje

Ova aktivnost predstavlja proces testiranja u kojoj je važno da ima nadzor i kontrola resursa tijekom testiranja i jasno definiran način prikupljanja rezultata. Kako bi se osigurao nadzor i kontrola resursa koriste se već navedeni alati *Observium* i *Munin*. Pored alata postoje i dodatne kontrole sustava koje su dio novog modela (poglavlje 7.3.1.5). Kontrole su ostvarene u novoj aplikaciji koja će se koristiti prilikom validacije modela i omogućavaju dodatnu kontrolu i nadzor sustava. One se sastoje od nekoliko mehanizama koji omogućavaju: neprekidnost izvršavanja sustava (aplikacije koja je razvijena pomoću novog modela), nadzor resursa, provjera konzistentnosti u radu poslužitelja i mogućnost upozorenja ako je došlo do greške ili do preopterećenja računalnih resursa. Osim navedenih mehanizama kao dio aplikacije razvijen je i sustav za bilježenje promjena koji cijelo vrijeme prati sustav i bilježi: kada je došlo

do promjene resursa (CPU i memorija), opterećenje resursa, dostupnost Web-poslužitelja, metoda promjene resursa i sl. Navedeni kontrolni sustavi, alati za nadzor računalni resursa i sustavi za bilježenje promjena omogućavaju da se rezultati mogu pohraniti i naknadno provjeriti ako za to bude potrebe.

Nakon što je razvijena aplikacija po uzoru na predloženi model izvršeno je puno eksperimenta za simulaciju korisničkih zahtjeva koji su rezultirali opterećenje računalnih resursa (CPU i memorije). Temeljem provedenih eksperimenata i rezultata iz stvarnih sustava (Tablica 7.1.) definirani su reprezentativni eksperimenti koji su prikazani u nastavku rada kroz tri faze.

7.5.6.1 Faza I - validacija prve hipoteze

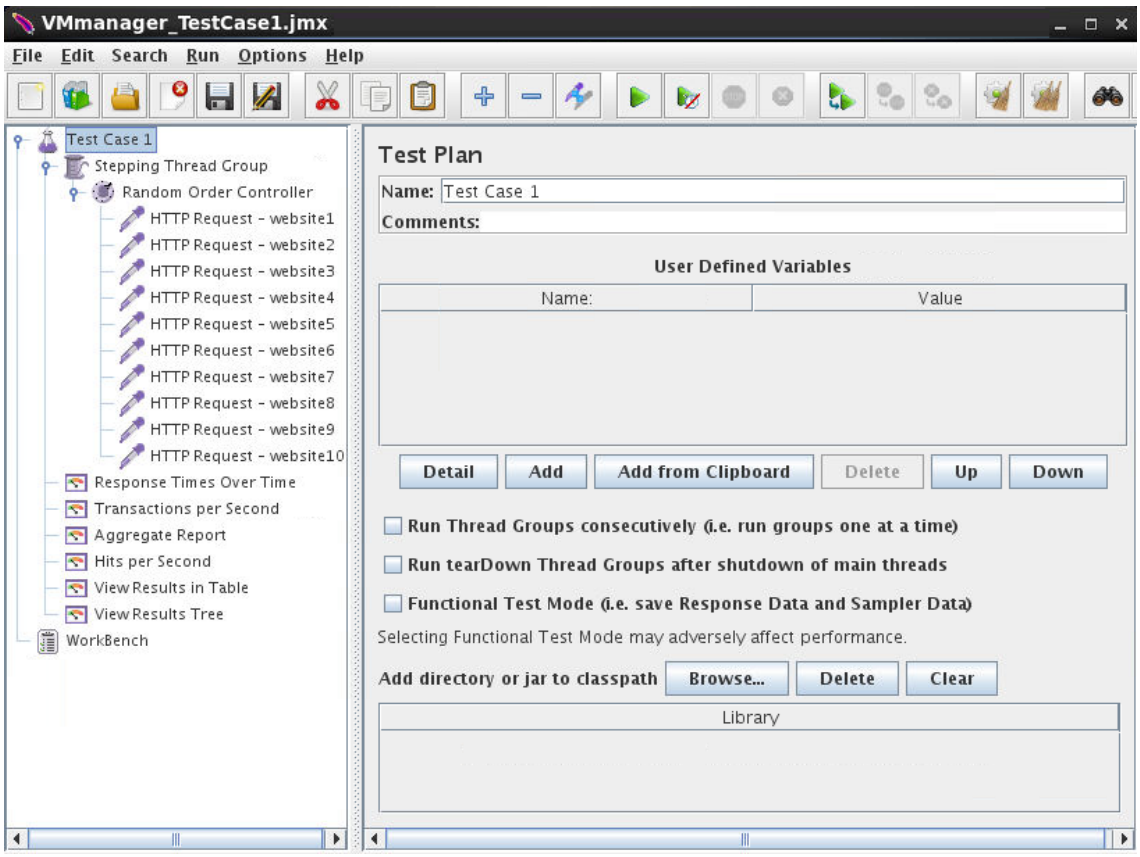
Prva faza predstavlja provjeru prve hipoteze: *Primjena predloženog modela nad virtualiziranim sustavima omogućava dodavanje/oduzimanje postojećih računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja*.

Prvi cilj predstavlja provjeru je li moguće dodavanje i oduzimanje računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja. Danas gotovo svi operacijski sustavi imaju ostvarene mehanizme za provjeru koliko je sustav aktivan odnosno kad je izvršeno zadnje pokretanje sustava. Upravo se taj mehanizam koristio kako bi se provjerilo je li došlo do ponovnog pokretanja poslužitelja nakon raspodjele resursa. U nastavku rada slijede testovi koji su podijeljeni u grupe kako bi se provjerile sljedeće tri mogućnosti:

1. dodavanje i oduzimanje računalnih resursa,
2. napredno dodavanje i oduzimanje računalnih resursa i
3. primjena na više Web-klastera.

Eksperiment 1 - Dodavanje i oduzimanje računalnih resursa

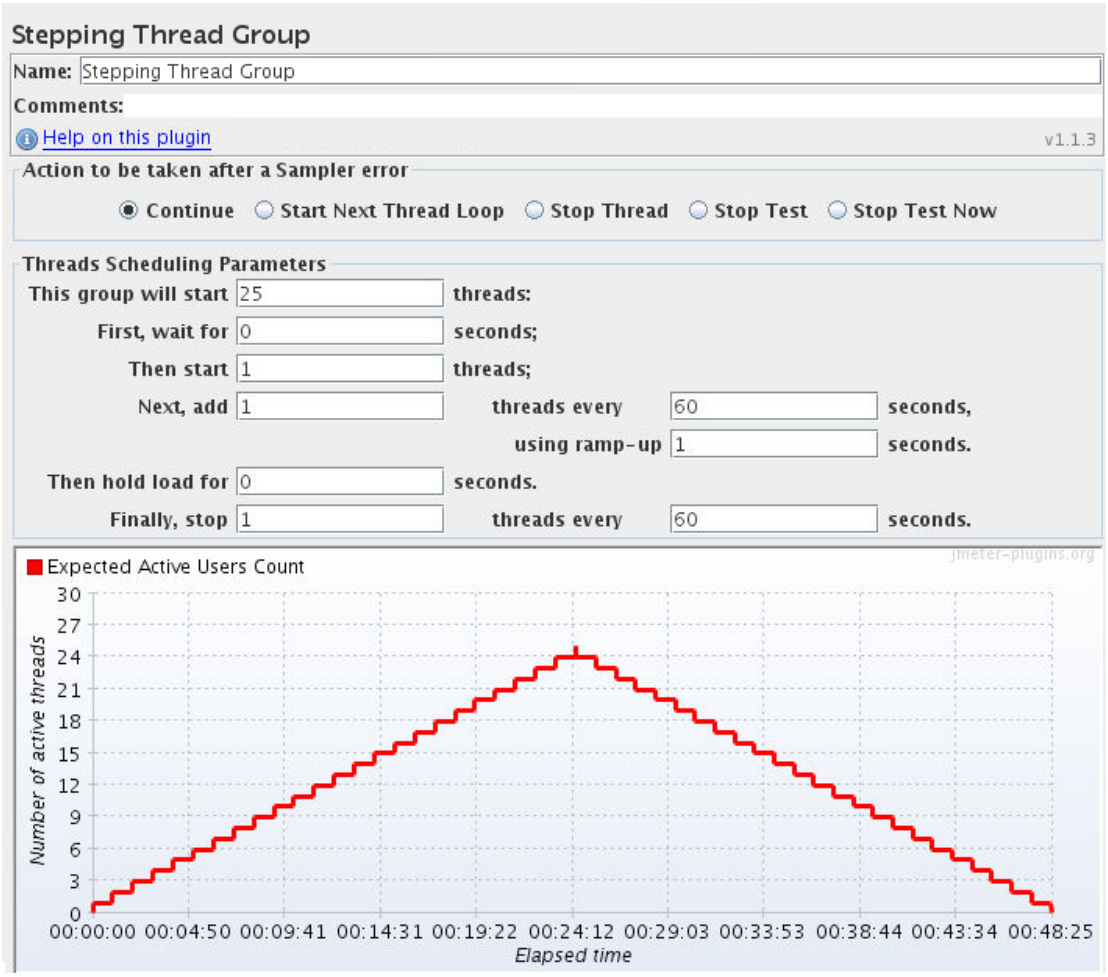
U prvom testu zahtjevi linearno rastu te zatim opadaju, a test je izvršen na jednom Web-klasteru od deset Web-stranica. Svrha testa je pokazati kako je moguće povećati i smanjiti računalne resurse (CPU i memoriju) bez ponovnog pokretanja Web-poslužitelja. Test se sastoji od dvije glavne komponente, a one su: broj dretvi (broj simultanih korisnika) i Web-stranice nad kojom se ispitivanje izvršava (ukupno deset). Slika 7.38. predstavlja grafičko sučelje *Apache Jmeter* alata s postavkama prvog testa.



Slika 7.38. - Prikaz postavki u alatu *Apache JMeter* za prvi eksperiment

U prvom testu u alatu *Apache JMeter* se koristi komponente koja se zove *Stepping Thread Group*. Ona omogućava stvaranje simulacija u kojima broj korisničkih zahtjeva raste i pada. Druga komponenta koja se koristi u testu je *Random Order Controller* koja omogućava da se svaki element u slučajnom redoslijedu izvrši najviše jednom. U ovom slučaju elementi su HTTP zahtjevi kojih ima 10, jer svaki Web-klaster ima 10 Web-stranica.

Slika 7.39. predstavlja glavnu konfiguraciju prvog testa koji započinje s jednim simultanim korisnikom čiji se broj povećava za jedan svake minute. Postupak traje do trenutka kada je prisutno 25 simultanih korisnika čiji se onda broj linearno smanjuje za jedan svake minute dok ne dođe do nule kada test i završava. Cijeli test traje ukupno 50 minuta.



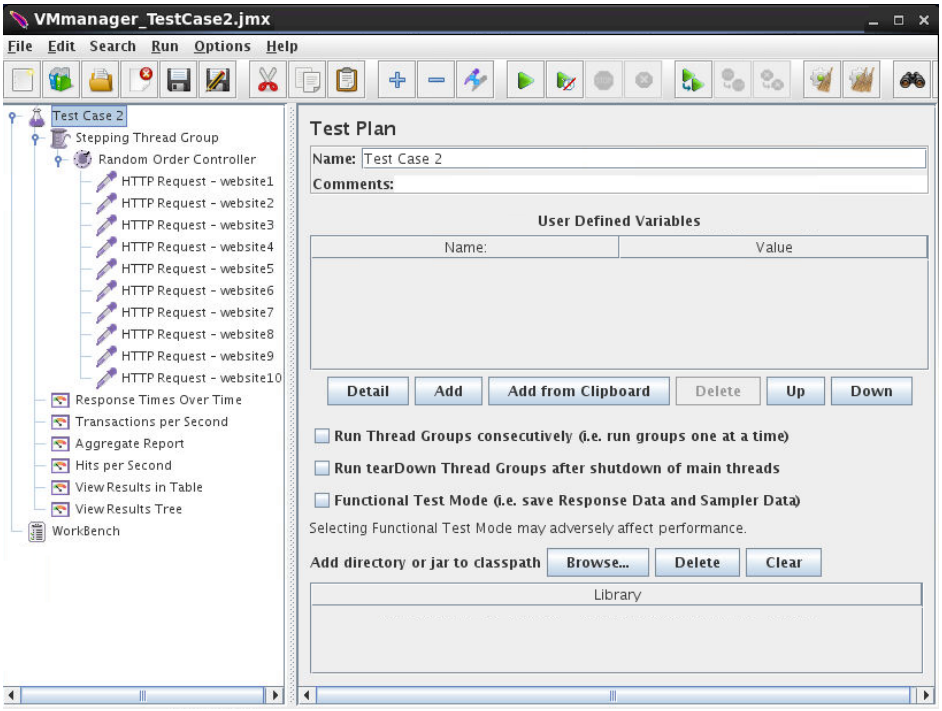
Slika 7.39. - Prikaz parametara u alatu *Apache JMeter* za prvi eksperiment

Ovim testom se želi provjeriti ponašanje Web-klastera u slučaju kada zahtjevi proporcionalno rastu odnosno padaju što rezultira potrebu za povećanjem odnosno smanjenjem računalnih resursa.

Eksperiment 2 - Napredno dodavanje i oduzimanje računalnih resursa

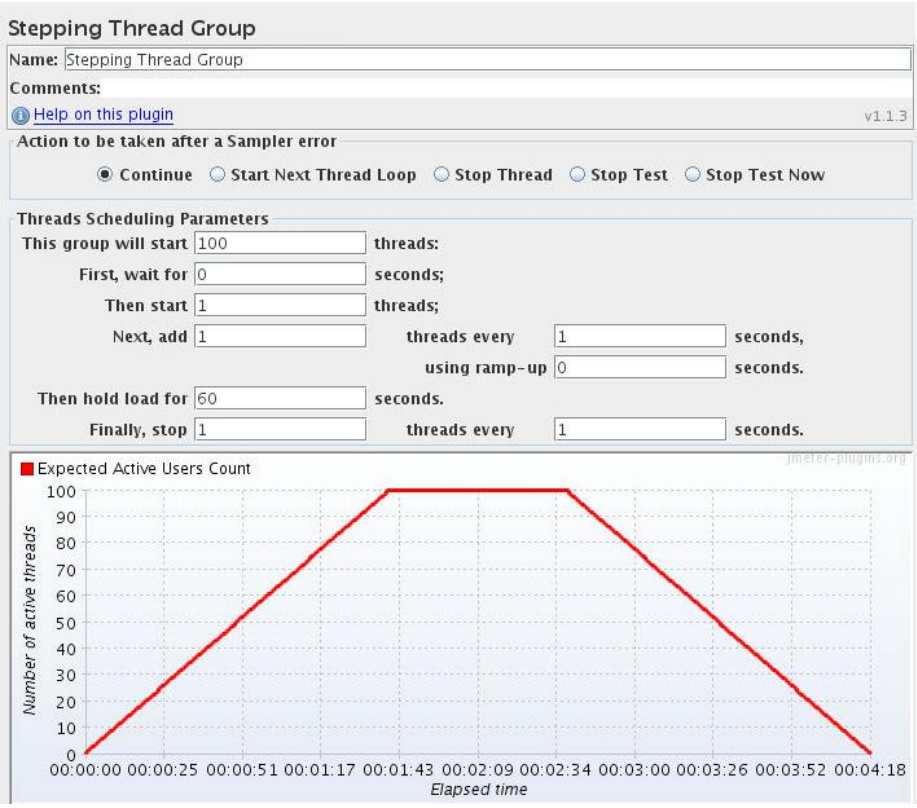
Cilj ovog testa je provjeriti mogućnost napredne dodjele računalnih resursa u situacijama kada dođe do naglog opterećenja sustava za razliku od prethodnog testa. U ovom slučaju resursi bi se trebali naglo povećati kako ne bih došlo do zagušenja sustava. Međutim, to povećanje mora također biti optimalno tj. sustav mora dodijeliti točno onoliko resursa koliko je potrebno Web-poslužitelju.

Kako bi ovo provjerili test je koncipiran na način da se u kratkom periodu dođe veliki broj simultanih korisnika (ukupno 100) što bi trebalo rezultirati veliku potrebu za resursima. Test se također provodi nad jednim Web-klasterom od deset Web-stranica što je i prikazano na slici 7.40. U ovom testu su korištene iste komponente kao u prvom testu (*Stepping Thread Group* i *Random Order Controller*).



Slika 7.40. - Prikaz postavki u alatu *Apache JMeter* za drugi eksperiment

Broj simultanih korisnika se povećava svake sekunde za jedan dok se broj ne poveća na 100 korisnika. Nakon toga, test od 100 korisnika se izvršava 60 sekundi nakon čega dolazi do pada simultanih korisnika svake sekunde za jedan korisnik. U trenutku kada broj korisnika padne do broja nula test se i završava. Ukupno trajanje testa je 260 sekundi. Slika 7.41. predstavlja pregled glavnih parametara drugog testa.

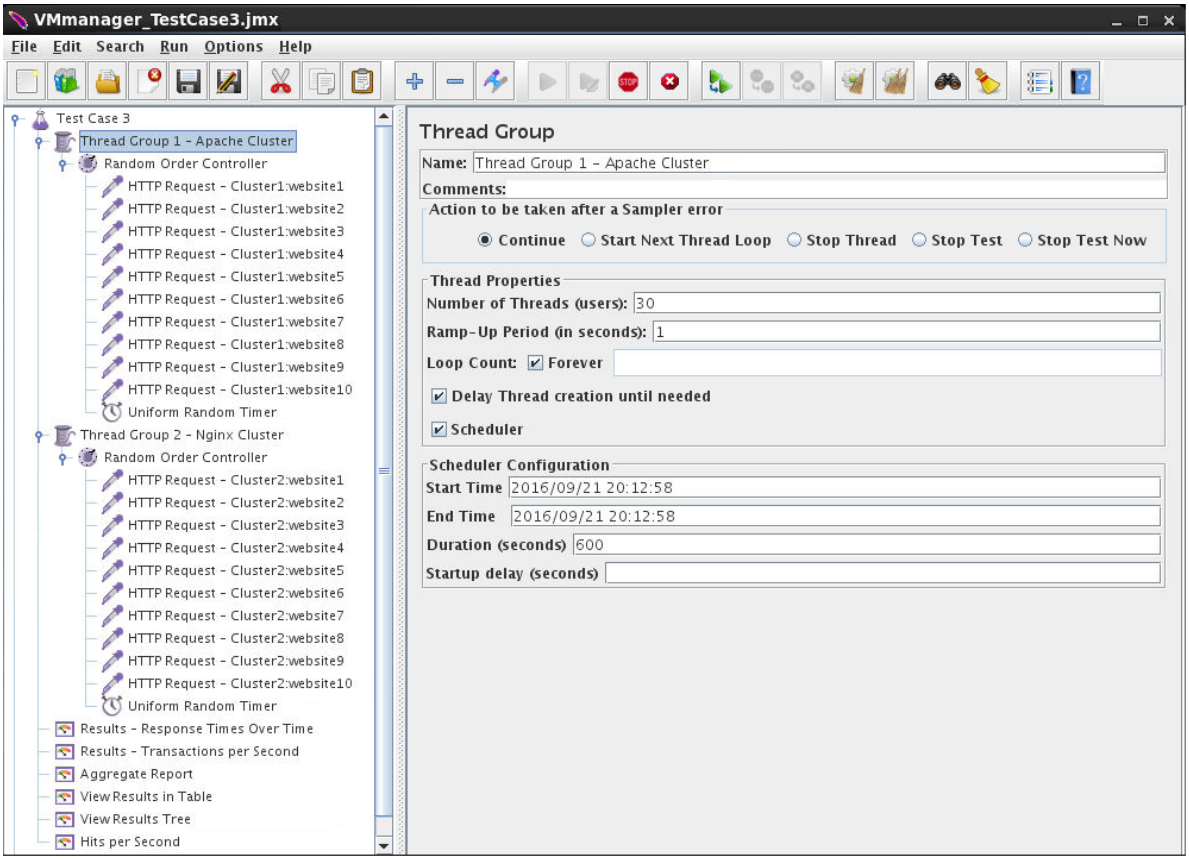


Slika 7.41. - Prikaz parametara u alatu *Apache JMeter* za drugi eksperiment

Kako bi se rezultati čim prije vidjeli Web-poslužitelju je kao i u prvom testu na početku dodijeljen manji broj resursa (1 CPU jezgra i 0.6 GB memorije).

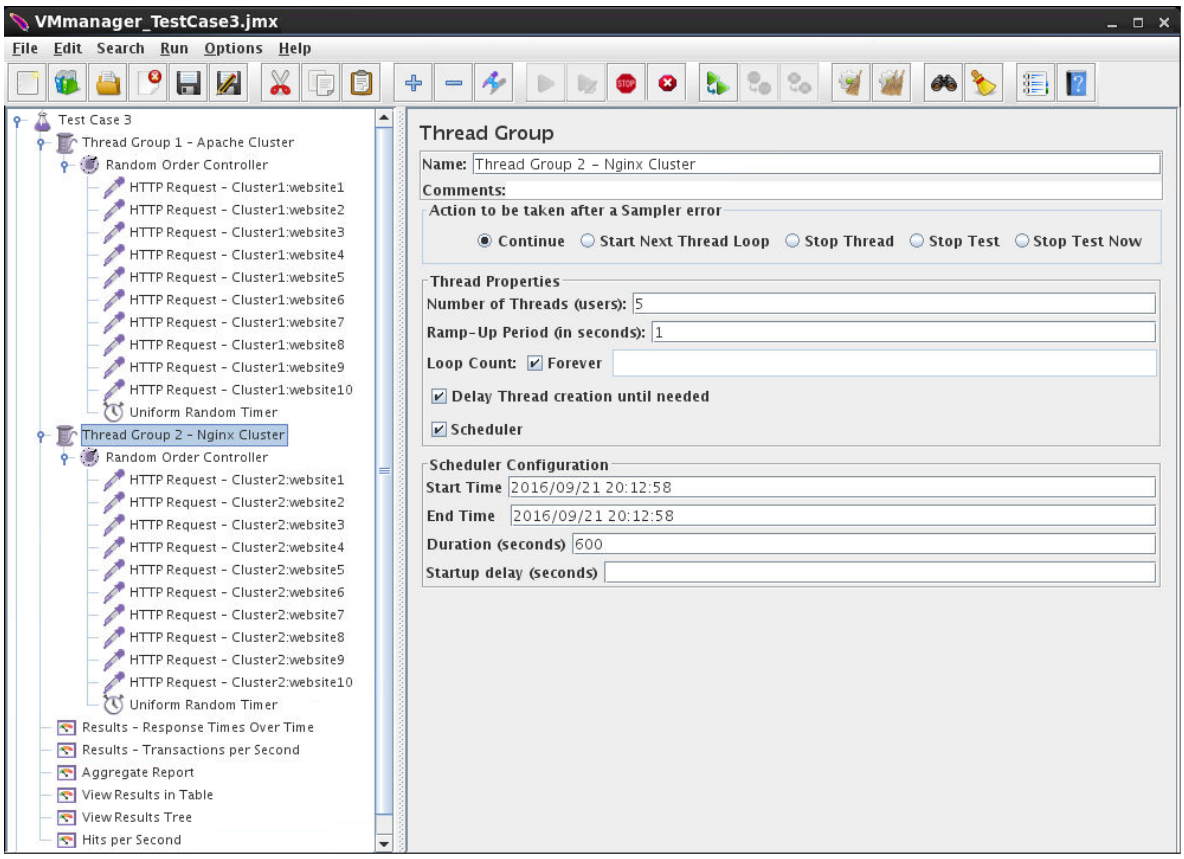
Eksperiment 3 - Primjena na više Web-klastera

Ovim testom se želi provjeriti je li model skalabilan odnosno primjenjiv na više poslužitelja. U ovom slučaju provjera je izvršena na primjeru Web-klastera. Ukupno su dva Web-klastera od kojih svaki ima po tri Web-poslužitelja i deset Web-stranica. Test je koncipiran na način da je na prvom Web-klasteru (*apache*) 30 simultanih korisnika, a na drugom Web-klasteru (*nginx*) pet simultanih korisnika. Slika 7.42. predstavlja prikaz postavki u alatu *Apache JMeter* za treći eksperiment za *apache* Web-klaster.



Slika 7.42. - Prikaz postavki u alatu *Apache JMeter* za treći eksperiment za *apache* Web-klaster

Slika 7.43. predstavlja prikaz postavki u alatu *Apache JMeter* za treći eksperiment za *nginx* Web-klaster.



Slika 7.43. - Prikaz postavki u alatu *Apache JMeter* za treći eksperiment za *nginx* Web-klaster

Ovim bi se provjerilo ne samo je li model primjenjiv na više Web-poslužitelja koji se nalaze na više fizičkih poslužitelja već i više tehnologija (*apache i nginx*). Cijeli test traje ukupno deset minuta. U ovom testu se pored komponenta iz prva dva testa (*Stepping Thread Group* i *Random Order Controller*) koristi još i *Uniform Random Timer* čija je svrha da odgodi uzastopne zahtjeve iste dretve prema definiranim donjim i gornjim granicama.

7.5.6.2 Faza II - validacija druge hipoteze

Druga faza predstavlja provjeru druge hipoteze: *Primjena predloženog modela ne narušava konzistentan rad poslužitelja*.

Prethodno je navedeno da novi model ima komponentu koja se zove kontrola sustava i ona se sastoji od nekoliko mehanizama koji su opisani u poglavlju 7.3.1.5. Slika 7.44. predstavlja parametre virtualnog poslužitelja koje fizički poslužitelj dohvaća prilikom svake provjere sustava. Osim parametara koji su se koristili u prvoj fazi (dodijeljeni resursi i opterećenje), dodana je i provjera glavnog servisa (u ovom slučaju Web-poslužitelja).

```
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close
Content-Length: 136

load_1=0.54
load_5=1.86
load_15=2.21
cpu=4
total=8367772
free=7874896
available=7939980
free_percentage=94.10
```

Slika 7.44. - Primjer parametara i vrijednosti virtualnog poslužitelja koje dohvaća fizički poslužitelj

Ovom provjerom se vrši i provjerava samog operacijskog sustava virtualnog poslužitelja. Glavni servis predstavlja namjenu poslužitelja i to su najčešće servisi kao što su: baze podataka, Web-poslužitelj, sustav za sigurnosne kopije, sustav za bilježenje i sl.

7.5.6.3 Faza III - validacija treće hipoteze

Treća faza predstavlja provjeru treće hipoteze: *Primjenom predloženog modela nad virtualiziranim sustavima postiže se bolja iskoristivost postojećih računalnih resursa (CPU i memorija) s obzirom na prvobitno stanje*.

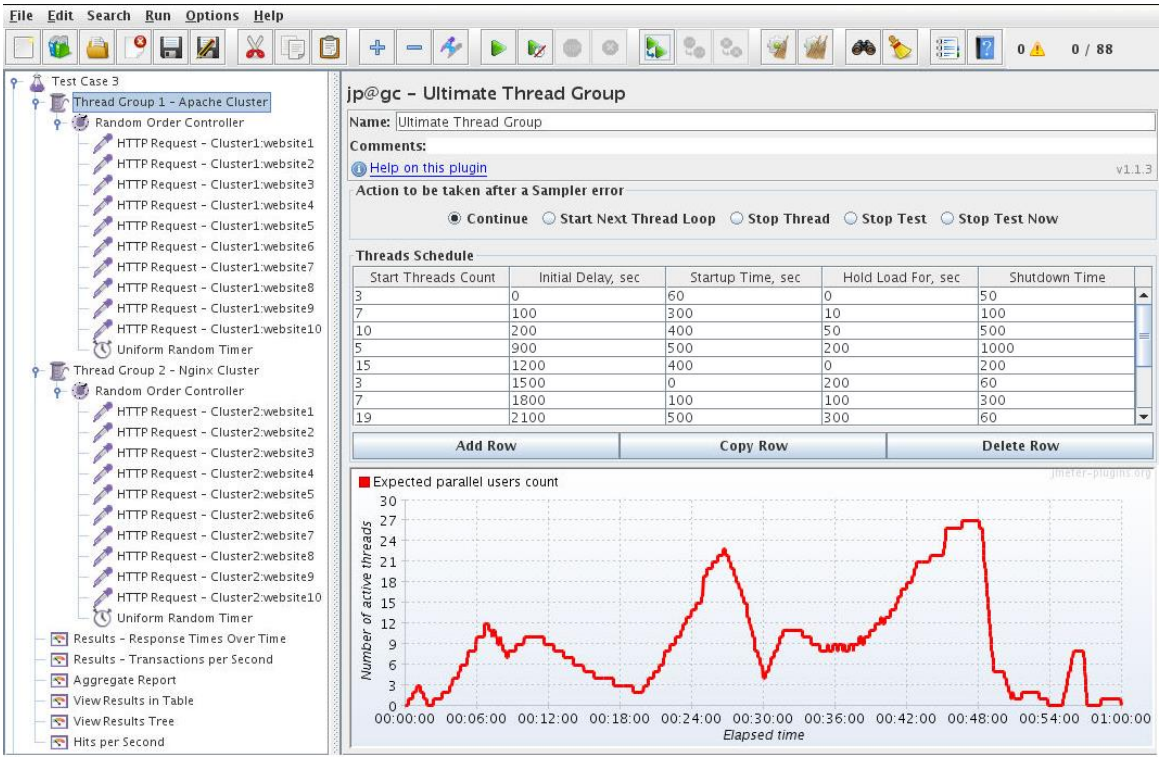
Kako bi se provjerila ova hipoteza ponovljen je jedan veći eksperiment koji predstavlja kombinaciju testova iz prve faze, jer su oni konkretni primjeri iz prakse. Prvo su obavljeni testovi bez primjene nove aplikacije koja je razvijena na temelju novog modela za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa. Nakon toga su izvršeni isti testovi, ali s primjenom razvijene aplikacije. U oba slučaja je izvršen nadzor i praćenje računalnih resursa poslužitelja. Nakon ispitivanja odrađena je usporedba rezultata kako bi se provjerilo je li došlo do poboljšanog iskorištenja postojećih računalnih resursa. Ako je novi model dobar, poboljšanje u iskoristivosti postojećih računalnih resursa u odnosu na prvobitno rješenje (CPU, odnosno broj jezgri i memorija) bi trebalo biti vidljivo na samim slikama koje predstavljaju rezultate eksperimenta. Već je spomenuto da je temeljem provedenih eksperimenata i rezultata iz stvarnih sustava (Tablica 7.1.) definirani reprezentativni eksperiment koji se sastoji od nekoliko scenarija, a oni su:

- linearni rast odnosno pad korisničkih zahtjeva,
- iznenadni rast odnosno pad korisničkih zahtjeva i
- stanje mirovanja, odnosno kada ima malo ili nikako korisničkih zahtjeva.

Drugim riječima, kako bi se provjerila treća hipoteza izvršen je jedan veći eksperiment koji predstavlja kombinaciju testova iz prve faze, jer su u njoj predstavljeni konkretni primjeri iz prakse. Važno je napomenuti da vjerojatno postoje varijacije (scenariji) koji nisu uključeni u

ovaj veći eksperimenti. Razlog tome je što je glavni cilj ovo istraživanja pokazati da se postojeći računalni resursi (CPU i memorija) bolje iskorištavaju bez potrebe za ponovnim pokretanjem sustava u najčešćim scenarijima koji su prepoznati u stvarnim sustavima.

Slika 7.45. predstavlja prikaz parametara za eksperiment u alatu *Apache Jmeter*. Test se sastoji od broja simultanih korisnika (korisničkih zahtjeva) koji se izmjenjuju u periodu od šezdeset minuta. Kao što je već na početku napomenuto vremenska jedinica, odnosno trajanje eksperimenta nije važno, jer se u ovom slučaju promatra opterećenje sustava. Cilj je da se u određenom periodu (u ovom slučaju šezdeset minuta) provjere navedeni scenariji (broj korisničkih zahtjeva) koji su prepoznati u stvarnim sustavima iz prakse.



Slika 7.45. - Prikaz parametara u alatu *Apache JMeter*

Na slici 7.45. se vidi da je eksperiment definiran na način kako bi se mogla provjeriti sva tri prethodno navedena scenarija: linearni rast odnosno pad korisničkih zahtjeva, iznenadni rast odnosno pad korisničkih zahtjeva i stanje mirovanja. U ovom eksperimentu su korištene komponente alata *Apache Jmeter* koje su već prethodno opisane. Test je izvršen nad oba Web-klastera (*apache i nginx*). Na prethodnoj slici su u tabelarnom dijelu navedeni korisnički zahtjevi odnosno dretve (njihova količina, kad se pokreću, koliko traju i nakon kojeg vremena završavaju).

U nastavku rada slijedi posljednja aktivnost tj. predstavljanje rezultata za navedene eksperimente iz ove aktivnosti.

7.5.7 Aktivnost 7. - *Analizirati rezultate, izvješća i ponoviti testiranje*

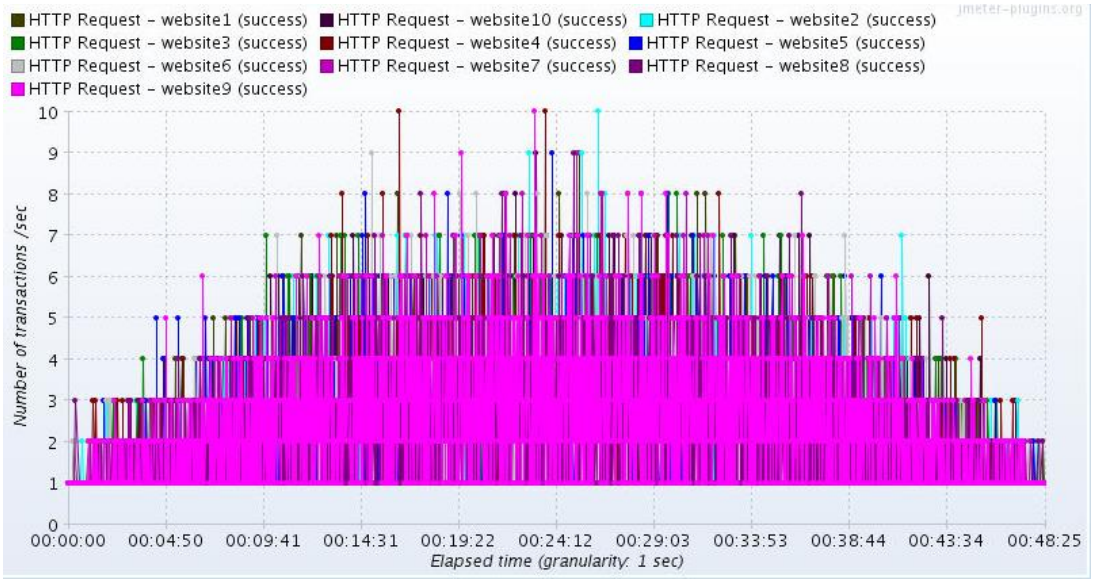
Posljednja aktivnost predstavlja prikupljanje i predstavljanje dobivenih rezultata. Prezentacija rezultata ovisi o važnosti samih rezultata i za koga su rađeni odnosno tko je auditorij (npr. prezentacija za upravu i dioničare, tehnički izvještaj i sl.). Prilikom provjere rezultata potrebno je provjeriti li je došlo do nekih problema kao npr. narušavanje zadanih limita u testovima (engl. *thresholds*), odnosno metričke vrijednosti unutar prihvaćenih granica. U nastavku istraživanja slijedi prikaz rezultata koji su grupirane u tri faze prema izvršenim testovima iz prijašnjeg koraka.

7.5.7.1 *Faza I - Rezultat validacije prve hipoteze*

U prošloj aktivnosti u prvoj fazi su provedena tri različita testa kako bi se provjerilo je li moguće dodavanje i oduzimanje računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja. Tijekom sva tri testa korišten je isti algoritam u sustavu za raspodjelu opterećenja HTTP/HTTPS prometa odnosno zahtjeva. Algoritam se zove *Weighted Least Connections* i on se najčešće koristi kada se sustav poznaje, odnosno kada se zna koliko sustav može podnijeti zahtjeva. Na temelju te informacije definira se tzv. težište odnosno maksimalni broj zahtjeva (veza) koje jedan čvor (Web-poslužitelj) može primiti tj. obraditi. Proces se odvija ciklično i zahtjevi idu prvo na čvor s najvećim slobodnim brojem veza odnosno raspoloživih resursa za njihovu obradu. Upravo ovakav pristup jamči da su svi Web-poslužitelji jednako opterećeni zbog čega je u nastavku prikazan samo jedan reprezentativni uzorak.

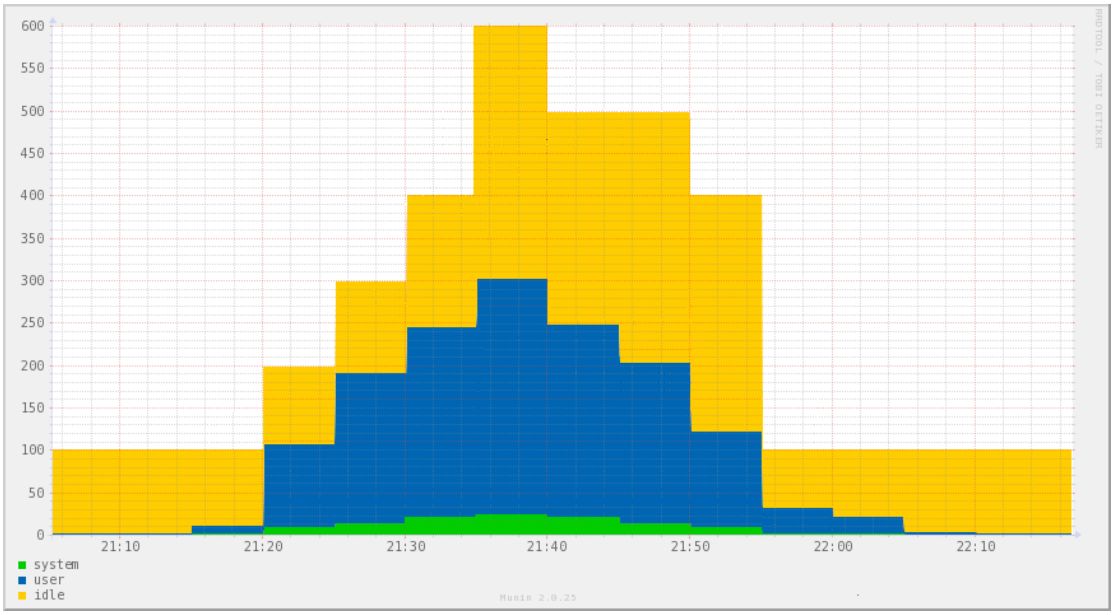
Rezultati za eksperiment 1 - Dodavanje i oduzimanje računalnih resursa

Slike 7.46. predstavljaju broj izvršenih zahtjeva u vremenu za prvi eksperiment u alatu *Apache JMeter*. Kao što je već definirano u prethodnoj aktivnosti test je trajao 50 minuta.



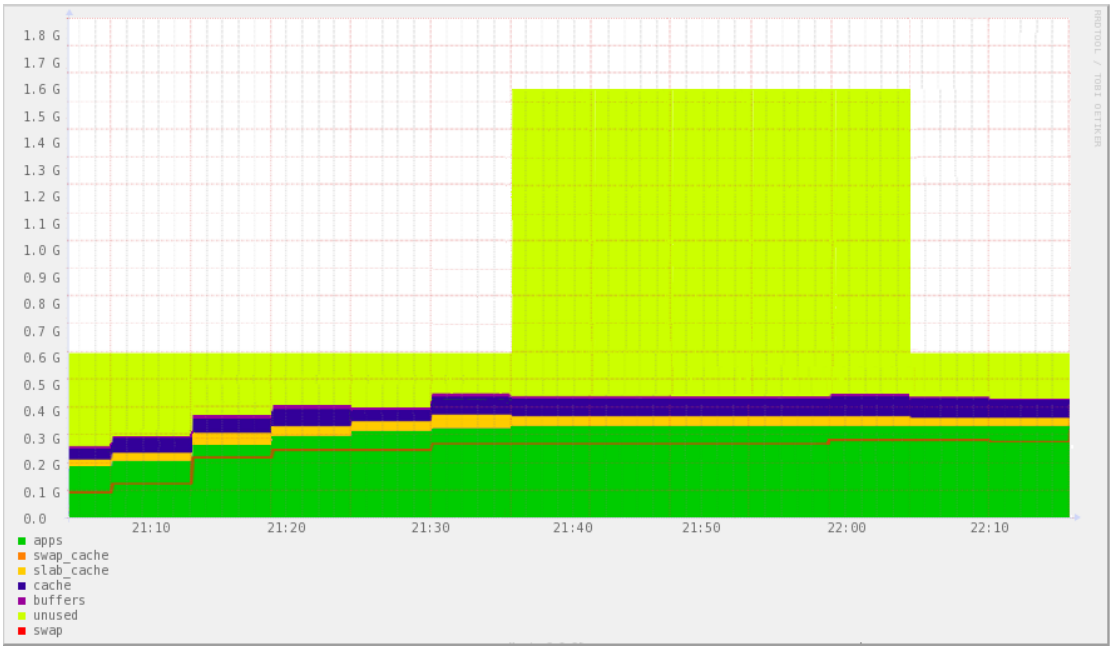
Slika 7.46. - Broj zahtjeva u sekundi za prvi eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj izvršenih zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta)

Na slici 7.46. se vidi kako broj zahtjeva postepeno raste do određenog broja, te se zatim smanjuje što je i cilj prvog testa. Povećanje broja zahtjeva odnosno broja simultanih korisnika je rezultiralo i povećanje CPU i memorije za sve Web-poslužitelje promatranog klastera. Nakon što se broj zahtjeva počeo smanjivati, smanjivali su se i dodijeljeni resursi. Slika 7.47. predstavlja iskoristivost resursa procesora jednog Web-poslužitelja za prvi eksperiment u alatu *Munin*.



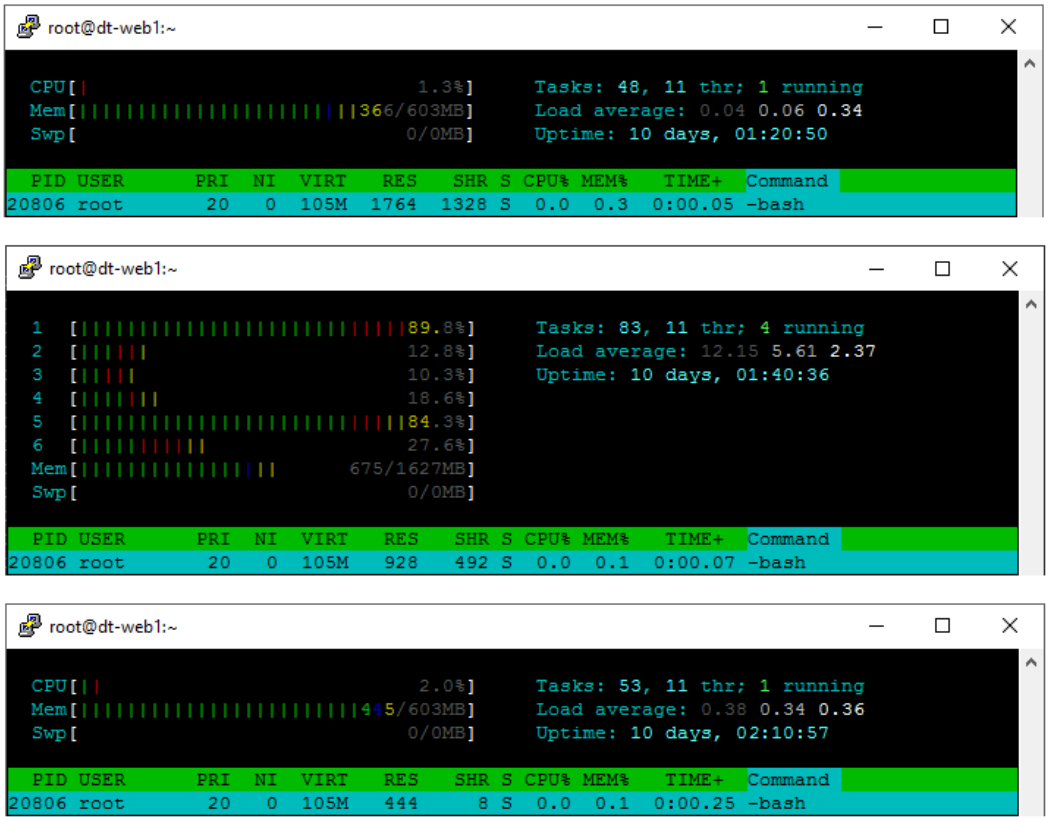
Slika 7.47. - Iskoristivost resursa procesora za prvi eksperiment (ordinata predstavlja postotak opterećenja jezgri gdje 100 predstavlja jednu jezgru, a apscisa vrijeme) u alatu *Munin*

Slika 7.48. predstavlja iskoristivost memorije jednog Web-poslužitelja za prvi eksperiment u alatu *Munin*.



Slika 7.48. - Iskoristivost memorije za prvi eksperiment (ordinata predstavlja memoriju izraženu u GB, a apscisa vrijeme) u alatu *Munin*

Slika 7.49. prikazuje stanje resursa pomoću naredbe *htop* za Web-poslužitelja (dt-web1) nad kojim je ispitivanje izvršeno. Na slici su prikazana tri stanja za prvi eksperiment: prije testa, tijekom najvećeg opterećenja i poslije ispitivanja.



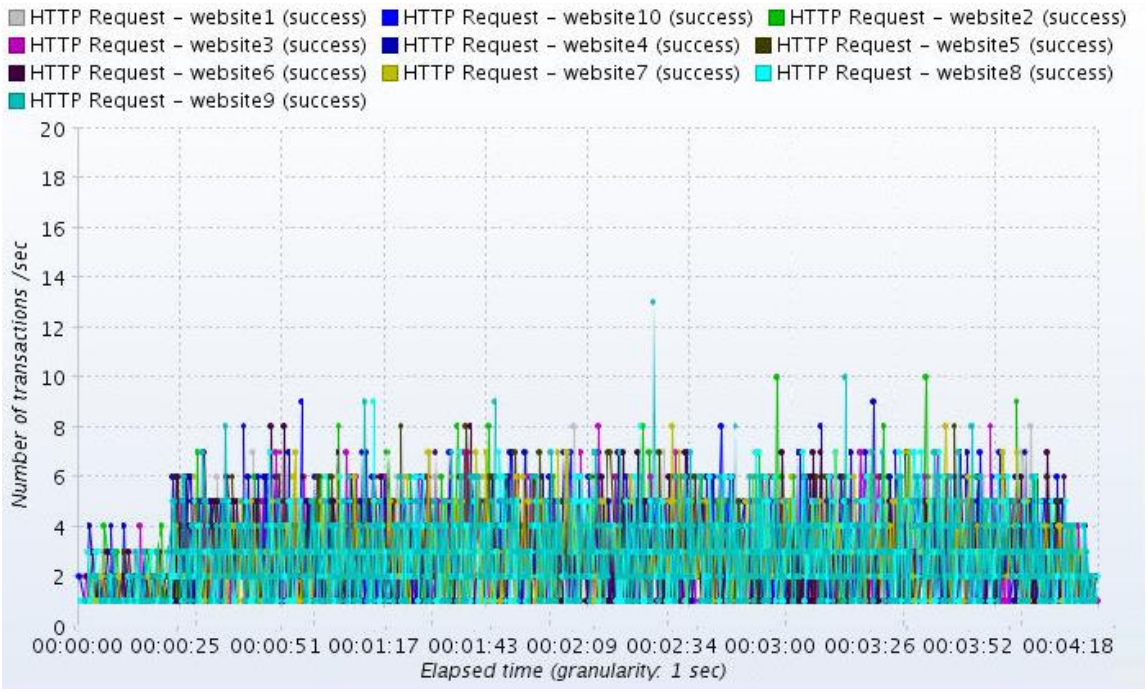
Slika 7.49. - Pregled računalnih resursa poslužitelja (dt-web1) pomoću naredbe *htop* za prvi eksperiment: prije testa (gornja slika), tijekom testa (srednja slika) i nakon testa (donja slika)

Prije testa Web-poslužitelj je imao dodijeljenu jednu jezgru i 0.6 GB memorije što je u trenutku najvećeg opterećenja poraslo na šest jezgri i 1.6 GB memorije jer je definirani inkrement za jezgre i memoriju bio jedan. Nakon što je test obavljen Web-poslužitelj se vratio u prvobitno stanje tj. na jednu jezgru i 0.6 GB memorije.

Iz slike 7.49. je također vidljivo da nije došlo do prekida u radu Web-poslužitelja te da je cijelo vrijeme bio aktivan tijekom izmjene resursa (engl. *uptime*).

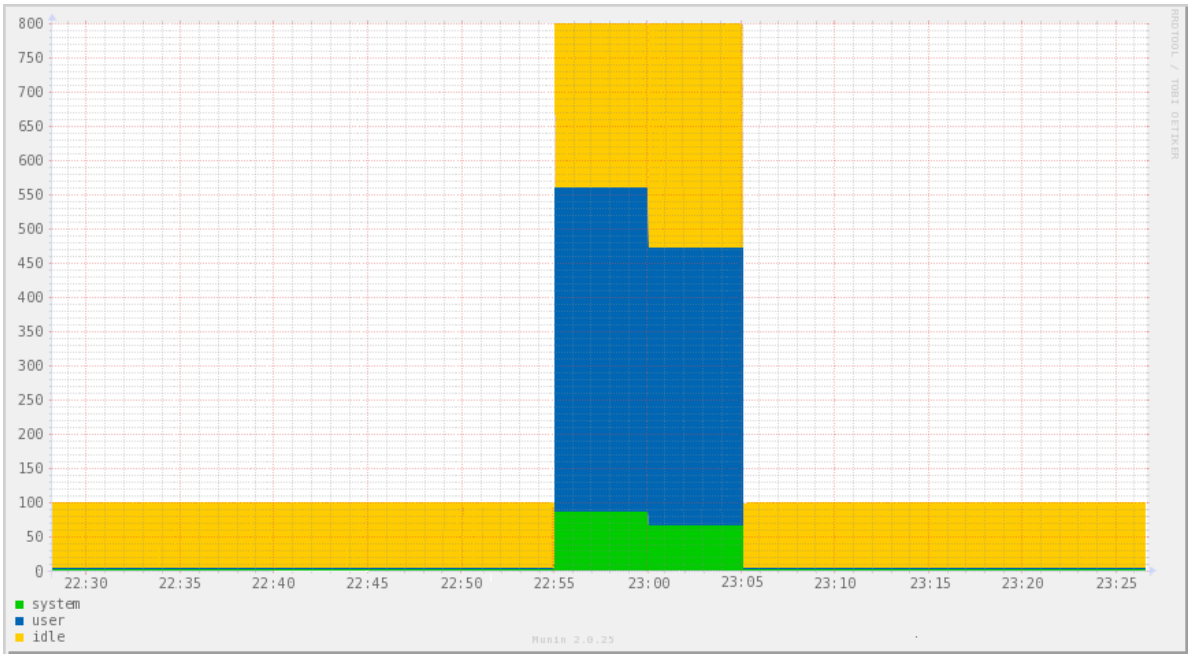
Rezultati za eksperiment 2 - Napredno dodavanje i oduzimanje računalnih resursa

Drugi test je sličan prvom, ali je vremenski dosta kraći i broj zahtjeva odnosno simultanih korisnika je puno veći jer se cilj bio ispitati ponašanje sustava u iznenadnom opterećenju tj. da li će resursi postepeno rasti ili će sustav odmah dodijeliti točno onoliko resursa koliko potrebno Web-poslužitelju u promatranom trenutku. Slika 7.50. predstavlja broj zahtjeva u sekundi za drugi eksperiment u alatu *Apache JMeter*. Test je ukupno trajao 260 sekundi.



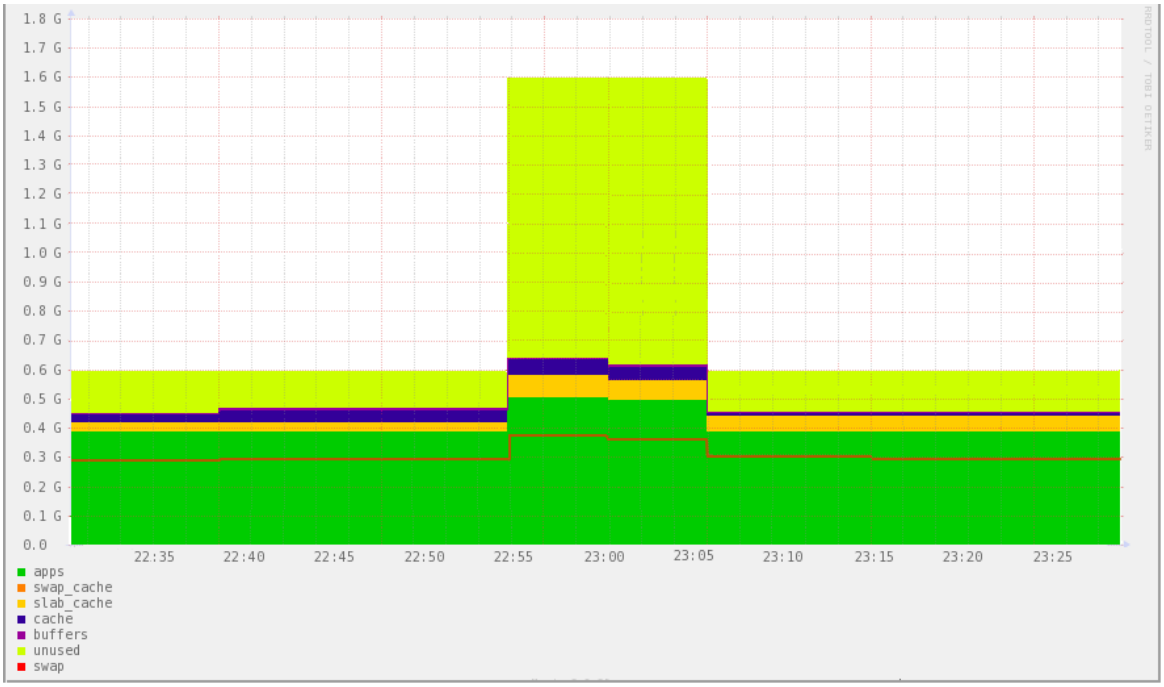
Slika 7.50. - Broj zahtjeva u sekundi za drugi eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj izvršenih zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta)

Na slikama 7.51. i 7.52. je vidljivo da su resursi naglo porasli tj. da nisu postepeno dodjeljivani što je i bio cilj ovog testa. Nakon što je ispitivanje obavljeno isti ti resursi su se naglo smanjili kako bi se postigla što bolja iskoristivost računalnih resursa. Slika 7.51. predstavlja iskoristivost resursa procesora jednog Web-poslužitelja za drugi eksperiment u alatu *Munin*.



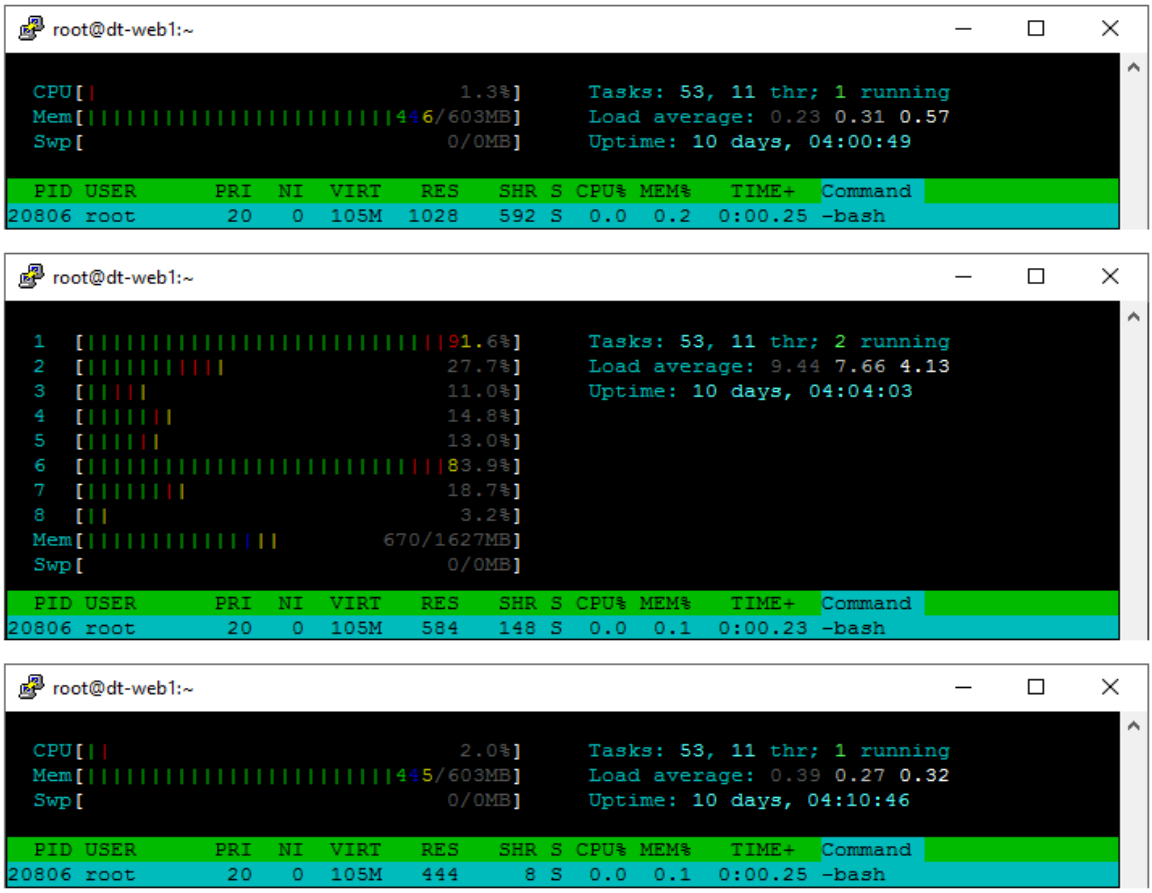
Slika 7.51. - Iskoristivost resursa procesora za drugi eksperiment (ordinata predstavlja postotak opterećenja jezgri gdje 100 predstavlja jednu jezgru, a apscisa vrijeme) u alatu *Munin*

Slika 7.52. predstavlja iskoristivost memorije jednog Web-poslužitelja za drugi eksperiment u alatu *Munin*.



Slika 7.52. - Iskoristivost memorije za drugi eksperiment (ordinata predstavlja memoriju izraženu u GB, a apscisa vrijeme) u alatu *Munin*

U ovom testu početno stanje resursa je također jedna jezgra i 0.6 GB memorije. U trenutku kada se desilo naglo opterećenje povećan je broj jezgri s jedne na osam što je ujedno i maksimalni dopušteni broj po virtualnom Web-poslužitelju zbog ograničenja resursa fizičkog poslužitelja. Memorija je povećana s 0.6 GB na 1.6 GB. Nakon opterećenja resursi su se vratili u prvobitno stanje tj. procesor na jednu jezgru, a memorija na 0.6 GB. Slika 7.53. prikazuje stanje resursa pomoću naredbe *htop* za Web-poslužitelja (dt-web1) nad kojim je ispitivanje izvršeno. Na slici su prikazana tri stanja za drugi eksperiment: prije testa, tijekom najvećeg opterećenja i poslije ispitivanja.



Slika 7.53. - Pregled računalnih resursa poslužitelja (dt-web1) pomoću naredbe *htop* za drugi eksperiment: prije testa (gornja slika), tijekom testa (srednja slika) i nakon testa (donja slika)

Slika 7.53. također pokazuje da nije došlo do prekida u radu Web-poslužitelja tijekom izmjene resursa.

Rezultati za eksperiment 3 - Više Web-klastera

Cilj trećeg testa je provjera da li sustav radi kada je potrebno oduzeti ili dodati resurse na više Web-klastera. Ovo je bitna stvar jer se u praksi (kao i u ovom primjeru) Web-poslužitelji jednog klastera nalaze na više fizičkih poslužitelja, te je bitno provjeriti njihovo ponašanje kada su samo pojedini virtualni Web-poslužitelji pod opterećenjem. Kako bi se to postiglo prvi Web-klaster (*apache*) ima šest puta više simultanih korisnika od drugog Web-klastera (*nginx*). Slika 7.54. predstavlja dio akcija iz glavne datoteke o zapisima prvog fizičkog poslužitelja. Zapisi omogućuju da se sve promjene na poslužiteljima vremenski prate u svrhu bolje analize i usporedbe rezultata s onim dobivenih iz alata za nadzor računalnih resursa.

```
(1.)    2016-11-06 21:39:04:899  ********************
(2.)    2016-11-06 21:40:01:955  VM check STARTED!
(3.)    2016-11-06 21:40:01:959  Get total CPU and RAM assigned to VMs...
(4.)    2016-11-06 21:40:02:137  Total CPU assigned: 3
(5.)    2016-11-06 21:40:02:142  Total RAM assigned: 3145728kB (3.00GB)
(6.)    2016-11-06 21:40:02:147  Start iterating through nodes...
(7.)    2016-11-06 21:40:02:153  START checking VM dt-web1.int.ch!
(8.)    2016-11-06 21:40:02:156  VM manager setup for VM dt-web1.int.ch => CPU min: 1; CPU max: ; RAM min:
        1; RAM max: ; CPU balance logic: ; RAM balance logic:
(9.)    2016-11-06 21:40:02:165  VM manager setup after loading defaults for VM dt-web1.int.ch => CPU min: 1;
        CPU max: 8; RAM min: 1; RAM max: 8; CPU balance logic: 2:1;8:2;16:4; RAM balance logic: 2:1;8:2;16:4
(10.)   2016-11-06 21:40:02:170  Getting current VM setup...
(11.)   2016-11-06 21:40:02:233  Current VM setup => CPU: 1; RAM: 1048576kB
(12.)   2016-11-06 21:40:02:239  Getting VM dt-web1.int.ch current RAM and CPU info...
(13.)   2016-11-06 21:40:03:623  VM dt-web1.int.ch LA(5): 5.53
(14.)   2016-11-06 21:40:03:647  VM dt-web1.int.ch available RAM: 142456kB
(15.)   2016-11-06 21:40:03:660  VM dt-web1.int.ch available RAM: 23.04%
(16.)   2016-11-06 21:40:03:663  Calculating resource usage...
(17.)   2016-11-06 21:40:03:670  CPU usage: 553.00%
(18.)   2016-11-06 21:40:03:674  RAM usage: 76.96%
(19.)   2016-11-06 21:40:03:677  Checking tresholds...
(20.)   2016-11-06 21:40:03:682  CPU usage higher than UP treshold: 553.00 > 90
(21.)   2016-11-06 21:40:03:693  Free RAM lower than limit: 142456kB < 262144kB
(22.)   2016-11-06 21:40:03:697  Calculating new CPU count (Logic: LA)...
(23.)   2016-11-06 21:40:03:715  New CPU count: 7
(24.)   2016-11-06 21:40:03:720  Calculating new RAM size...
(25.)   2016-11-06 21:40:03:740  New RAM size: 2097152
(26.)   2016-11-06 21:40:03:744  Checking host limits...
(27.)   2016-11-06 21:40:03:753  Total VMs CPU count with new dt-web1.int.ch configuration under host limit
(28.)   2016-11-06 21:40:03:758  Total VMs RAM with new dt-web1.int.ch configuration under host limit
(29.)   2016-11-06 21:40:03:763  Setting VM dt-web1.int.ch CPU to 7 cores
(30.)   2016-11-06 21:40:03:851  Setting VM dt-web1.int.ch RAM to 2097152kb
(31.)   2016-11-06 21:40:04:922  Checking dt-web1.int.ch Apache...
(32.)   2016-11-06 21:40:04:935  Apache is listening. VM dt-web1.int.ch OK!
(33.)   2016-11-06 21:40:04:939  Total CPU and RAM assigned to VMs after update...
(34.)   2016-11-06 21:40:04:942  Total CPU assigned: 9
(35.)   2016-11-06 21:40:04:946  Total RAM assigned: 4194304kB (4.00GB)
(36.)   2016-11-06 21:40:04:950  Checking VM dt-web1.int.ch FINNISHED!
(37.)   2016-11-06 21:40:04:954  START checking VM dt-web2.int.ch!
(38.)   2016-11-06 21:40:04:958  VM manager setup for VM dt-web2.int.ch => CPU min: 1; CPU max: ; RAM min:
    1; RAM max: ; CPU balance logic: ; RAM balance logic:
                                        •
                                        •
                                        •
(39.)   2016-11-06 21:40:08:182  New values same as current, skipping VM dt-web6.int.ch update.
(40.)   2016-11-06 21:40:08:185  Checking VM dt-web6.int.ch FINNISHED!
(41.)   2016-11-06 21:40:08:188  Iterating through nodes finnished.
(42.)   2016-11-06 21:40:08:192  VM check FINNISHED!
(43.)   2016-11-06 21:40:08:196  ********************
```

Slika 7.54. - Prikaz dijela zapisa za fizički poslužitelj 2 (host2)

Test je ukupno trajao 10 minuta. Na slici 7.54. se vidi da je sustav prepoznao da je prvi Web-klaster (odnosno dva Web-poslužitelja (dt-web1 i dt-web2)) ima opterećenje procesora (linija 20.). Obavio je dodjelu resursa procesora, odnosno povećao je broj jezgri (linija 29.). Sustav je također prepoznao da memorija nije pod opterećenjem, što je rezultiralo da smanji ukupnu količinu dodijeljene memorije (linija 30.). Ostale Web-poslužitelje koji su se nalazili na tom fizičkom poslužitelju, a nisu dio navedenog Web-klastera nije mijenjao jer nisu bili pod tolikim opterećenjem da bi za to imalo potrebe. Važno je napomenuti da nije došlo do prekida u radu niti jednog Web-poslužitelja neovisno kojem Web-klasteru oni pripadali (linija 32.).

Slika 7.55. prikazuje dijelove akcija iz datoteke o zapisima drugog fizičkog poslužitelja gdje je vidljivo da su resursi procesora dodijeljeni (linija 23.) samo jednom Web-poslužitelju (dt-web3), jer je samo on na ovom fizičkom poslužitelju dio prvog Web-klastera. Njegovo

opterećenje memorije je bilo u zadanim granicama tako da nije došlo do promjene (linija 21.). Ovdje također nije došlo da prekida u radu (linija 29.) ostalih Web-poslužitelja, odnosno svi imaju upravo onoliko resursa koliko im je dovoljno za pravilan rad. Sljedeća slika predstavlja dio akcija iz glavne datoteke o zapisima drugog fizičkog poslužitelja.

```
(1.)    2016-11-06 21:40:01:804  ********************
(2.)    2016-11-06 21:40:01:865  VM check STARTED!
(3.)    2016-11-06 21:40:01:870  Get total CPU and RAM assigned to VMs...
(4.)    2016-11-06 21:40:02:133  Total CPU assigned: 3
(5.)    2016-11-06 21:40:02:138  Total RAM assigned: 3145728kB (3.00GB)
(6.)    2016-11-06 21:40:02:143  Start iterating through nodes...
(7.)    2016-11-06 21:40:02:147  START checking VM dt-web3.int.ch!
(8.)    2016-11-06 21:40:02:152  VM manager setup for VM dt-web3.int.ch => CPU min: 1; CPU max: ; RAM min: 1;
        RAM max: ; CPU balance logic: ; RAM balance logic:
(9.)    2016-11-06 21:40:02:162  VM manager setup after loading defaults for VM dt-web3.int.ch => CPU min: 1; CPU
        max: 8; RAM min: 1; RAM max: 8; CPU balance logic: 2:1;8:2;16:4; RAM balance logic: 2:1;8:2;16:4
(10.)   2016-11-06 21:40:02:167  Getting current VM setup...
(11.)   2016-11-06 21:40:02:251  Current VM setup => CPU: 1; RAM: 1048576kB
(12.)   2016-11-06 21:40:02:256  Getting VM dt-web3.int.ch current RAM and CPU info...
(13.)   2016-11-06 21:40:03:630  VM dt-web3.int.ch LA(5): 3.66
(14.)   2016-11-06 21:40:03:662  VM dt-web3.int.ch available RAM: 323204kB
(15.)   2016-11-06 21:40:03:678  VM dt-web3.int.ch available RAM: 52.29%
(16.)   2016-11-06 21:40:03:683  Calculating resource usage...
(17.)   2016-11-06 21:40:03:692  CPU usage: 366.00%
(18.)   2016-11-06 21:40:03:697  RAM usage: 47.71%
(19.)   2016-11-06 21:40:03:701  Checking tresholds...
(20.)   2016-11-06 21:40:03:708  CPU usage higher than UP treshold: 366.00 > 90
(21.)   2016-11-06 21:40:03:721  RAM usage within optimum range: 40 - 80; Free (kb): 323204 (limit: 262144)
(22.)   2016-11-06 21:40:03:727  Calculating new CPU count (Logic: LA)...
(23.)   2016-11-06 21:40:03:745  New CPU count: 5
(24.)   2016-11-06 21:40:03:750  Checking host limits...
(25.)   2016-11-06 21:40:03:762  Total VMs CPU count with new dt-web3.int.ch configuration under host limit
(26.)   2016-11-06 21:40:03:768  Total VMs RAM with new dt-web3.int.ch configuration under host limit
(27.)   2016-11-06 21:40:03:773  Setting VM dt-web3.int.ch CPU to 5 cores
(28.)   2016-11-06 21:40:04:878  Checking dt-web3.int.ch Apache...
(29.)   2016-11-06 21:40:04:890  Apache is listening. VM dt-web3.int.ch OK!
(30.)   2016-11-06 21:40:04:895  Total CPU and RAM assigned to VMs after update...
(31.)   2016-11-06 21:40:04:899  Total CPU assigned: 7
(32.)   2016-11-06 21:40:04:904  Total RAM assigned: 3145728kB (3.00GB)
(33.)   2016-11-06 21:40:04:908  Checking VM dt-web3.int.ch FINNISHED!
(34.)   2016-11-06 21:40:04:913  START checking VM dt-web4.int.ch!
(35.)   2016-11-06 21:40:04:917  VM manager setup for VM dt-web4.int.ch => CPU min: ; CPU max: ; RAM min: ;
    RAM max: ; CPU balance logic: ; RAM balance logic:
                                                    •
                                                    •
                                                    •
(36.)   2016-11-06 21:40:05:834  New values same as current, skipping VM dt-web5.int.ch update.
(37.)   2016-11-06 21:40:05:838  Checking VM dt-web5.int.ch FINNISHED!
(38.)   2016-11-06 21:40:05:843  Iterating through nodes finnished.
(39.)   2016-11-06 21:40:05:847  VM check FINNISHED!
(40.)   2016-11-06 21:40:05:852  ********************
```

Slika 7.55. - Prikaz dijela zapisa za fizički poslužitelj 3 (host3)

Rezultati provedenih ispitivanja potvrđuju prvu hipotezu, odnosno primjena predloženog modela nad virtualiziranim sustavima omogućava dodavanje/oduzimanje postojećih računalnih resursa (CPU i memorija) bez potrebe za ponovnim pokretanjem poslužitelja.

7.5.7.2 Faza II - Rezultat validacije druge hipoteze

U ovoj fazi su predstavljeni rezultati koji se odnose na provjeru konzistentnosti u radu poslužitelja. Proces pristupanju resursa (CPU i memorije) neće biti posebno dokazivan jer je dio složenijeg procesa kao što je oduzimanje i dodavanje resursa. Slika 7.56. predstavlja djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno oduzimanje

resursa procesora (linija 7., 14., 17. i 23.), nakon čega je također potvrđeno da Web-poslužitelj radi ispravno (linija 25.). Drugim riječima virtualni Web-poslužitelj je odgovorio s HTTP kodom 200 čime je dokazano da oduzimanjem memorije nije narušen rad Web-poslužitelja.

```
                                                    •
                                                    •
                                                    •
(1.)    2017-09-14 10:55:38:369   START checking VM dt-web2.int.ch!
(2.)    2017-09-14 10:55:38:372   VM manager setup for VM dt-web2.int.ch => CPU min: 1; CPU max: ; RAM min: 1; RAM max:
        ; CPU balance logic: ; RAM balance logic:
(3.)    2017-09-14 10:55:38:380   VM manager setup after loading defaults for VM dt-web2.int.ch => CPU min: 1; CPU max: 8;
        RAM min: 1; RAM max: 8; CPU balance logic: 2:1;8:2;16:4; RAM balance logic: 2:1;8:2;16:4
(4.)    2017-09-14 10:55:38:384   Getting current VM setup...
(5.)    2017-09-14 10:55:38:439   Current VM setup => CPU: 2; RAM: 1048576kB
(6.)    2017-09-14 10:55:38:443   Getting VM dt-web2.int.ch current RAM and CPU info...
(7.)    2017-09-14 10:55:38:607   VM dt-web2.int.ch LA(5): 0.84
(8.)    2017-09-14 10:55:38:638   VM dt-web2.int.ch available RAM: 215416kB
(9.)    2017-09-14 10:55:38:650   VM dt-web2.int.ch available RAM: 34.85%
(10.)   2017-09-14 10:55:38:656   Calculating resource usage...
(11.)   2017-09-14 10:55:38:664   CPU usage: 42.00%
(12.)   2017-09-14 10:55:38:667   RAM usage: 65.15%
(13.)   2017-09-14 10:55:38:671   Checking tresholds...
(14.)   2017-09-14 10:55:38:679   CPU usage lower than DOWN treshold: 42.00 < 80
(15.)   2017-09-14 10:55:38:685   RAM usage higher than UP treshold: 65.15 > 65
(16.)   2017-09-14 10:55:38:690   Calculating new CPU count (Logic: LA)...
(17.)   2017-09-14 10:55:38:704   New CPU count: 1
(18.)   2017-09-14 10:55:38:709   Calculating new RAM size...
(19.)   2017-09-14 10:55:38:730   New RAM size: 2097152
(20.)   2017-09-14 10:55:38:736   Checking host limits...
(21.)   2017-09-14 10:55:38:745   Total VMs CPU count with new dt-web2.int.ch configuration under host limit
(22.)   2017-09-14 10:55:38:751   Total VMs RAM with new dt-web2.int.ch configuration under host limit
(23.)   2017-09-14 10:55:38:755   Setting VM dt-web2.int.ch CPU to 1 cores
(24.)   2017-09-14 10:55:38:823   Setting VM dt-web2.int.ch RAM to 2097152kb
(25.)   2017-09-14 10:55:39:853   Checking dt-web2.int.ch Apache...
(26.)   2017-09-14 10:55:39:863   Apache is listening. VM dt-web2.int.ch OK!
(27.)   2017-09-14 10:55:39:867   Total CPU and RAM assigned to VMs after update...
(28.)   2017-09-14 10:55:39:870   Total CPU assigned: 3
(29.)   2017-09-14 10:55:39:874   Total RAM assigned: 4194304kB (4.00GB)
(30.)   2017-09-14 10:55:39:881   Checking VM dt-web2.int.ch FINISHED!
                                                    •
                                                    •
                                                    •
```

Slika 7.56. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno oduzimanje resursa procesora nakon čega je potvrđeno da Web-poslužitelj radi ispravno

Slika 7.57. predstavlja djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno dodavanje resursa procesora (linija 7., 14., 17. i 21.), nakon čega je također potvrđeno da Web-poslužitelj radi ispravno (linija 23.).

•
•
•
•

(1.) 2017-09-14 10:55:27:989 START checking VM dt-web2.int.ch!
(2.) 2017-09-14 10:55:27:993 VM manager setup for VM dt-web2.int.ch => CPU min: 1; CPU max: ; RAM min: 1; RAM max: ; CPU balance logic: ; RAM balance logic:
(3.) 2017-09-14 10:55:28:001 VM manager setup after loading defaults for VM dt-web2.int.ch => CPU min: 1; CPU max: 8; RAM min: 1; RAM max: 8; CPU balance logic: 2:1;8:2;16:4; RAM balance logic: 2:1;8:2;16:4
(4.) 2017-09-14 10:55:28:005 Getting current VM setup...
(5.) 2017-09-14 10:55:28:062 Current VM setup => CPU: 1; RAM: 1048576kB
(6.) 2017-09-14 10:55:28:065 Getting VM dt-web2.int.ch current RAM and CPU info...
(7.) 2017-09-14 10:55:28:231 VM dt-web2.int.ch LA(5): 0.99
(8.) 2017-09-14 10:55:28:262 VM dt-web2.int.ch available RAM: 216944kB
(9.) 2017-09-14 10:55:28:273 VM dt-web2.int.ch available RAM: 35.10%
(10.) 2017-09-14 10:55:28:279 Calculating resource usage...
(11.) 2017-09-14 10:55:28:287 CPU usage: 99.00%
(12.) 2017-09-14 10:55:28:290 RAM usage: 64.90%
(13.) 2017-09-14 10:55:28:294 Checking tresholds...
(14.) 2017-09-14 10:55:28:300 CPU usage higher than UP treshold: 99.00 > 90
(15.) 2017-09-14 10:55:28:310 RAM usage within optimum range: 40 - 65; Free (kb): 216944 (limit: 204800)
(16.) 2017-09-14 10:55:28:317 Calculating new CPU count (Logic: LA)...
(17.) 2017-09-14 10:55:28:330 New CPU count: 2
(18.) 2017-09-14 10:55:28:334 Checking host limits...
(19.) 2017-09-14 10:55:28:344 Total VMs CPU count with new dt-web2.int.ch configuration under host limit
(20.) 2017-09-14 10:55:28:350 Total VMs RAM with new dt-web2.int.ch configuration under host limit
(21.) 2017-09-14 10:55:28:353 Setting VM dt-web2.int.ch CPU to 2 cores
(22.) 2017-09-14 10:55:29:426 Checking dt-web2.int.ch Apache...
(23.) 2017-09-14 10:55:29:436 Apache is listening. VM dt-web2.int.ch OK!
(24.) 2017-09-14 10:55:29:440 Total CPU and RAM assigned to VMs after update...
(25.) 2017-09-14 10:55:29:443 Total CPU assigned: 4
(26.) 2017-09-14 10:55:29:446 Total RAM assigned: 3145728kB (3.00GB)
(27.) 2017-09-14 10:55:29:454 Checking VM dt-web2.int.ch FINISHED!

•
•
•

Slika 7.57. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno dodavanja resursa procesora nakon čega je potvrđeno da Web-poslužitelj radi ispravno

Nakon što je provjereno oduzimanje i dodavanje resursa procesora, slijedi ista provjera za memoriju. Slika 7.58. predstavlja djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno oduzimanje memorije (linija 8., 15., 19. i 24.), nakon čega je potvrđeno da Web-poslužitelj radi ispravno (linija 26.).

•
•
•

(1.) 2017-09-14 12:28:36:971 START checking VM dt-web2.int.ch!
(2.) 2017-09-14 12:28:36:974 VM manager setup for VM dt-web2.int.ch => CPU min: 1; CPU max: ; RAM min: 1; RAM max: ; CPU balance logic: ; RAM balance logic:
(3.) 2017-09-14 12:28:36:983 VM manager setup after loading defaults for VM dt-web2.int.ch => CPU min: 1; CPU max: 8; RAM min: 1; RAM max: 8; CPU balance logic: 2:1;8:2;16:4; RAM balance logic: 2:1;8:2;16:4
(4.) 2017-09-14 12:28:36:986 Getting current VM setup...
(5.) 2017-09-14 12:28:37:041 Current VM setup => CPU: 1; RAM: 2102272kB
(6.) 2017-09-14 12:28:37:045 Getting VM dt-web2.int.ch current RAM and CPU info...
(7.) 2017-09-14 12:28:37:185 VM dt-web2.int.ch LA(5): 0.44
(8.) 2017-09-14 12:28:37:217 VM dt-web2.int.ch available RAM: 1249888kB
(9.) 2017-09-14 12:28:37:229 VM dt-web2.int.ch available RAM: 74.76%
(10.) 2017-09-14 12:28:37:235 Calculating resource usage...
(11.) 2017-09-14 12:28:37:242 CPU usage: 44.00%
(12.) 2017-09-14 12:28:37:246 RAM usage: 25.24%
(13.) 2017-09-14 12:28:37:249 Checking tresholds...
(14.) 2017-09-14 12:28:37:257 CPU usage lower than DOWN treshold: 44.00 < 80
(15.) 2017-09-14 12:28:37:265 RAM usage lower than DOWN treshold: 25.24 < 30
(16.) 2017-09-14 12:28:37:270 Calculating new CPU count (Logic: LA)...
(17.) 2017-09-14 12:28:37:283 New CPU count: 1
(18.) 2017-09-14 12:28:37:288 Calculating new RAM size...
(19.) 2017-09-14 12:28:37:316 New RAM size: 1053696
(20.) 2017-09-14 12:28:37:322 Checking host limits...
(21.) 2017-09-14 12:28:37:332 Total VMs CPU count with new dt-web2.int.ch configuration under host limit
(22.) 2017-09-14 12:28:37:338 Total VMs RAM with new dt-web2.int.ch configuration under host limit
(23.) 2017-09-14 12:28:37:342 Setting VM dt-web2.int.ch CPU to 1 cores
(24.) 2017-09-14 12:28:37:422 Setting VM dt-web2.int.ch RAM to 1053696kb
(25.) 2017-09-14 12:28:38:455 Checking dt-web2.int.ch Apache...
(26.) 2017-09-14 12:28:38:465 Apache is listening. VM dt-web2.int.ch OK!
(27.) 2017-09-14 12:28:38:468 Total CPU and RAM assigned to VMs after update...
(28.) 2017-09-14 12:28:38:472 Total CPU assigned: 3
(29.) 2017-09-14 12:28:38:476 Total RAM assigned: 4199424kB (4.00GB)
(30.) 2017-09-14 12:28:38:483 Checking VM dt-web2.int.ch FINISHED!

•
•
•

Slika 7.58. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno oduzimanje memorije nakon čega je potvrđeno da Web-poslužitelj radi ispravno

Slika 7.59. predstavlja djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno dodavanje memorije (linija 8., 15., 19. i 24.), nakon čega je također potvrđeno da Web-poslužitelj radi ispravno (linija 26.).

•
•
•

(1.) 2017-09-14 12:28:32:347 START checking VM dt-web2.int.ch!
(2.) 2017-09-14 12:28:32:351 VM manager setup for VM dt-web2.int.ch => CPU min: 1; CPU max: ; RAM min: 1; RAM max: ; CPU balance logic: ; RAM balance logic:
(3.) 2017-09-14 12:28:32:359 VM manager setup after loading defaults for VM dt-web2.int.ch => CPU min: 1; CPU max: 8; RAM min: 1; RAM max: 8; CPU balance logic: 2:1;8:2;16:4; RAM balance logic: 2:1;8:2;16:4
(4.) 2017-09-14 12:28:32:363 Getting current VM setup...
(5.) 2017-09-14 12:28:32:419 Current VM setup => CPU: 1; RAM: kB
(6.) 2017-09-14 12:28:32:423 Getting VM dt-web2.int.ch current RAM and CPU info...
(7.) 2017-09-14 12:28:32:565 VM dt-web2.int.ch LA(5): 0.48
(8.) 2017-09-14 12:28:32:597 VM dt-web2.int.ch available RAM: 201320kB
(9.) 2017-09-14 12:28:32:608 VM dt-web2.int.ch available RAM: 32.30%
(10.) 2017-09-14 12:28:32:614 Calculating resource usage...
(11.) 2017-09-14 12:28:32:622 CPU usage: 48.00%
(12.) 2017-09-14 12:28:32:625 RAM usage: 67.70%
(13.) 2017-09-14 12:28:32:628 Checking tresholds...
(14.) 2017-09-14 12:28:32:636 CPU usage lower than DOWN treshold: 48.00 < 80
(15.) 2017-09-14 12:28:32:645 Free RAM lower than limit: 201320kB < 204800kB
(16.) 2017-09-14 12:28:32:650 Calculating new CPU count (Logic: LA)...
(17.) 2017-09-14 12:28:32:663 New CPU count: 1
(18.) 2017-09-14 12:28:32:668 Calculating new RAM size...
(19.) 2017-09-14 12:28:32:689 New RAM size: 2102272
(20.) 2017-09-14 12:28:32:695 Checking host limits...
(21.) 2017-09-14 12:28:32:705 Total VMs CPU count with new dt-web2.int.ch configuration under host limit
(22.) 2017-09-14 12:28:32:711 Total VMs RAM with new dt-web2.int.ch configuration under host limit
(23.) 2017-09-14 12:28:32:715 Setting VM dt-web2.int.ch CPU to 1 cores
(24.) 2017-09-14 12:28:32:788 Setting VM dt-web2.int.ch RAM to 2102272kb
(25.) 2017-09-14 12:28:33:856 Checking dt-web2.int.ch Apache...
(26.) 2017-09-14 12:28:33:866 Apache is listening. VM dt-web2.int.ch OK!
(27.) 2017-09-14 12:28:33:869 Total CPU and RAM assigned to VMs after update...
(28.) 2017-09-14 12:28:33:873 Total CPU assigned: 3
(29.) 2017-09-14 12:28:33:876 Total RAM assigned: 4199424kB (4.00GB)
(30.) 2017-09-14 12:28:33:883 Checking VM dt-web2.int.ch FINISHED!

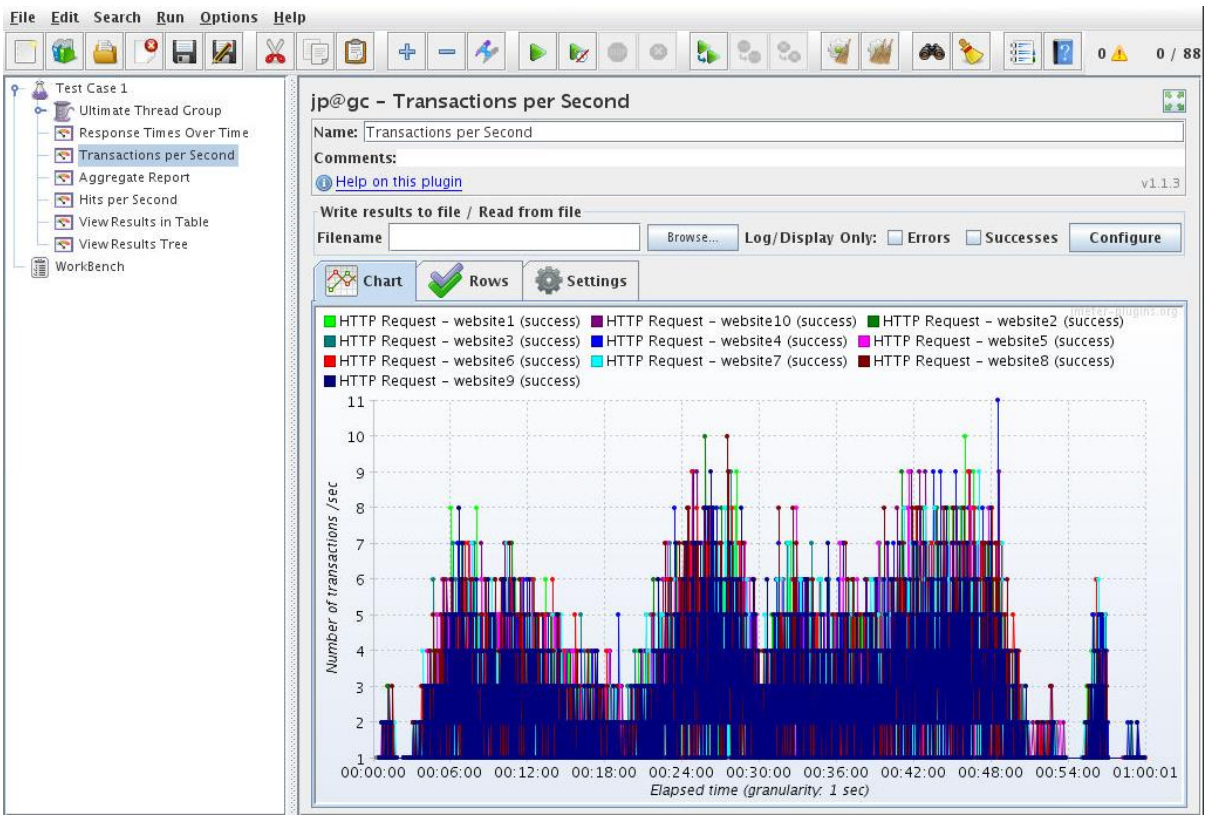
•
•
•

Slika 7.59. - Djelomični prikaz zapisa s istaknutim dijelovima u kojima se vidi da je izvršeno dodavanje memorije nakon čega je potvrđeno da Web-poslužitelj radi ispravno

Slika 7.59. ujedno predstavlja i posljednju stavku koju je potrebno provjeriti ako želimo provjeriti konzistentan rad poslužitelja. Iz svega navedenog može se zaključiti da rezultati provedenih ispitivanja potvrđuju drugu hipotezu, odnosno primjena predloženog modela ne narušava konzistentan rad poslužitelja.

7.5.7.3 Faza III - Rezultat validacije treće hipoteze

U ovo fazi su prikazani rezultati koji su služili za provjeru da li primjena novog modela povećava iskoristivost postojećih računalnih resursa s obzirom na prvobitno rješenje. Slika 7.60. predstavlja broj izvršenih zahtjeva u vremenu za eksperiment koji je definiran na slici 7.45. Test je ukupno trajao 60 minuta.

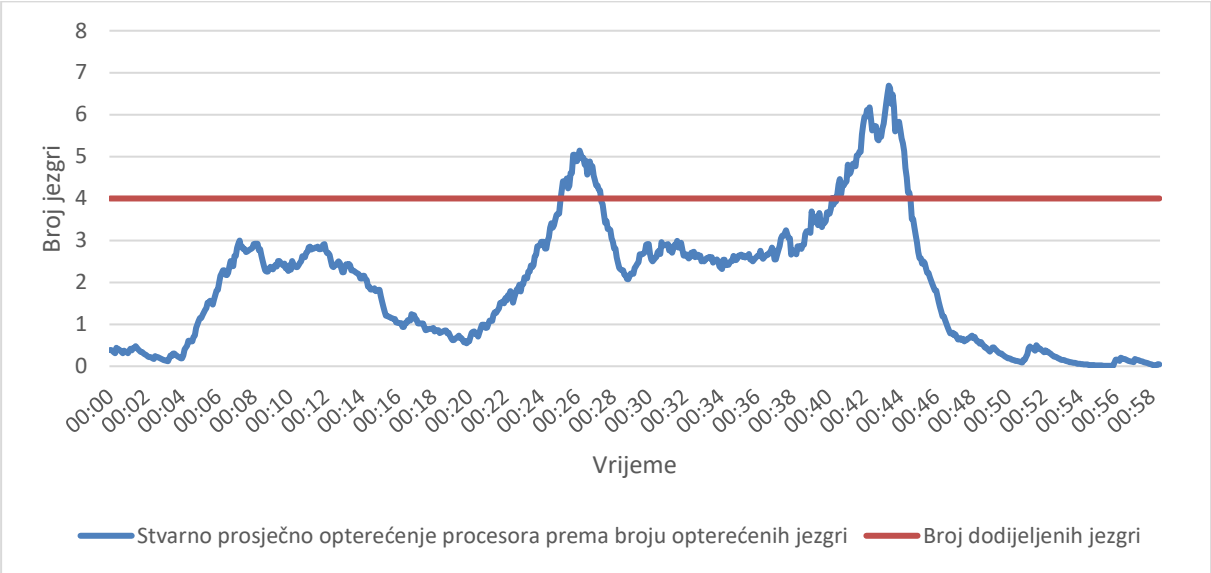


Slika 7.60. - Broj zahtjeva u sekundi za izvršeni eksperiment u alatu *Apache JMeter* (ordinata predstavlja broj izvršenih zahtjeva u sekundi, a apscisa vrijeme trajanja eksperimenta)

Na slici 7.60. se vidi da se definirani parametri sa slike 7.45. poklapaju s prikazom izvršenih zahtjeva, što je bilo i za očekivati. Navedeni eksperiment je ponovljen dva puta (bez i s primjenom aplikacije koja je razvijena na temelju novog modela) kako bi provjerili omogućava li primjena novog modela bolju iskoristivost postojećih računalnih resursa (CPU i memorija). Prvo su prikazani rezultati testova bez primjene nove aplikacije.

U nastavku se promatra opterećenje samo jednog virtualnog Web-poslužitelja jer je riječ o Web-klasteru, odnosno arhitekturi koja se sastoji od sustava za raspodjelu HTTP/HTTPS zahtjeva koja uz pomoć algoritma osigurava da svaki Web-poslužitelj dobije jednak broj zahtjeva.

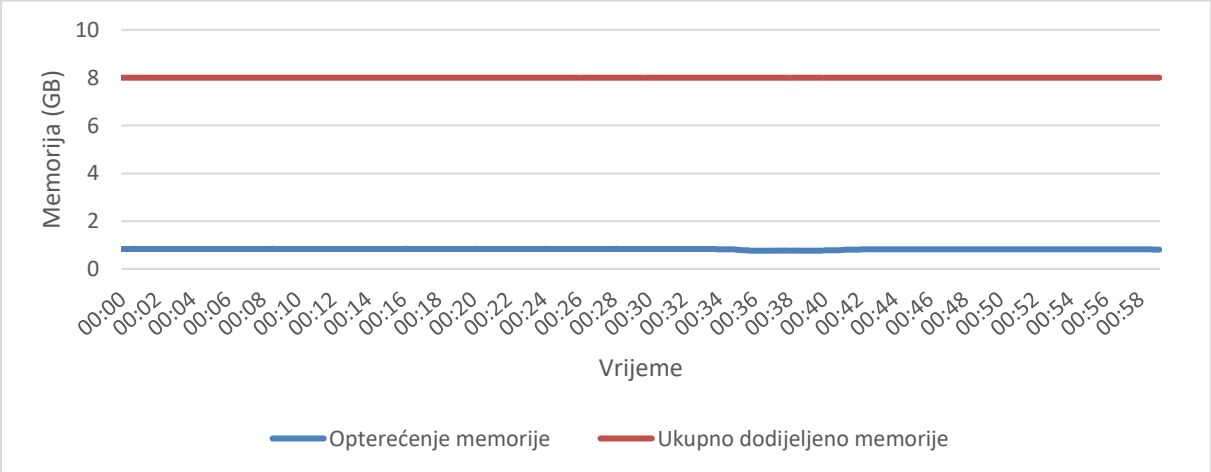
Slika 7.61. predstavlja prikaz stvarnog prosječnog opterećenja procesora prema broju opterećenih jezgri po naredbi *htop* za izvršeni eksperiment bez primjene aplikacije koja je razvijena na temelju novog modela. Promatrani Web-poslužitelj tijekom cijelog testa ima četiri jezgre jer nije primijenjena aplikacija za promjenu resursa.



Slika 7.61. - Prikaz stvarnog prosječnog opterećenja procesora prema broju opterećenih jezgri po naredbi *htop* za izvršeni eksperiment bez primjene aplikacije koja je razvijena na temelju novog modela

Slika 7.61. potvrđuje glavnu problematiku ovog istraživanja, a to je da postoje periodi kada postojeći dodijeljeni računalni resursi (u ovom slučaju procesor) nisu dovoljno iskorišteni. Također su prisutni dva perioda kada je sustav zagušen, odnosno dodijeljeni broj jezgri nije bio dovoljan za obradu broja korisničkih zahtjeva, što rezultira da sustav radi sporije. Već je napomenuto na početku sedmog poglavlja da su prema naredbi *htop* moguća opterećenja procesora iznad 100%, jer je riječ o prosječnom opterećenje procesora u određenom vremenskom periodu (u ovom slučaju jedan minut), a ne stvarnom opterećenju jer to nije moguće.

Slika 7.62. predstavlja prikaz opterećenja memorije i dodijeljene memorije prema naredbi *htop* za izvršeni eksperiment bez primjene aplikacije koja je razvijena na temelju novog modela.



Slika 7.62. - Prikaz opterećenja memorije i dodijeljene memorije prema naredbi *htop* za izvršeni eksperiment bez primjene aplikacije koja je razvijena na temelju novog modela

Na slici 7.62. opterećenje memorije je prikazano u GB kako bi se što jasnije vidjelo koliko resursa nikad nije iskorišteno. Razlog zbog kojeg oscilacije u opterećenju memorije nisu

vidljive na slici je to što su one izražene u MB. Memorija nije pod znatnijim opterećenjem jer su korištene generičke Web-stranice kako bi se test mogao što lakše ponoviti u svrhu daljnjeg istraživanja i unapređenja modela. Na temelju slike 7.62. se može zaključiti da sustav nije nikad pod opterećenjem. Međutim, takvo rješenje nije dobro jer su prisutni resursi (u ovom slučaju memorija) koja je zauzeta od strane navedenog poslužitelja i ne može biti dio drugih sustava (poslužitelja), što također potvrđuje opravdanost ovog istraživanja.

U nastavku istraživanja slijede rezultati eksperimenta u kojem je primijenjena aplikacija koja je razvijena na temelju novog modela. Važno je napomenuti da su u ovom slučaju korišteni isti parametri eksperimenta (broj simultanih korisnika u periodu od šezdeset minuta) kao i za prvi slučaj. Model je razvijen na način da je prilagodljiv sustavima, odnosno administrator odlučuje koji su ulazni parametri sustava (globalni parametri, početni parametri, itd.). Slika 7.63. predstavlja konkretan primjer ulaznih parametra modela koji su korišteni u ovom eksperimentu.

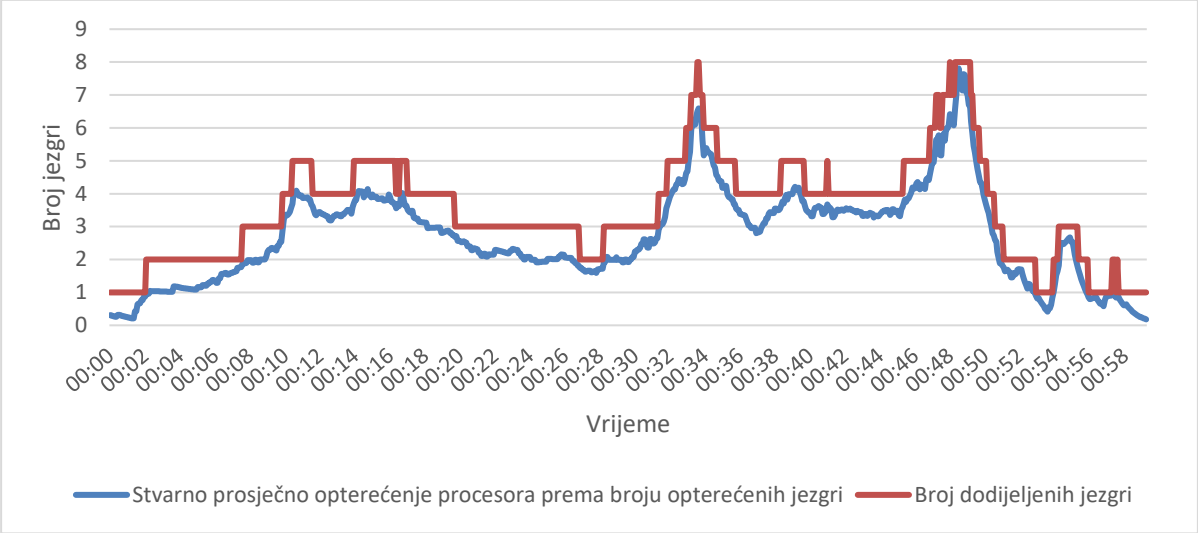
```
(1.)  VM_LA_LOGIC='LA'
(2.)  VM_LA_MODIFICATOR='1.10'
(3.)  VM_LA_ROUNDING='UP'
(4.)  VM_LA_TRESHOLD_UP='90'
(5.)  VM_LA_TRESHOLD_DOWN='80'
(6.)  VM_RAM_TRESHOLD_UP='70'
(7.)  VM_RAM_TRESHOLD_DOWN='30'
```

Slika 7.63. - Ulazni parametri modela koji su korišteni u eksperimentu

Svi parametri su detaljno opisani u poglavlju 7.3. U ovom slučaju korištena je LA metoda za dodjelu resursa (linija 1.), koja je moguća samo za procesor jer se kod njega mjeri prosječno opterećenje. Za memoriju se uvijek koristi INCREMENT metoda zbog čega nije naveden parametar koji to posebno naglašava. Linija 2. predstavlja multiplikator kojeg koristi LA metoda, odnosno tu vrijednost množi s trenutnim opterećenjem procesora i na temelju toga odjeljuje nove resurse. U liniji 3. je definirana metoda zaokruživanja dobivenog rezultata nakon množenja, što je u ovom slučaju zaokruživanje na višu vrijednost (UP). Sljedeća dva parametra (linija 4. i 5.) predstavljaju maksimalno odnosno minimalno dozvoljeno opterećenje resursa procesora (izraženo u %). Zadnja dva parametra sa slike 7.63. (linija 6. i 7.) predstavljaju maksimalno odnosno minimalno dozvoljeno opterećenje memorije (izraženo u %).

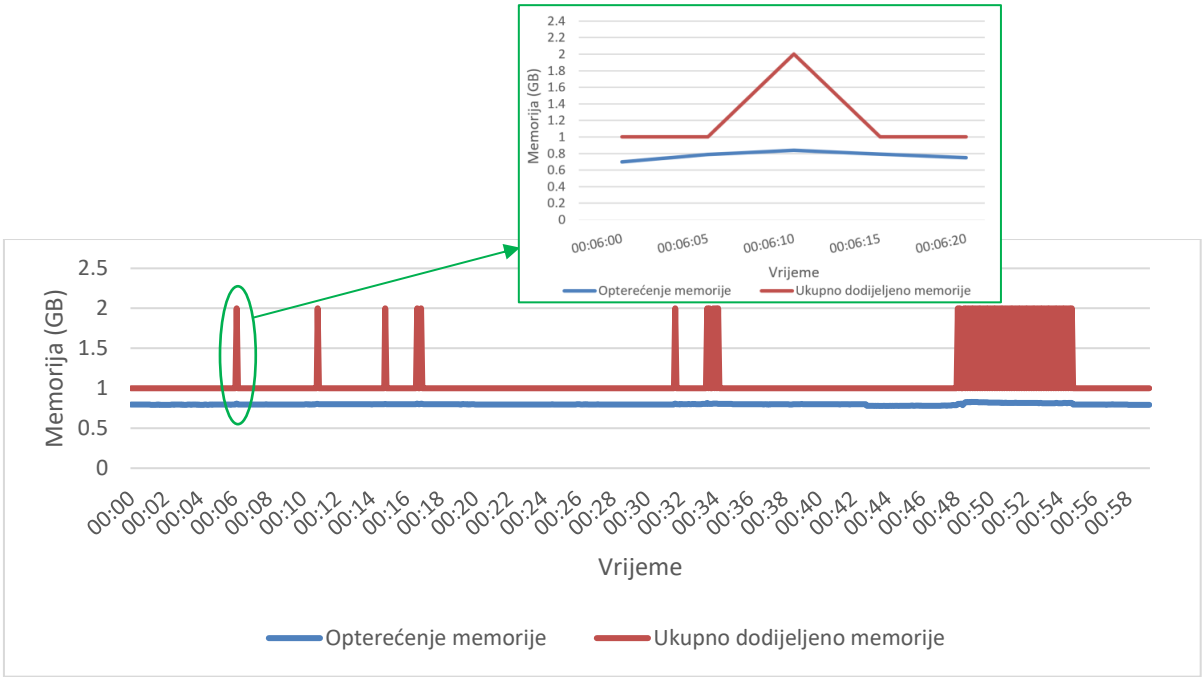
S obzirom da administrator sustava ima slobodu definiranja ulaznih parametara, oni su podešeni na način da što više iskoriste procesor, ali i da izvrše povećanje odnosno smanjenje memorije u određenom trenutku kako bi se još jednom dokazala tvrdnja iz prve faze, odnosno da je moguće dodavanje i oduzimanje ne samo procesora (broja jezgri) već i memorije. Slika 7.64. predstavlja prikaz stvarnog prosječnog opterećenja procesora prema broju opterećenih

jezgri po naredbi *htop* za izvršeni eksperiment s primjenom aplikacije koja je razvijena na temelju novog modela.



Slika 7.64. - Prikaz stvarnog prosječnog opterećenja procesora prema broju opterećenih jezgri po naredbi *htop* za izvršeni eksperiment s primjenom aplikacije koja je razvijena na temelju novog modela

Na slici 7.64. je vidljivo da je tijekom cijelog eksperimenta dodijeljen dovoljan broj jezgri za izvršenje korisničkih zahtjeva, odnosno nisu prisutni periodi kada je procesor neiskorišten ili preopterećen. Slika 7.65. predstavlja prikaz opterećenja memorije i dodijeljene memorije prema naredbi *htop* za izvršeni eksperiment s primjenom aplikacije koja je razvijena na temelju novog modela. Na slici je također prikazan uvećani dio (označeno zelenim) kako bi se bolje vidjeli razlozi zbog kojih je došlo do povećanja memorije. Do promjene resursa je došlo jer je gornja granica dozvoljenog opterećenja bila iznad dozvoljene (0.8 GB).



Slika 7.65. - Prikaz opterećenja memorije i dodijeljene memorije prema naredbi *htop* za izvršeni eksperiment s primjenom aplikacije koja je razvijena na temelju novog modela

Iako je već napomenuto da promatrani sustav, odnosno Web-stranice ne zahtijevaju veću količinu memorije, iz rezultata je vidljivo da se ona puno učinkovitije iskorištava, odnosno odstupanja su ispod dva GB (tako definirano od strane administratora), što je puno bolje od slučaja kada se ne primjenjuje aplikacija koja je razvijena na temelju novog modela. Periodi u kojima se desilo povećanje odnosno smanjenje memorije su namjerno podešeni korigiranjem ulaznih parametara (gornje i donje dozvoljene granice opterećenja) kako bi se još jednom dokazalo da je moguće dodavanje odnosno smanjivanje memorije tijekom rada poslužitelja.

Iz svega navedenog može se zaključiti da rezultati provedenih ispitivanja potvrđuju treću hipotezu, odnosno primjenom predloženog modela nad virtualiziranim sustavima postiže se bolja iskoristivost postojećih računalnih resursa (CPU i memorija) s obzirom na prvobitno stanje.

Prema Microsoftovom priručniku za testiranje Web-aplikacija u sedmom koraku je također potrebno navesti preporuke za buduća poboljšanja. Ona su navedena u osmom poglavlju ovog istraživanja.

7.6 Komunikacija

Zadnji korak prema istraživačkoj paradigmi znanosti o dizajnu predstavlja komunikacija, odnosno proces u kojem je važno istraživanje predstaviti javnosti. Dijelovi ovog istraživanja su već predstavljeni u obliku radova u časopisu i na međunarodnoj konferenciji [22][23][96].

8 Zaključak i smjernice za buduće istraživanje

Danas gotovo da i ne postoji poslovni sustav koji nema dodira s računarstvom u oblaku. Kako bi takvi poslovni sustavi i rješenja mogli postojati, potrebna je složena infrastruktura kao što su podatkovni centri, čiji je glavni cilj osigurati njihov neprekidni rad. Neprestane investicije u infrastrukturu podatkovnih centara na svim razinama predstavljaju jedan od načina kako je to moguće postići (strujne instalacije, neprekidni izvor energije, klima uređaji, mrežna infrastruktura i sl.). Kako bi se osigurala visoka razina dostupnosti podatkovnog centra potrebno je imati i redundantna rješenja na svim razinama infrastrukture kao što su: struja (UPS i agregat), Internet poveznice, druga geografska kolokacija i sl. Osim složene infrastrukture, danas su prisutna i sve složenija IT rješenja koja dodatno povećavaju složenost cijelog sustava jednog podatkovnog centra. Sve navedene stavke generiraju jako velike CAPEX i OPEX troškove zbog čega se nastoji napraviti ušteda na svim razinama, kako kod infrastrukture tako i kod servisa.

Tijekom istraživanja utvrđeno je da su poslužitelji jedan od najvećih uzročnika visokih troškova podatkovnog centra. Daljnjom analizom utvrđeno je da postoje sustavi kao što su Web-farme odnosno njihovi Web-klasteri, gdje su postojeći računalni resursi nedovoljno iskorišteni, što je ujedno i jedan od motiva da se izgradi model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa. Takvo rješenje bi trebalo rezultirati lakšem održavanju sustava, ali i velikim financijskim uštedama.

Tijekom detaljnog proučavanja dosadašnjih znanstvenih istraživanja, ali i rješenja iz prakse, utvrđeno je da ne postoji rješenje koje bi dovoljno učinkovito riješilo ovaj problem što je ujedno i motivacija ovog istraživanja. Nedostatci postojećih rješenja je to što se oni ne bave kako učinkovitije iskoristiti postojeće resurse već u kritičnim situacijama dodaju nove ili vrše migraciju virtualnih poslužitelja na druge fizičke poslužitelje za što je potrebno još više računalnih resursa. Drugi pristup rješavanju ovog problema je prioritizacija procesa, odnosno da se poslužiteljima kojima su prijeko potrebni resursi dodijeli najveći prioritet u izvršavanju procesa. Nedostatak ovog pristupa je što se resursi ne mogu povećati ili smanjiti već samo prioritizirati što i dalje rezultira da su prisutni resursi koji se ne koriste. Nedostatak postojećih rješenja je što nije moguće obaviti dodavanja i oduzimanje računalnih resursa (CPU i memorije) bez potrebe za ponovnim pokretanjem poslužitelja. Veliki broj postojećih rješenja ima fokus samo na CPU ili memoriju, ali ne i na oboje. Jedan od nedostataka postojećih rješenja je taj što ne postoji niti jedan oblik napredne raspodjele resursa koji bi omogućio da se resursi još brže oduzimaju, odnosno dodaju u kritičnim trenutcima. Navedeni razlozi indirektno djeluju na postizanje još veće dostupnosti sustava, ali i višestruke iskoristivosti postojećih resursa.

U ovoj disertaciji se predlaže novi model za automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa bez potrebe za ponovnim pokretanjem poslužitelja. Na temelju novog modela razvijena je aplikacija pomoću koje je izvršena validacija modela na primjeru Web-poslužitelja. Komponente i njihovi mehanizmi od kojih se sastoji novi model su omogućili da aplikacija obavlja automatiziranu i poboljšanu iskoristivost postojećih računalnih resursa (CPU i memorija) bez da je potrebno ponovno pokretanje poslužitelja. Upravo je to omogućilo da se uspješno potvrde sve tri navedene hipoteze. Za razliku od postojećih rješenja, novo rješenje je učinkovitije i neovisno o broju korisničkih zahtjeva, tj. jedino ograničenje u ovom slučaju su resursi samog fizičkog poslužitelja, što je i za očekivati.

Dvije su ciljne skupine ovog istraživanja. Prva su male i srednje tvrtke koje se bave IT uslugama, ali si ne mogu priuštiti skupa komercijalna rješenja, koja najčešće rezultiraju još većom potrošnjom računalnih resursa, a druga ciljna skupina je akademska zajednica u svrhu daljnjeg istraživanja na ovu temu. Upravo zbog toga je cjelokupno rješenje detaljno opisano počevši od infrastrukture sustava pa do testnog okruženja. Važno je napomenuti da je cijelo rješenje otvorenog koda, kako bi bilo pristupačnije za daljnja poboljšanja i unapređenja.

Literatura

[1] S. S. Islam, M. B. Mollah, M. I. Huq, and M. A. Ullah, "Cloud computing for future generation of computing technology," *2012 IEEE Int. Conf. Cyber Technol. Autom. Control. Intell. Syst.*, pp. 129–134, 2012.

[2] Y. Jadeja and K. Modi, "Cloud computing - Concepts, architecture and challenges," *2012 Int. Conf. Comput. Electron. Electr. Technol. ICCEET 2012*, pp. 877–880, 2012.

[3] C. G. Gruber, "CAPEX and OPEX in Aggregation and Core Networks," *2009 Conf. Opt. Fiber Commun. - incudes post deadline Pap.*, pp. 9–11, 2009.

[4] Y. Li, H. Wang, J. Dong, J. Li, and S. Cheng, "Operating cost reduction for distributed Internet data centers," *Proc. - 13th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2013*, pp. 589–596, 2013.

[5] M. Wiboonrat, "Life Cycle Cost Analysis of Data Center Project," *Int. Conf. Ecol. Veh. Renew. Energies*, vol. Ninth, 2014.

[6] J. Sampson and D. M. Tullsen, "Battery Provisioning and Associated Costs for Data Center Power Capping," UC San Diego, 2012.

[7] Q. L. O. L. H. Gdul *et al.*, "Total Cost of Ownership Model for Data Center Technology Evaluation," Sunnyvale, CA, 2017.

[8] W. Zhang, Y. Wen, S. Member, and L. L. Lai, "Electricity Cost Minimization for Interruptible Workload in Datacenter Servers," *IEEE Trans. Serv. Comput.*, vol. 1374, no. c, pp. 1–14, 2017.

[9] L. Ganesh, H. Weatherspoon, T. Marian, and K. Birman, "Integrated Approach To Data Center Power Management," *IEEE Trans. Comput.*, vol. 62, no. Dc, pp. 1–13, 2013.

[10] C. Ma *et al.*, "Virtual machine power metering and its applications," *2013 IEEE Glob. High Tech Congr. Electron. GHTCE 2013*, no. Llc, pp. 153–156, 2013.

[11] V. Kontorinis *et al.*, "Managing distributed UPS energy for effective power capping in data centers," *Proc. - Int. Symp. Comput. Archit.*, no. 39, pp. 488–499, 2012.

[12] L. Gu, D. Zeng, and S. Guo, "Type-aware Task Placement in Geo-distributed Data Centers with Low OPEX using Data Center Resizing," pp. 211–215, 2014.

[13] J. Yao, X. Liu, and C. Zhang, "Predictive electricity cost minimization through energy buffering in data centers," *IEEE Trans. Smart Grid*, vol. 5, no. 1, pp. 230–238, 2014.

[14] V. Soundararajan and B. Herndon, "Benchmarking a Virtualization Platform," *IEEE Int. Symp. Workload Charact.*, pp. 99–109, 2014.

[15] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "An approach to enable cloud service providers to arrange IaaS, PaaS, and SaaS using external virtualization infrastructures," *Proc. - 2011 IEEE World Congr. Serv. Serv. 2011*, pp. 607–611, 2011.

[16] G. Teodoro, T. Tavares, B. Coutinho, W. . J. Meira, and D. Guedes, "Load balancing on stateful clustered Web servers," *Proceedings. 15th Symp. Comput. Archit. High Perform. Comput.*, 2003.

[17] E. Yanmaz, O. K. Tonguz, and R. Rajkumar, "Is there an optimum dynamic load balancing scheme?," *GLOBECOM - IEEE Glob. Telecommun. Conf.*, vol. 1, pp. 598–602, 2005.

[18] D. Duplyakin, M. Haney, and H. Tufo, "Highly available cloud-based cluster management," *Proc. - 2015 IEEE/ACM 15th Int. Symp. Clust. Cloud, Grid Comput.*

CCGrid 2015, pp. 1201–1204, 2015.

[19] I. Haddad and G. Butler, "Experimental Studies of Scalability in Clustered Web Systems," *18th Int. Parallel Distrib. Process. Symp.*, vol. 0, no. C, 2004.

[20] X. Wang, "Load Balancing Control of Web-server Clusters : N -Tanks Model and a CTCT Method *," *Intell. Control Autom. 2008. WCICA 2008. 7th World Congr.*, no. 1, pp. 4069–4074, 2008.

[21] Yong Meng Teo and R. Ayani, "Comparison of Load Balancing Strategies on Cluster-based Web Servers," *Sage Journals - Simul.*, vol. 77, no. 5–6, pp. 185–195, 2001.

[22] D. Alagić and K. Arbanas, "Analysis and comparison of algorithms in advanced Web clusters solutions," in *Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Hrvatska*, 2016, pp. 208–213.

[23] D. Alagić and D. Maček, "Metamodeling as an Approach for Better Computer Resources Allocation in Web Clusters," in *Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Hrvatska*, 2016, pp. 214–219.

[24] P. Padala, K. G. Shin, X. Zhu, Z. Wang, S. Singhal, and K. Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments," in *Electrical Engineering and Computer Science - Computer Science and Engineering*, 2007, pp. 289–302.

[25] X. You, J. Wan, X. Xu, C. Jiang, W. Zhang, and J. Zhang, "ARAS-M: Automatic resource allocation strategy based on market mechanism in cloud computing," *J. Comput.*, vol. 6, no. 7, pp. 1287–1296, 2011.

[26] Z. Zhanjun, Y. Xueliang, and H. Chengde, "Key Issues of Resource Management in Distributed Multimedia Computer Systems," *Int. Conf. Commun. Technol. Proc.*, pp. 1628–1632, 2000.

[27] L. Aversa, "Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting," *IEEE*, pp. 24–29, 2000.

[28] V. Pham, E. Larsen, Ø. Kure, and P. E. Engelstad, "Gateway load balancing in future tactical networks," *Proc. - IEEE Mil. Commun. Conf. MILCOM*, pp. 1844–1850, 2010.

[29] M. J. Zaki, W. L. W. Li, and S. Parthasarathy, "Customized dynamic load balancing for a network of workstations," *Proc. 5th IEEE Int. Symp. High Perform. Distrib. Comput.*, pp. 282–291, 1996.

[30] S. Ristov, M. Gusev, K. Cvetkov, and G. Velkoski, "Implementation of a network based cloud load balancer," *Comput. Sci. Inf. Syst. (FedCSIS), 2014 Fed. Conf.*, vol. 2, pp. 775–780, 2014.

[31] B. F. Xu, F. Liu, H. Jin, and A. V Vasilakos, "Managing Performance Overhead of VirtualMachines in Cloud Computing: A Survey , State of the Art , and Future Directions," *Proc. IEEE*, vol. Vol. 102, no. No. 1, pp. 11–31, 2014.

[32] Z. Xiao, S. Member, W. Song, and Q. Chen, "Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. Volume: 24, no. Issue: 6, pp. 1–11, 2013.

[33] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," *IEEE*, vol. 5, pp. 119–128, 1960.

[34] F. Ma, F. Liu, and Z. Liu, "Distributed load balancing allocation of virtual machine in

cloud data center," *Softw. Eng. Serv. Sci. (ICSESS), 2012 IEEE 3rd Int. Conf.*, pp. 20–23, 2012.

[35] S. Marrone and R. Nardone, "Automatic resource allocation for high availability cloud services," *Procedia Comput. Sci.*, vol. 52, no. 1, pp. 980–987, 2015.

[36] Y. Ichikawa and N. Komoda, "Scenario-Based Task Executor for IT Resource Management," *2016 5th IIAI Int. Congr. Adv. Appl. Informatics*, pp. 888–893, 2016.

[37] A. P. Chester, M. Leeke, M. Al-Ghamdi, A. Jhumka, and S. A. Jarvis, "A Framework for Data Center Scale Dynamic Resource Allocation Algorithms," *2011 IEEE 11th Int. Conf. Comput. Inf. Technol.*, pp. 67–74, 2011.

[38] F. Tseng, M. Tsai, C. Tseng, and Y. Yang, "A Lightweight Auto - Scaling Mechanism for Fog Computing in Industrial Applications," *IEEE Trans. Ind. Informatics*, vol. 3203, no. c, 2018.

[39] A. Gandhi, M. O. R. Harchol-balter, and R. A. M. Raghunathan, "AutoScale : Dynamic , Robust Capacity Management for Multi-Tier Data Centers," vol. 30, no. 4, 2012.

[40] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-Scaling Web Applications in Clouds : A Taxonomy and Survey," *ACM Comput. Surv.*, vol. 51, no. 4, p. 33, 2018.

[41] M. C. Calzarossa, L. Massari, M. I. M. Tabash, and D. Tessera, "Cloud autoscaling for HTTP/2 workloads," *Int. Conf. Cloud Comput. Technol. Appl.*, no. 3, p. 6, 2017.

[42] A. Gandhi and S. Thota, "Autoscaling for Hadoop Clusters," *IEEE Int. Conf. Cloud Eng.*, pp. 109–118, 2016.

[43] A. Gandhi, P. Dube, A. Kochut, and L. Zhang, "Model-driven Autoscaling for Hadoop clusters," *IEEE 12th Int. Conf. Auton. Comput.*, pp. 155–156, 2015.

[44] Y. Hu, B. Deng, and F. Peng, "Autoscaling Prediction Models for Cloud Resource Provisioning," *IEEE Int. Conf. Comput. Commun.*, no. 2nd, pp. 1364–1369, 2016.

[45] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting," *IEEE Int. Conf. Cloud Comput.*, no. 4th, p. 8, 2011.

[46] M. N. A. H. Khan, Y. Liu, H. Alipour, and S. Singh, "Modeling the Autoscaling Operations in Cloud with Time Series Data," *IEEE Symp. Reliab. Distrib. Syst. Work.*, no. 34th, pp. 7–12, 2015.

[47] A. Jindal, V. Podolskiy, and M. Gerndt, "Multilayered Cloud Applications Autoscaling Performance Estimation," *IEEE Int. Symp. Cloud Serv. Comput.*, no. 7th, pp. 24–31, 2017.

[48] C. Self-aware, "Toward a Smarter Cloud: Self-Aware Autoscaling of Cloud Configurations and Resources," *IEEE Comput. Soc.*, no. September, pp. 93–96, 2015.

[49] E. Kern R., C. Hill, and N. (US), "Autonomic Scaling of Virtual Machines in a Cloud Computing Environment," US008572612B2, 2013.

[50] R. W. Schmidt and S. Rajagopal, "High Availability Virtual Machine Cluster," US008554981B2, 2013.

[51] F. Machida, "Provisioning Astandby Virtual Machine Based on The Predction of a Provisioning Request Being Generated," USOO8677353B2, 2014.

[52] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Futur. Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[53] X. Song, J. Shi, H. Chen, and B. Zang, "Schedule Processes, not VCPUs," *APSys'13*, pp. 1–7, 2013.

[54] H. Guan, R. Ma, and J. Li, "Workload-Aware Credit Scheduler for Improving Network I/O Performance in Virtualization Environment," *IEEE Trans. Cloud Comput.*, vol. 7161, no. c, pp. 1–1, 2014.

[55] OpenStack, "OpenStack Documentation," 2016. URL: http://docs.openstack.org. [05-06-2016].

[56] Eucalyptus, "Eucalyptus 4.4.4 Administration Guide," Ent. Services Development Corporation LP, California, United States, 2018.

[57] Eucalyptus, "Eucalyptus 4.3.1 User Guide," Hewlett Packard Enterprise Development LP, California, United States, 2017.

[58] VMware, *Performance Best Practices for VMware vSphere 6.0*, Revision: California, United States: VMware, 2015.

[59] VMware, "Understanding Memory Resource Management in VMware® ESXTM Server," *VMware, Inc.*, 2009. URL: http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/perf-vsphere-memory_management.pdf. [07-06-2016].

[60] J. Boche, "vSphere Memory Hot Add/CPU Hot Plug," *Boche*, 2009. URL: http://www.boche.net/blog/index.php/2009/05/10/vsphere-memory-hot-add-cpu-hot-plug/. [02-07-2016].

[61] S. Lowe, "vSphere 5.1: Hot add RAM and CPU," *VirtualizationAdmin.*, 2013. URL: http://www.virtualizationadmin.com/blogs/lowe/news/vsphere-51-hot-add-ram-and-cpu.html. [07-o6-2016].

[62] VMware, "vSphere 5 Documentation Center," *VMware, Inc.*, 2011. URL: https://pubs.vmware.com/vsphere-50/index.jsp#com.vmware.vsphere.vm_admin.doc_50/GUID-3CDA4DEF-3DE0-4A64-89C7-F31BB77222CB.html. [10-06-2016].

[63] R. Ganguly, "Onboarding Microsoft Hyper-V resources," *BMC Software*, 2017. URL: https://docs.bmc.com/docs/cloudlifecyclemanagement/46/onboarding-microsoft-hyper-v-resources-669201921.html. [28-10-2018].

[64] S. Halbe, "Hyper-V," *BMC Software, Inc.*, 2015. URL: https://docs.bmc.com/docs/display/public/btco103/Hyper-V. [08-08-2016].

[65] EVault, *Hyper-V Agent 7.4 User Guide*, 4.7. Wilmington, New Castle: EVault Inc., 2014.

[66] L. Davies, Kathy Poggemeyer and H. Justin, "Manage Hyper-V on Windows Server," *Microsoft*, 2018. URL: https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/manage/manage-hyper-v-on-windows-server. [27-10-2018].

[67] Microsoft, "Hyper-V Dynamic Memory Overview," *Microsoft*, 2014. URL: https://technet.microsoft.com/en-us/library/hh831766. [02-09-2016].

[68] Amazon Wev Services, "Amazon Elastic MapReduce Developer Guide," California, 2016.

[69] Amazon Wev Services, "Amazon EMR Developer Guide," *Amazon Wev Services*, 2017. URL: http://docs.aws.amazon.com/emr/latest/DeveloperGuide/emr-dg.pdf. [10-09-2016].

[70] A. S. Rodge, C. Pramanik, J. Bose, and S. K. Soni, “Multicast Routing with Load Balancing Using Amazon Web Service Multicast Routing with Load Balancing Using Amazon Web Service,” no. JANUARY 2014, 2015.

[71] I. Bermudez, S. Traverso, M. Mellia, and M. Munafo, “Exploring the cloud from passive measurements: The Amazon AWS case,” *Proc. - IEEE INFOCOM*, pp. 230–234, 2013.

[72] I. Bermudez, S. Traverso, M. Munafo, and M. Mellia, “A distributed architecture for the monitoring of clouds and CDNs: Applications to Amazon AWS,” *IEEE Trans. Netw. Serv. Manag.*, vol. 11, no. 4, pp. 516–529, 2014.

[73] Mitchell Anicas, “How To Resize Your Droplets on DigitalOcean,” *DigitalOcean*. URL: https://www.digitalocean.com/community/tutorials/how-to-resize-your-droplets-on-digitalocean. [07-11-2016].

[74] L. M. Qaisi and I. Aljarah, “A Twitter Sentiment Analysis for Cloud Providers: A Case Study of Azure vs. A WS,” pp. 1–6, 2016.

[75] G. Carutasu, M. A. Botezatu, C. Botezatu, and M. Pirnau, “Cloud computing and windows azure,” *2016 8th Int. Conf. Electron. Comput. Artif. Intell.*, pp. 1–6, 2016.

[76] J. Jann, L. M. Browning, and R. S. Burugula, “Dynamic reconfiguration : Basic building blocks for autonomic computing on IBM pSeries servers,” *IBM Syst. J.*, vol. 42, no. 1, pp. 29–37, 2003.

[77] M. Cordero, H. Lin, V. Thatikonda, and R. Xavier, *IBM PowerVM Virtualization Introduction and Configuration*, Sixth Edit. New York, U.S.: IBM Redbooks, 2013.

[78] T. C. Group, “Clustering: A basic 101 tutorial,” *Ibm*, no. April, p. 19, 2002.

[79] S. G. Bueno *et al.*, *IBM PowerVM Virtualization Managing and Monitoring*, Fifth Edit. New York, U.S.: IBM Redbooks, 2013.

[80] L. Cherkasova, D. Gupta, and A. Vahdat, “Comparison of the three CPU schedulers in Xen,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, 2007.

[81] A. Grattafiori, *Understanding and Hardening Linux Containers*, Version 1. San Francisco: NCC Group, 2016.

[82] Docker Inc., “Building a Continuous Integration Pipeline with Docker,” *Docker Inc.*, 2015. URL: https://www.docker.com/sites/default/files/UseCase/RA_CI with Docker_08.25.2015.pdf. [21-11-2016].

[83] U. Ismail and B. Sheikh, *Continuous Integration and Deployment with Docker and Rancher*, no. January. Rancher Labs, 2016.

[84] R. Hill, L. Hirsch, P. Lake, and S. Moshiri, “Containers & Docker: Emerging Roles & Future of Cloud Technology,” pp. 65–89, 2013.

[85] B. Ivan, “Samooblikujuća arhitektura sustava zasnovanih na uslugama,” Doktorska disertacija, Fakultet elektrotehnike i računarstva, Sveučilišta u Zagrebu, 2008.

[86] S. R. Jiri Herrmann, Dayle Parker, *Red Hat Enterprise Linux 7 Virtualization Tuning and Optimization Guide*. North Carolina: Red Hat, Inc., 2015.

[87] L. Novich, S. Radvan, D. Parker, and T. Richardson, *Red Hat Enterprise Linux 7 Virtualization Deployment and Administration Guide*. North Carolina: Red Hat, Inc., 2015.

[88] VMware, “vSphere Resource Management,” *EN-000793-00*, 2012. URL: https://pubs.vmware.com/vsphere-51/topic/com.vmware.ICbase/PDF/vsphere-esxi-

vcenter-server-51-resource-management-guide.pdf. [02-10-2016].

[89] M. Kircher and P. Jain, *Pattern-Oriented Software Architecture, Patterns for Resource Management*, Volume 3., vol. 3. Wiley, 2004.

[90] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–78, 2007.

[91] D. Spengler, "Understanding and using htop to monitor system resources," 2012. URL: http://www.deonsworld.co.za/2012/12/20/understanding-and-using-htop-monitor-system-resources/. [02-10-2017].

[92] J. Šoltýs, "Linux Kernel 2 . 6 Documentation," Comenius University Faculty of Mathematics, Physics and Informatics. Bratislava, 2006.

[93] P. Long, "VMware vSphere Hot Add and Hot Plug," *PeteNetLive*, 2013. URL: http://www.petenetlive.com/KB/Article/0000527. [02-07-2017].

[94] D. R. J.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, *Performance Testing Guidance for Web Applications*, 1 edition. Microsoft Press, 2007.

[95] W. Tarreau, "HAProxy - Configuration Manual," *version 1.9*, 2018. URL: http://cbonte.github.io/haproxy-dconv/. [28-10-2018].

[96] D. Alagić and I. Magdalenić, "Model for Automated and Improved Utilization of Existing Computer Resources on an Example of Web Servers," *J. Comput. Sci. - Sci. Publ.*, vol. 14, no. 2, pp. 286–303, 2018.

Popis priloga

Prilog 1. Izvorni kôd Host agenta

```
1. #!/bin/bash
2. # VM Manager config
3. HOSTNAME=$(hostname)
```

```
4.  HOME_PATH="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)/"
5.  LOG_NAME='vmmanager'
6.  LOG_NAME_RESOURCE_USAGE='resource_usage'
7.  LOCK_FILE="${HOME_PATH}/lock.file"
8.  SKIP_COUNT_FILE="${HOME_PATH}/skip.count"
9.  SKIP_COUNT_LIMIT=3
10. VM_MANAGER_CONFIG_FILE=${HOME_PATH}/vm_config.csv
11. MAIL_SUBJECT="VM manager (${HOSTNAME})"
12. MAIL_TO="dialagic@gmail.com" # For more recepients use space
13. VM_INFO_PORT='9299'
14. VM_LA_LOGIC='LA'        # LA, INCREMENT
15. VM_LA_MODIFICATOR='1.10'
16. VM_LA_ROUNDING='UP'     # HALF_UP, DOWN, UP
17. VM_LA_TRESHOLD_UP='90'
18. VM_LA_TRESHOLD_DOWN='80'
19. VM_RAM_TRESHOLD_UP='70'
20. VM_RAM_TRESHOLD_DOWN='30'
21. VM_RAM_MIN_FREE=204800  # in kB ()
22. VM_TOTAL_CPU_LIMIT=16
23. VM_TOTAL_RAM_LIMIT=26
24. VM_TOTAL_RAM_KB_LIMIT=$(echo "${VM_TOTAL_RAM_LIMIT}*1024*1024" | bc)
25. KVM_RAM_OFFSET=.411
26.
27. # VM config defaults
28. CPU_MIN_DEF=1
29. CPU_MAX_DEF=8
30. MEM_MIN_DEF=1
31. MEM_MAX_DEF=8
32. CPU_BALANCE_LOGIC_DEF='2:1;8:2;16:4'
33. MEM_BALANCE_LOGIC_DEF='2:1;8:2;16:4'
34.
35. # Global variables (DO NOT EDIT!!!)
36. VM_NAME="
37. CPU_MIN="
38. CPU_MAX="
39. CPU_CURR="
40. MEM_MIN="
41. MEM_MIN_KB="
42. MEM_MAX="
43. MEM_MAX_KB="
44. MEM_CURR_KB="
45. CPU_BALANCE_LOGIC="
46. MEM_BALANCE_LOGIC="
47. VM_TOTAL_CPU=0
48. VM_TOTAL_RAM_KB=0
49. VM_TOTAL_RAM="
50. VM_LA="
51. VM_RAM="
52. VM_RAM_FREE_KB="
53. VM_RAM_TOTAL="
54. VM_RAM_USED="
55. VM_CPU_NEW="
56. VM_RAM_NEW="
57.
58. function log {
59.
60.     log_file="${HOME_PATH}/logs/${LOG_NAME}-`date +%Y.%m.%d`.log"
61.     now=$(date "+%Y-%m-%d %H:%M:%S:%N" | cut -c1-23)
62.     log_data="$1"
63.     # log
64.     echo -e "${now}   ${log_data}" >> ${log_file}
65. }
66.
```

```
67. function log_resource_usage {
68.     log_file="${HOME_PATH}/logs/${LOG_NAME_RESOURCE_USAGE}-`date
    +%Y.%m.%d`.log"
69.     now=$(date "+%Y-%m-%d %H:%M:%S" | cut -c1-19)
70.     # log
71.     echo -e
    "${now}\t${VM_NAME}\t${VM_LA}\t${VM_CPU_NEW}\t${VM_RAM_USED}\t${VM_RAM_N
    EW}" >> ${log_file}
72. }
73.
74. function send_mail {
75.     body=$1
76.     echo -e "${body}" | mail -s "${MAIL_SUBJECT}" "${MAIL_TO}"
77. }
78.
79. function vm_resource_setup_total() {
80.     log "Get total CPU and RAM assigned to VMs..."
81.     while IFS=',' read VM_NAME cpu_min_conf cpu_max_conf mem_min_conf mem_max_conf
    cpu_balance_logic_conf mem_balance_logic_conf
82.     do
83.         if [ "${VM_NAME}" != "vm" ]
84.         then
85.             VM_TOTAL_CPU=$(echo "${VM_TOTAL_CPU}+$(vm_config_read 'cpu')" | bc)
86.             VM_TOTAL_RAM_KB=$(echo "${VM_TOTAL_RAM_KB}+$(vm_config_read
    'ram')" | bc)
87.         fi
88.     done < ${VM_MANAGER_CONFIG_FILE}
89.     VM_TOTAL_RAM=$(echo "scale=2; ${VM_TOTAL_RAM_KB}/1024/1024" | bc)
90.     log "Total CPU assigned: ${VM_TOTAL_CPU}"
91.     log "Total RAM assigned: ${VM_TOTAL_RAM_KB}kB (${VM_TOTAL_RAM}GB)"
92. }
93.
94. function vm_check() {
95.     curl -s -m 2 http://${VM_NAME} > /dev/null
96.     response_code=$?
97.     if [ "${response_code}" == '0' ]
98.     then
99.         log "Apache is listening. VM ${VM_NAME} OK!"
100.    else
101.        log "Apache is not listening, check ${VM_NAME}!!!"
102.    fi
103. }
104.
105. function vm_config_set() {
106.     vm_set_new_cpu=$1
107.     vm_set_new_ram=$2
108.     if [ "${vm_set_new_cpu}" != '0' ]
109.     then
110.         log "Setting VM ${VM_NAME} CPU to ${vm_set_new_cpu} cores"
111.         virsh setvcpus ${VM_NAME} --count ${vm_set_new_cpu}
112.         VM_TOTAL_CPU=$(echo "${VM_TOTAL_CPU}-${CPU_CURR}+${vm_set_new_cpu}"
    | bc)
113.     fi
114.     if [ "${vm_set_new_ram}" != '0' ]
115.     then
116.         log "Setting VM ${VM_NAME} RAM to ${vm_set_new_ram}kb"
117.         virsh setmem ${VM_NAME} ${vm_set_new_ram}
118.         VM_TOTAL_RAM_KB=$(echo "${VM_TOTAL_RAM_KB}-
    ${MEM_CURR_KB}+${vm_set_new_ram}" | bc)
119.         VM_TOTAL_RAM=$(echo "scale=2; ${VM_TOTAL_RAM_KB}/1024/1024" | bc)
120.     fi
121. }
122.
```

```
123. function vm_config_read() {
124.     vm_property=$1 # cpu ram
125.     case "${vm_property}" in
126.         'cpu')
127.             virsh dominfo ${VM_NAME} | awk -F' ' '/CPU\(s\)/ {print $2}'
128.         ;;
129.         'ram')
130.             virsh dominfo ${VM_NAME} | awk -F' ' '/Used memory:/ {print $3}'
131.         ;;
132.     esac
133. }
134.
135. function vm_get_new_value() {
136.     vm_property=$1 # ram cpu
137.     vm_change=$2 # more less
138.     increment_apply=1
139.     if [ "${vm_change}" == 'more' ] ; then operand='+'; else operand='-'; fi
140.     if [ "${vm_property}" == 'ram' ]
141.     then
142.         log "Calculating new RAM size..."
143.         #vm_current_ram_kb=$(vm_config_read 'ram')
144.         vm_current_ram_gb=$(echo "scale=2; ${MEM_CURR_KB}/1024/1024" | bc)
145.         for config in $(echo ${MEM_BALANCE_LOGIC} | sed 's|;| |g')
146.         do
147.             limit=$(echo ${config} | awk -F':' '{print $1}')
148.             increment=$(echo ${config} | awk -F':' '{print $2}')
149.             if (( $(echo "${vm_current_ram_gb} >= ${limit}" | bc -l) ))
150.             then
151.                 increment_apply=${increment}
152.             else
153.                 break
154.             fi
155.         done
156.         ram_new_value=$(echo "${MEM_CURR_KB}${operand}${increment_apply}*1024*1024"
    | bc)
157.         if (( $(echo "${ram_new_value} < ${MEM_MIN_KB}" | bc -l) ))
158.         then
159.             ram_new_value=${MEM_MIN_KB}
160.         else
161.             if (( $(echo "${ram_new_value} > ${MEM_MAX_KB}" | bc -l) ))
162.             then
163.                 ram_new_value=${MEM_MAX_KB}
164.             fi
165.         fi
166.         log "New RAM size: ${ram_new_value}"
167.         echo ${ram_new_value}
168.     else
169.         log "Calculating new CPU count (Logic: ${VM_LA_LOGIC})..."
170.         if [ ${VM_LA_LOGIC} == 'INCREMENT' ]
171.         then
172.             for config in $(echo ${CPU_BALANCE_LOGIC} | sed 's|;| |g')
173.             do
174.                 limit=$(echo ${config} | awk -F':' '{print $1}')
175.                 increment=$(echo ${config} | awk -F':' '{print $2}')
176.                 if (( $(echo "${CPU_CURR} >= ${limit}" | bc -l)  ))
177.                 then
178.                     increment_apply=${increment}
179.                 else
180.                     break
181.                 fi
182.             done
183.             cpu_new_value=$(echo "${CPU_CURR}${operand}${increment_apply}" | bc)
184.         else
```

```
185.                    case "${VM_LA_ROUNDING}" in
186.                        "HALF_UP")
187.                            cpu_new_value=$(printf '%.0f\n'  $(echo
    "${VM_LA}*${VM_LA_MODIFICATOR}" | bc))
188.                        ;;
189.                        "DOWN")
190.                            cpu_new_value=$(echo $(echo "${VM_LA}*${VM_LA_MODIFICATOR}"
    | bc) | awk -F'.' '{print $1}')
191.                            if [ "${cpu_new_value}" == '' ] ; then cpu_new_value='1'; fi
192.                        ;;
193.                        "UP")
194.                            tmp=$(echo $(echo "${VM_LA}*${VM_LA_MODIFICATOR}" | bc) | awk -
    F'.' '{print $1}')
195.                            if [ "${tmp}" == '' ] ; then tmp='0'; fi
196.                            cpu_new_value=$(echo "${tmp}+1" | bc)
197.                        ;;
198.                    esac
199.                fi
200.                if (( $(echo "${cpu_new_value} < ${CPU_MIN}" | bc -l) ))
201.                then
202.                    cpu_new_value=${CPU_MIN}
203.                else
204.                    if (( $(echo "${cpu_new_value} > ${CPU_MAX}" | bc -l) ))
205.                    then
206.                        cpu_new_value=${CPU_MAX}
207.                    fi
208.                fi
209.                log "New CPU count: ${cpu_new_value}"
210.                echo ${cpu_new_value}
211.            fi
212.}
213.
214. function vm_update() {
215.     change_cpu=''
216.     change_ram=''
217.     vm_cpu_new=0
218.     vm_ram_new=0
219.     local OPTIND
220.     while getopts ":c:r:" opt; do
221.         case "${opt}" in
222.             c)
223.                 change_cpu="${OPTARG}"
224.                 ;;
225.             r)
226.                 change_ram="${OPTARG}"
227.                 ;;
228.         esac
229.     done
230.     if [ ! -z ${change_cpu} ]
231.     then
232.         vm_cpu_new=$(vm_get_new_value 'cpu' "${change_cpu}")
233.         VM_CPU_NEW=${vm_cpu_new}
234.     fi
235.     if [ ! -z ${change_ram} ]
236.     then
237.         vm_ram_new=$(vm_get_new_value 'ram' "${change_ram}")
238.         VM_RAM_NEW=$(echo "scale=3; ${vm_ram_new}/1024/1024" | bc)
239.     fi
240.     log "Checking host limits..."
241.     vm_total_cpu_new=$(echo "${VM_TOTAL_CPU}-${CPU_CURR}+${vm_cpu_new}" | bc)
242.     vm_total_ram_new=$(echo "${VM_TOTAL_RAM_KB}-
    ${MEM_CURR_KB}+${vm_ram_new}" | bc)
243.     if (( $(echo "${vm_total_cpu_new}>${VM_TOTAL_CPU_LIMIT}" | bc -l) ))
```

```
244.      then
245.          log "Total VMs CPU count with new ${VM_NAME} configuration exceedes host limit
      (${vm_total_cpu_new}>${VM_TOTAL_CPU_LIMIT}), skipping CPU update"
246.          vm_cpu_new=0
247.          VM_CPU_NEW=${CPU_CURR}
248.      else
249.          log "Total VMs CPU count with new ${VM_NAME} configuration under host limit"
250.      fi
251.      if (( $(echo "${vm_total_ram_new}>${VM_TOTAL_RAM_KB_LIMIT}" | bc -l) ))
252.      then
253.          log "Total VMs RAM with new ${VM_NAME} configuration exceedes host limit
      (${vm_total_ram_new}>${VM_TOTAL_RAM_KB_LIMIT}), skipping RAM update"
254.          vm_ram_new=0
255.          VM_RAM_NEW=$(echo "scale=3; ${MEM_CURR_KB}/1024/1024" | bc)
256.      else
257.          log "Total VMs RAM with new ${VM_NAME} configuration under host limit"
258.      fi
259.      if [[ "${vm_cpu_new}" == "0" || "${vm_cpu_new}" == "${CPU_CURR}" ]] && [[
      "${vm_ram_new}" == "0" || "${vm_ram_new}" == "${MEM_CURR_KB}" ]]
260.      then
261.          log "New values same as current, skipping VM ${VM_NAME} update."
262.      else
263.          vm_config_set ${vm_cpu_new} ${vm_ram_new}
264.          sleep 1
265.
266.          log "Checking ${VM_NAME} Apache..."
267.          vm_check
268.
269.          log "Total CPU and RAM assigned to VMs after update..."
270.          log "Total CPU assigned: ${VM_TOTAL_CPU}"
271.          log "Total RAM assigned: ${VM_TOTAL_RAM_KB}kB (${VM_TOTAL_RAM}GB)"
272.      fi
273.  }
274.
275.  function vm_info_get () {
276.      log "Getting VM ${VM_NAME} current RAM and CPU info..."
277.      # curl --max-time 10 --retry 3 --retry-delay 5 --retry-max-time 32
278.      response=$(curl --max-time 10 --retry 3 --retry-delay 2 --retry-max-time 45
      http://${VM_NAME}:${VM_INFO_PORT})
279.      exit_status=$?
280.      retry_count=1
281.      while [ "${exit_status}" != "0" ] && (( $(echo "${retry_count}<=3" | bc -l) ))
282.      do
283.          log "Getting info failed (exit code: ${exit_status}), sleep for 2 seconds before retry
      (${retry_count})"
284.          sleep 2
285.          response=$(curl --max-time 10 --retry 3 --retry-delay 2 --retry-max-time 45
      http://${VM_NAME}:${VM_INFO_PORT})
286.          exit_status=$?
287.          retry_count=$(echo "${retry_count}+1" | bc)
288.      done
289.      if [ "${exit_status}" == "0" ]
290.      then
291.          for info in $response
292.          do
293.              param=$(echo ${info} | awk -F'=' '{print $1}')
294.              value=$(echo ${info} | awk -F'=' '{print $2}')
295.
296.              case ${param} in
297.                  load_1)
298.                      VM_LA=${value}
299.                      log "VM ${VM_NAME} LA(5): ${VM_LA}"
300.                  ;;
```

```
301.                        available_percentage)
302.                            VM_RAM=${value}
303.                            log "VM ${VM_NAME} available RAM: ${VM_RAM}%"
304.                        ;;
305.                        available)
306.                            VM_RAM_FREE_KB=${value}
307.                            log "VM ${VM_NAME} available RAM: ${VM_RAM_FREE_KB}kB"
308.                        ;;
309.                        total)
310.                            VM_RAM_TOTAL=$(echo "scale=3; ${value}/1024/1024" | bc)
311.                        ;;
312.                    esac
313.                done
314.                VM_RAM_USED=$(echo "scale=3; ${VM_RAM_TOTAL}-
    (${VM_RAM_FREE_KB}/1024/1024)+${KVM_RAM_OFFSET}" | bc)
315.                return 0
316.            else
317.                log "Getting info failed after 3 retries (last exit code: ${exit_status})!!! Check for issues!"
318.                send_mail "Getting info for VM ${VM_NAME} failed!!! Check for issues!"
319.                return "${status}"
320.            fi
321.        }
322.
323.        function vm_monitor() {
324.            local vm_cpu_change
325.            local vm_ram_change
326.            log "VM check STARTED!"
327.            vm_resource_setup_total
328.            log "Start iterating through nodes..."
329.            while IFS=',' read VM_NAME cpu_min_conf cpu_max_conf mem_min_conf mem_max_conf
    cpu_balance_logic_conf mem_balance_logic_conf
330.            do
331.                if [ "${VM_NAME}" != "vm" ]
332.                then
333.                    log "START checking VM ${VM_NAME}!"
334.                    log "VM manager setup for VM ${VM_NAME} => CPU min: ${cpu_min_conf}; CPU
    max: ${cpu_max_conf}; RAM min: ${mem_min_conf}; RAM max: ${mem_max_conf}; CPU balance
    logic: ${cpu_balance_logic_conf}; RAM balance logic: ${mem_balance_logic_conf}"
335.                    if [ ! -z "${cpu_min_conf}" ] ; then CPU_MIN="${cpu_min_conf}"; else
    CPU_MIN="${CPU_MIN_DEF}"; fi
336.                    if [ ! -z "${cpu_max_conf}" ] ; then CPU_MAX="${cpu_max_conf}"; else
    CPU_MAX="${CPU_MAX_DEF}"; fi
337.                    if [ ! -z "${mem_min_conf}" ] ; then MEM_MIN="${mem_min_conf}"; else
    MEM_MIN="${MEM_MIN_DEF}"; fi
338.                    MEM_MIN_KB=$(echo "${MEM_MIN}*1024*1024" | bc)
339.                    if [ ! -z "${mem_max_conf}" ] ; then MEM_MAX="${mem_max_conf}"; else
    MEM_MAX="${MEM_MAX_DEF}"; fi
340.                    MEM_MAX_KB=$(echo "${MEM_MAX}*1024*1024" | bc)
341.                    if [ ! -z "${cpu_balance_logic_conf}" ] ; then
    CPU_BALANCE_LOGIC="${cpu_balance_logic_conf}"; else
    CPU_BALANCE_LOGIC="${CPU_BALANCE_LOGIC_DEF}"; fi
342.                    if [ ! -z "${mem_balance_logic_conf}" ] ; then
    MEM_BALANCE_LOGIC="${mem_balance_logic_conf}"; else
    MEM_BALANCE_LOGIC="${MEM_BALANCE_LOGIC_DEF}"; fi
343.                    log "VM manager setup after loading defaults for VM ${VM_NAME} => CPU min:
    ${CPU_MIN}; CPU max: ${CPU_MAX}; RAM min: ${MEM_MIN}; RAM max: ${MEM_MAX};
    CPU balance logic: ${CPU_BALANCE_LOGIC}; RAM balance logic:
    ${MEM_BALANCE_LOGIC}"
344.                    log "Getting current VM setup..."
345.                    CPU_CURR=$(vm_config_read 'cpu')
346.                    MEM_CURR_KB=$(vm_config_read 'ram')
347.                    log "Current VM setup => CPU: ${CPU_CURR}; RAM: ${MEM_CURR_KB}kB"
348.                    vm_info_get
```

```
349.                exit_status=$?
350.                if [ "${exit_status}" == "0" ]
351.                then
352.                    log "Calculating resource usage..."
353.                    vm_cpu_usage=$(echo "scale=2; ${VM_LA}/${CPU_CURR}*100" | bc)
354.                    vm_ram_usage=$(echo "scale=2; 100-${VM_RAM}" | bc)
355.                    log "CPU usage: ${vm_cpu_usage}%"
356.                    log "RAM usage: ${vm_ram_usage}%"
357.                    log "Checking tresholds..."
358.                    if (( $(echo "${vm_cpu_usage} > ${VM_LA_TRESHOLD_UP}" | bc -l) ))
359.                    then
360.                        log "CPU usage higher than UP treshold: ${vm_cpu_usage} >
    ${VM_LA_TRESHOLD_UP}"
361.                        vm_cpu_change='-c more'
362.                    else
363.                        if (( $(echo "${vm_cpu_usage} < ${VM_LA_TRESHOLD_DOWN}" | bc -l)
    ))
364.                        then
365.                            log "CPU usage lower than DOWN treshold: ${vm_cpu_usage} <
    ${VM_LA_TRESHOLD_DOWN}"
366.                            vm_cpu_change='-c less'
367.                        else
368.                            log "CPU usage within optimum range:
    ${VM_LA_TRESHOLD_DOWN} - ${VM_LA_TRESHOLD_UP}"
369.                            VM_CPU_NEW=${CPU_CURR}
370.                        fi
371.                    fi
372.                    if (( $(echo "${vm_ram_usage} > ${VM_RAM_TRESHOLD_UP}" | bc -l) ))
373.                    then
374.                        log "RAM usage higher than UP treshold: ${vm_ram_usage} >
    ${VM_RAM_TRESHOLD_UP}"
375.                        vm_ram_change='-r more'
376.                    else
377.                        if (( $(echo "${vm_ram_usage} < ${VM_RAM_TRESHOLD_DOWN}" | bc -
    l) ))
378.                        then
379.                            log "RAM usage lower than DOWN treshold: ${vm_ram_usage} <
    ${VM_RAM_TRESHOLD_DOWN}"
380.                            vm_ram_change='-r less'
381.                        else
382.                            if (( $(echo "${VM_RAM_FREE_KB} < ${VM_RAM_MIN_FREE}" |
    bc -l) ))
383.                            then
384.                                log "Free RAM lower than limit: ${VM_RAM_FREE_KB}kB <
    ${VM_RAM_MIN_FREE}kB"
385.                                vm_ram_change='-r more'
386.                            else
387.                                log "RAM usage within optimum range:
    ${VM_RAM_TRESHOLD_DOWN} - ${VM_RAM_TRESHOLD_UP}; Free (kb):
    ${VM_RAM_FREE_KB} (limit: ${VM_RAM_MIN_FREE})"
388.                                VM_RAM_NEW=$(echo "scale=3;
    ${MEM_CURR_KB}/1024/1024" | bc)
389.                            fi
390.                        fi
391.                    fi
392.                    if [ ! -z "${vm_cpu_change}" ] || [ ! -z "${vm_ram_change}" ]
393.                    then
394.                        vm_update ${vm_cpu_change} ${vm_ram_change}
395.                    fi
396.                    log_resource_usage
397.                    log "Checking VM ${VM_NAME} FINISHED!"
398.                else
399.                    log "Skipping VM ${VM_NAME}!"
```

```
400.                    fi
401.                fi
402.        done < ${VM_MANAGER_CONFIG_FILE}
403.        log "Iterating through nodes finnished."
404.        log "VM check FINNISHED!"
405.        log "********************"
406.    }
407.
408.    function lock_exec {
409.        case $1 in
410.            "lock")
411.                if test -f ${LOCK_FILE}
412.                then
413.                    count_old=$(cat ${SKIP_COUNT_FILE} 2>/dev/null || echo '0')
414.                    log "VM manager script already running, skipping (previous skipps:
    ${count_old})..."
415.                    count_new=$(echo "${count_old}+1" | bc)
416.                    if (( $(echo "${count_new} >= ${SKIP_COUNT_LIMIT}" | bc -l) ))
417.                    then
418.                        log "Skipping for ${count_new} times in row (Treshold:
    ${SKIP_COUNT_LIMIT})!!!"
419.                        send_mail "Script run skiped for ${count_new} times in row (Treshold:
    ${SKIP_COUNT_LIMIT})!!!\nCheck for problems on ${HOSTNAME}"
420.                    fi
421.                    echo "${count_new}" > ${SKIP_COUNT_FILE}
422.                    exit 0
423.                else
424.                    echo '0' > ${SKIP_COUNT_FILE}
425.                    touch ${LOCK_FILE}
426.                fi
427.            ;;
428.            "unlock")
429.                rm -f ${LOCK_FILE}
430.            ;;
431.        esac
432.    }
433.
434.    lock_exec 'lock'
435.    vm_monitor
436.    lock_exec 'unlock'
```

Prilog 2. Izvorni kôd VM agenta

```
1.  #!/bin/bash
2.
3.  LOAD=''
4.  RAM=''
5.
6.  function response() {
7.      get_load
8.      get_ram_usage
9.
10.     load_raw=$(echo -e ${LOAD})
11.     ram_raw=$(echo -e ${RAM})
12.     content_lenght="Content-Length: $((${#load_raw} + ${#ram_raw}))\r\n"
13.
14.     echo -en "HTTP/1.1 200 OK\r\n"
15.     echo -en "Content-Type: text/plain\r\n"
```

```
    echo -en "Connection: close\r\n"
    echo -en ${content_lenght}
    echo -en "\r\n"
    echo -en ${LOAD}
    echo -en ${RAM}
}

function get_load() {
    load_curr_1=$(cat /proc/loadavg | awk -F" " "{print \$1}")
    load_curr_5=$(cat /proc/loadavg | awk -F" " "{print \$2}")
    load_curr_15=$(cat /proc/loadavg | awk -F" " "{print \$3}")

   LOAD=$(echo "load_1=${load_curr_1}\nload_5=${load_curr_5}\nload_15=${load_curr_15}")
}

function get_ram_usage() {
    total=$(free | awk '/Mem/ {print $2}')
    free=$(free | awk '/Mem/ {print $4}')
    available=$(free | awk '/buffers\/cache/ {print $4}')
    free_percentage=$(echo "scale=2; ${free}*100/${total}" | bc)
    available_percentage=$(echo "scale=2; ${available}*100/${total}" | bc)

RAM="\ntotal=${total}\nfree=${free}\navailable=${available}\nfree_percentage=${free_percentage}\
navailable_percentage=${available_percentage}"
}

#get_load
#get_ram_usage
response
```

Prilog 3. Konfiguracija početnih parametra (vm_config.csv)

```
vm,cpu_min,cpu_max,mem_min,mem_max,cpu_balance_logic,mem_balance_logic
dt-web1.int.ch,1,6,1,12,2:1;8:2,2:1;8:2;16:4
dt-web2.int.ch,,2,,4,,2:1;8:2
```

Životopis

Dino Alagić rođen je 1989. godine u Bihaću (Bosna i Hercegovina), gdje je završio osnovnu školu i srednju školu (Opća Gimnazija "Bihać"). Obrazovanje nastavlja u Varaždinu na Fakultetu organizacije i informatike, gdje upisuje preddiplomski studij koji završava 2010. godine. Tema završnog rada je bila “Programiranje u multimedijskim jezicima i alatima” (mentor, prof. dr. sc. Danijel Radošević). Nakon toga na istoimenom fakultetu upisuje diplomski studij, smjer Informacijsko i programsko inženjerstvo, koji završava 2012. godine kao jedan od najboljih studenata svoje generacije, zbog čega dobiva i pohvalu „Cum laude“.

Diplomski studij završava obranom rada pod temom "Procjena rizika metodom Octave Allegro" (mentor, prof. dr. sc. Željko Hutinski). Tijekom diplomskog studija dobio je Rektorovu nagradu Sveučilišta u Zagrebu za rad pod naslovom "Pojednostavljenje primjene metode procjene rizika uz regionalizaciju prijetnji informacijskom sustavu". Isti rad predstavlja na Dvadeset i drugoj međunarodnoj konferenciji u Varaždinu (22th Central European Conference on Information and Intelligent Systems, CECiiS 2011.). Za vrijeme diplomskog studija u akademskoj godini 2010/2011. dobio je Dekanovu nagradu za trud i izvrsnost u radu. Tijekom diplomskog studija učestvovao je u Erasmus programu (Intensive Programme E-Discovery), koji je 2012. godine održan u Manchesteru (Engleska). Svoje obrazovanje nastavlja 2012. godine upisom na poslijediplomski doktorski studij na Fakultetu organizacije i informatike u Varaždinu.

Nakon uspješno odrađene studentske prakse u tvrtki INFIGO IS d.o.o. u Zagrebu, prvo zaposlenje ostvaruje 2012. godine u tvrtki Samurai Digital d.o.o. (NTH grupa) u Varaždinu, gdje je radio kao tehničar na održavanju IT sustava. Nakon nekoliko mjeseci unutar grupe prelazi u drugu tvrtku pod nazivom NTH ICT d.d. gdje je radio na poziciji voditelja IT odjela. Godine 2013. pohađa tečaj pod nazivom Mikrotik Certified Network Associate - MTCNA, a godine 2014. Cisco CCNA Akademiju. Iste godine je bio na dva RIPE NCC treninga (LIR i Routing Security). Godine 2014. prelazi u NTH Mobile d.o.o. gdje radi kao voditelj ICT odjela.

Popis radova

- Alagić D., Magdalenić I., Model for Automated and Improved Utilization of Existing Computer Resources on an Example of Web Servers, Journal of Computer Science - Science Publications (1549-3636), Vol 14, Issue 2, pp. 286-303, 2018.
- Alagić D., Maček D., Metamodeling as an Approach for Better Computer Resources Allocation in Web Clusters, Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Hrvatska, pp. 214–219, 2016.
- Alagić D., Arbanas K., Analysis and comparison of algorithms in advanced Web clusters solutions, Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Hrvatska, pp. 208–213, 2016.

- Maček D., Alagić D., Comparisons of Bitcoin Cryptosystem with Other Common Internet Transaction Systems by AHP Technique," Journal of Information and Organizational Sciences of the Faculty of Organization and Informatics in Varazdin, Vol 41, No 1., pp. 69–87, 2017.
- Arbanas K., Alagić D., Requirements of practice in relation to the existing information technology and security management competencies, Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Hrvatska, pp. 1411-1416, 2014.