UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Janez Perme

Simulacija izvajanja poslov na upravljanem porazdeljenem sistemu

MAGISTRSKA NALOGA

prof. dr. Nikolaj Zimic
MENTOR

dr. Andrej Brodnik
SOMENTOR

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 138-MAG-RI/2011
Datum: 01. 04. 2011

Janez PERME, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Simulacija izvajanja poslov na upravljanem porazdeljenem sistemu**

Simulation of task execution in a managed distributed system

Tematika naloge:
Porazdeljena okolja, kjer računalniški viri niso neposredno rezervirani (volonterni sistemi) so s cenovnega stališča še posebno zanimiva. V principu so lahko povsem brezplačna. Gre za izkoriščanje prostih (neuporabljenih) računalniških virov povezanih v visoko prepustno omrežje, ki so v veliki meri povsem neizkoriščeni.

V magistrski nalogi izdelajte model porazdeljenega okolja ter ga ustrezno simulirajte. S simulacijo pokažite obnašanje sistema v različnih pogojih delovanja. Osnovni parameter naj bo sprememba intenzivnosti vhodnih paketov. Pri tem opazujte obnašanje vseh pomembnih elementov porazdeljenega okolja.

Mentor:

prof. dr. Nikolaj Zimic

Somentor:

doc. dr. Andrej Brodnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 138-MAG-RI/2011
Datum: 01. 04. 2011

Janez PERME, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Simulacija izvajanja poslov na upravljanem porazdeljenem sistemu**

Simulation of task execution in a managed distributed system

Tematika naloge:
Porazdeljena okolja, kjer računalniški viri niso neposredno rezervirani (volonterni sistemi) so s cenovnega stališča še posebno zanimiva. V principu so lahko povsem brezplačna. Gre za izkoriščanje prostih (neuporabljenih) računalniških virov povezanih v visoko prepustno omrežje, ki so v veliki meri povsem neizkoriščeni.

V magistrski nalogi izdelajte model porazdeljenega okolja ter ga ustrezno simulirajte. S simulacijo pokažite obnašanje sistema v različnih pogojih delovanja. Osnovni parameter naj bo sprememba intenzivnosti vhodnih paketov. Pri tem opazujte obnašanje vseh pomembnih elementov porazdeljenega okolja.

Mentor:

prof. dr. Nikolaj Zimic

Somentor:

doc. dr. Andrej Brodnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

magistrskega dela

Spodaj podpisani/-a ____Janez Perme_____,

z vpisno številko ____63960108_____,

sem avtor/-ica magistrskega dela z naslovom

_____**Simulacija izvajanja poslov na upravljanem porazdeljenem sistemu**_____

_____**(Simulation of job execution in managed distributed system)**_____

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal/-a samostojno pod vodstvom mentorja (naziv, ime in priimek)

  ____prof. dr. Nikolaj Zimic_____

  in somentorstvom (naziv, ime in priimek)

  ___dr. Andrej Brodnik_____

- so elektronska oblika magistrskega dela, naslova (slov., angl.), povzetka (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela

- in soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne ____21.6.2011_____ Podpis avtorja/-ice: _____

ZAHVALA

Zahvaljujem se mentorju, prof. dr. Nikolaju Zimicu, in somentorju, doc. dr. Andreju Brodniku, za strokovno vodenje, nasvete in pomoč pri nastajanju magistrske naloge. Posebna zahvala gre tudi Zoranu Nikolovskemu in dr. Radovanu Sernecu za konstruktivne diskusije.

Janez Perme, Ljubljana, junij 2011.

KAZALO

SEZNAM UPORABLJENIH KRATIC

ALE	Aritmetično logična enota
API	Application program interface
ARC	Advanced resource connector
BOINC	Berkeley open interface for network computing
BSD	Berkeley software design
CERN	Conseil européen pour la recherche nucléaire
CGI	Common gateway interface
CPE	Centralno procesna enota
CPI	Clocks per instruction
DAG	Directed acyclic graph
DAGMan	Directed acyclic graph manager
DoS	Denial of service
DSL	Digital subsciber line
EGEE	Enabling grids for e-science
FC	Fibre channel
FIFO	First in, first out
FLOPS	Floating point operations per second
GPC	Global positioning system
GPU	Graphics processing unit
HPC	High performance computing
HTC	High throughput computing
HTTP	Hyper text transfer protocol
ILP	Instruction level parallelism
IP	Internet protocol
IPS	Instructions per second
IPTV	Internet protocol television
ISA	Instruction set architecture
JVM	Java virtual machine
LAN	Local area network
LB	Load balancing
MIMD	Multiple instruction stream, multiple data stream
MISD	Multiple instruction stream, single data stream
MMU	Memory management unit
MMX	Multimedia extension ali Multiple math extension ali Matrix math extension
MPI	Message passing interface
MPP	Massively parallel processing
NAM	Network animator
NAT	Network address translation
NFS	Network file system
NUMA	Non-uniform memory access
P2P	Peer-to-peer
PaaS	Platform as a service
PVM	Parallel virtual machine
RMComm	Resource management and communication
RPC	Remote procedure call
QoS	Quality of service
SaaS	Software as a service
SIMD	Single instruction stream, multiple data stream

SISD	Single instruction stream, single data stream
SLA	Service level agreement
SMP	Symmetrical multiprocessing
SSE	Streaming SIMD extensions
SSL	Space science laboratory
UMA	Uniform memory access
UNICORE	Uniform interface to computing resources
URL	Uniform resource locator
VLIW	Very long instruction word
VoIP	Voice over IP
WAN	Wide area network
XML	Extensible markup language

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Janez Perme

Simulacija izvajanja poslov na upravljanem porazdeljenem sistemu

POVZETEK

Reševanje kompleksnih računskih problemov s pomočjo računalnikov je dandanes praktično povsod navzoče. Vse več je problemov, ki so časovno in prostorsko zahtevni. Rešujejo se s pomočjo namenskih vzporednih sistemov. Večino sodobnih sistemov te vrste umeščamo skoraj izključno v Flynnov razred MIMD. Primeri so predvsem tesno povezani sistemi SMP in ohlapno povezani sistemi MPP ter grozdi. Posebno na področju znanstvenega računanja pa so se, kot cenovno ugoden nadomestek vzporednih sistemov, uveljavili tudi porazdeljeni sistemi. Primerni so predvsem za reševanje problemov, pri katerih je dekompozicija na podprobleme trivialna. Pomembna razlika med obema pristopoma je predvsem v namembnosti. Vzporedni sistemi so namenjeni predvsem pohitritvi reševanja problemov (hitrejši odzivni čas sistema), medtem ko so porazdeljeni sistemi namenjeni okoljem, kjer se skuša rešiti čim večje število problemov (večja prepustnost sistema).

Posebna vrsta porazdeljenih sistemov so prostovoljni sistemi, ki omogočajo izkoriščanje računalniških virov (procesnih, pomnilniških). Najbolj znana in razširjena sistema sta BOINC in Condor. Cilj obeh je isti: izkoriščati računalniške vire v času, ko so nedejavni. S sistemom BOINC izkoriščamo računalniške vire predvsem v prostranem okolju, medtem ko je sistem Condor primeren za zaprta okolja. Obe okolji pa sta lahko med seboj komplementarni.

Možnost proučitve lastnosti in ovrednotenje zmogljivosti takšnih sistemov sta ključna, saj omogočata učinkovitejše izkoriščanje računalniških virov. Po vzoru sistemov BOINC in Condor smo po definiranih poenostavitvah v ta namen zasnovali simulacijski model v obliki odprte strežne mreže s povratno zanko. Poenostavitve so bile nujno potrebne, saj bi bil v nasprotnem primeru simulacijski model težje obvladljiv. Strežno mrežo so tvorile centralna strežna enota in delovne strežne enote. Delovne strežne enote prejemajo posle oziroma opravila iz centralne strežne enote, jih izvajajo, rezultate pa pošiljajo nazaj na centralno strežno enoto. Preverba modela je potrdila pravilnost delovanja, vrednotenje pa pravilno zasnovo. Implementacija simulacijskega modela oziroma izvedba simulacije smo opravili v diskretnem odprtokodnem simulacijskem orodju ns-2, ki se je izkazalo kot vsestransko in izjemno prožno. Strežna enota je bila zasnovana s pomočjo dveh omrežnih vozlišč in njuno medsebojno omrežno povezavo. Streženje se je realiziralo s prenosom paketa protokola UDP preko omrežne povezave med dvema vozliščema.

Iz zasnovanega modela smo sklepali na obnašanje in zmogljivosti realnega modela. Definirani so bili parametri modela, katerih spremembe bistveno vplivajo na njegovo obnašanje (na primer: zmogljivost strežnih enot, število strežnih enot, tip razvrščanja, verjetnost povratne zanke).

Rezultati eksperimentov nad simulacijskim modelom so omogočali vpogled v obnašanje modela pri spremembah njegovih parametrov. Za vhodno intenzivnost smo uporabili Poissonov proces paketov UDP. Ugotovili smo, da z naraščanjem vhodne intenzivnosti narašča tudi obremenitev modela, pri čemer centralna strežna enota lahko hitro postane ozko grlo. Njeno obremenitev dodatno povečujeta tudi večje število delovnih strežnih enot in verjetnost vračanja paketov UDP v centralno strežno enoto. V primeru dovolj zmogljive centralne strežne enote pa ozko grlo lahko tvorijo delovne strežne enote. Naključno razvrščanje ne omogoča enakomerne obremenitve vseh delovnih strežnih enot, še posebno, če imajo te različno zmogljivost. V tem primeru prihaja pri manj zmogljivih delovnih strežnih enotah do hitrega porasta čakalnih vrst in posledično do nasičenja modela. Uravnoteženost dosežemo z uporabo razvrščanja po principu »najkrajša čakalna vrsta najprej«. Čakalne vrste delovnih strežnih enot so v tem primeru skoraj enako dolge. Z uporabo tega načina razvrščanja in različnih zmogljivosti delovnih strežnih enot naletimo do presenetljivega obnašanja. Zmogljivejše delovne strežne enote so samodejno nagnjene k sprejemanju daljših paketov UDP, manj zmogljive pa k sprejemanju krajših. To pomeni, da bi utegnili biti zmogljivejši prosti računalniški viri nagnjeni k izvajanju zahtevnejših opravil, medtem ko se manj zmogljivi prosti računalniški viri osredotočajo na izvajanje manj zahtevnih opravil.

Zasnovani model ni omejen samo na simulacijo prostovoljnega sistema. Uporaben je pri katerem koli sistemu, ki ga sestavljajo centralna enota in niz delovnih enot. Edina zahteva je, da centralna enota pošilja posle oziroma opravila delovnim enotam, te pa po izvedbi posredujejo rezultate nazaj na centralno enoto.

Ključne besede: porazdeljeni sistemi, vzporedni sistemi, BOINC, Condor, simulacija, strežna mreža, ovrednotenje zmogljivosti sistemov, simulacijsko orodje ns-2

University of Ljubljani
Faculty of Computer and Information Science

Janez Perme

Simulation of job execution in managed distributed system

ABSTRACT

Nowadays, the use of computers to solve complex computational problems is present virtually everywhere. There are more and more temporally and spatially complex problems. They are solved using purpose-built parallel systems. Most modern systems of this type fit almost exclusively into Flynn's MIMD category, examples of which are tightly coupled SMP systems, loosely coupled MPP systems, and clusters. Especially in the field of scientific computation, distributed systems have become established as an affordable substitute for parallel systems. They are suitable mainly for solving problems where the decomposition into subproblems is trivial. An important difference between the two approaches lies in its intended purpose. Parallel systems are designed to speed up problem solving (to enable a faster response time of the system), while distributed systems are mainly intended for environments where a high number of problems are solved (increased throughput of the system).

A special type of distributed systems are voluntary systems that allow the exploitation of computing resources (processing and memory resources). The most famous and widespread systems are BOINC and Condor. The aim of both is the same, i.e. to exploit the computing resources during idle time. The BOINC system is used to exploit computing resources in a wide area environment, while the Condor system is suitable for closed environments. Both can be mutually complementary.

The possibility of examining and evaluating the performance characteristics of such systems is crucial because it allows more efficient use of computing resources. Following the example of the BOINC and Condor systems and after simplifications were defined, a simulation model was designed in the form of an open queuing network with feedback. Simplifications were necessary since the simulation model would be otherwise difficult to manage. The queueing network consisted of a central queue and working queues. Working queues receive task from the central queue, carry them out and send the results back. The model verification confirmed the correctness of the operation, and the validation confirmed correct design. The implementation of the simulation model and of the simulation were carried out in the ns-2 discrete oriented open source simulation tool, which proved extremely versatile and flexible. The queue was designed with the help of two network nodes and their mutual network connection. The serving was realized by transferring the package via UDP network connections between nodes.

Based on the designed model assumptions can be made about the behavior and performance of the real model. Any changes to the model parameters that were defined (such as queue capacity, number of queues, routing type, the probability of a feedback loop) can significantly affect its behaviour.

The experiment results with the simulation model provided insight into the behavior of the model as its parameters were changed. For the input intensity, the Poisson process of UDP packets were used. It was found that as input intensity increases, so does the the model load, which means that the central server unit can quickly become a bottleneck. Its load further increases the greater the number of feeding units, and with the probability of returning UDP packets to the central queue. In the case of sufficiently powerful central queue, the bottleneck may be caused by working queues. Random routing does not allow for a steady load of working queues, especially if these have different performance. In this case a rapid increase in tasks to be performed, and consequently model saturation, follows working queues with worse performance. Balance is achieved by using the "shortest queue first" principle in routing. Working queues in this case are almost the same length.

The use of this method of routing and of working queues with different performance enables us to discover some surprising behaviour. Working queues with better performance are automatically inclined to receiving long UDP packets, while those with worse performance receive shorter UDP packets. This might mean that more efficient free computer resources tend to implement more complex tasks, while less powerful free computer resources focus on implementing less demanding ones.

The designed model is not restricted to simulations of a voluntary system. It is useful in any system consisting of a central unit and a set of work units. The only requirement is that the central unit sends jobs or tasks to the work units, which communicate the results back to the central unit upon completion.

Keywords: distributed systems, parallel systems, BOINC, Condor, simulation, queuing network, performance evaluation systems, ns-2 simulation tool

1 Uvod

1.1 Predstavitev problematike

V zadnjih 30-ih letih smo priča revolucionarnemu razvoju na področju elektronike in računalniške industrije. Približno vsaki dve leti so na trgu cenovno sprejemljive procesne enote, ki imajo podvojeno količino komponent (na primer tranzistorjev) na integriranem vezju. Težnja je znana pod imenom Mooreov zakon [47]. Čeprav imajo vsi večji proizvajalci zaradi fizikalnih omejitev že težave pri zasnovi hitrejših procesnih enot [8], so prepričani, da se bo Mooreov zakon obdržal vsaj do leta 2015. Želja po vedno večjih frekvencah sistemske ure in vse manjših integriranih vezjih privede do intenzivnejšega segrevanja oziroma disipacije. Tudi povezovanje podenot procesne enote (na primer ALE, MMU, predpomnilnik) na integriranem vezju postaja vse kompleksnejše. Proizvajalci so zato pričeli večjo zmogljivost procesnih enot dosegati z izkoriščanjem vzporednosti. Procesne enote so zasnovali večjedrno [37]. Poskušajo preiti tudi v tridimenzionalno integracijo [68], kar vsekakor ne bo preprosto, saj gre za zahteven tehnološki proces. Danes so štiri-, šest- ali večjedrne procesne enote povsem običajne. Srečujemo jih celo v vgrajenih sistemih [33] in prenosnih računalnikih. Privzem tega novega pristopa s strani uporabnikov ni povsem transparenten. Na obstoječem trgu večina uporabniških programov sploh ne izkorišča principa vzporednosti oziroma prisotnosti več procesorskih jeder. Prehod med klasičnim programiranjem in programiranjem, ki upošteva vzporednost, zato ne bo povsem preprost. Proizvajalci procesnih enot že ponujajo različna razvojna, programska orodja, programerji pa bodo vse bolj potrebovali znanja o principu vzporednosti. Do konkretne uvedbe novih tehnologij, kot so na primer kvantni procesorji [64], bo princip vzporednosti predstavljal začasno rešitev za povečevanje zmogljivosti procesnih enot.

Idejo oziroma princip uporabe več jeder lahko obravnavamo tudi splošnejše, na primer na področju porazdeljenega računalništva. Tudi tu imamo več "jeder", ki so pravzaprav računalniki in so lahko fizično ločeni. Gre za množico avtonomnih računalniških vozlišč, povezanih z omrežno tehnologijo v celoto, z namenom reševanja skupnega problema. Na tem področju se je razvilo kar nekaj tehnologij: grozdi, sistemi GRID, sistemi P2P, globalni in lokalni prostovoljni sistemi in vse bolj razširjena okolja v oblakih (Amazon, Google, SalesForce). Zavedati se je treba, da se je način reševanja problemov, še posebno na področju znanosti, v zadnjih dveh desetletjih povsem spremenil. Uporabniki se vse bolj soočajo z novimi vrstami problemov, ki so sicer rešljivi, a zahtevni. Klasičen pristop je uporaba superračunalnika z večjim številom zmogljivejših virov (večjedrne centralne procesne enote, večje količine pomnilnika, hitri in visokoprepustni vhodno/izhodni vmesniki). Kot tak ni prav nič sporen, je pa gotovo stroškovno težje opravičljiv. Veliko cenejši pristop omogoča porazdeljeno računalniško okolje obstoječih namiznih računalnikov ali strežnikov z eno- ali večjedrnimi procesnimi enotami. Tehnološki napredek strojne opreme, hitro širjenje in prodor omrežja internet ter širokopasovne komunikacijske povezave omogočajo zanesljivo in kakovostno povezovanje geografsko porazdeljenih računalniških virov v enovito celoto. Dostop do teh je za uporabnike povsem transparenten.

Uporabniki lahko brez večjih ovir zasnujejo računalnik z zmogljivostjo, ki je primerljiva z zmogljivostjo superračunalnika. Takšne zmogljivosti so bile dostopne samo organizacijam, ki so imele več finančnih sredstev. Ravno zato je porazdeljeno računalništvo že dlje časa eno od bolj zaželenih, visokozmogljivih oziroma visokoprepustnih pristopov za reševanje različnih vrst računskih problemov. Viri v takšnih okoljih so lahko popolnoma heterogeni. Lahko so povsem običajni namizni računalniki, zmogljivi strežniki, grozdi, superračunalniki ali bolj namenski računalniški viri, kot so na primer strojni grafični vmesniki ali celo mobilne naprave. Torej vse naprave, ki so zmožne računati in se povezovati v celoto.

Porazdeljeni sistemi, kjer računalniški viri niso neposredno rezervirani (prostovoljni sistemi) so s stroškovnega stališča še posebno zanimivi. V principu so povsem brezplačni. Gre za izkoriščanje prostih oziroma neizkoriščenih računalniških virov, povezanih s pomočjo visokoprepustnega omrežja. To so viri, ki jih njihovi lastniki v nekem trenutku ne uporabljajo, se pa zato lahko izkoristijo v drugačne namene (na primer znanstveno-raziskovalne in komercialne). Cilj je združiti procesne in pomnilniške zmogljivosti takšnih računalniških virov v celoto in jo uporabiti za reševanje problemov v znanosti, filmski industriji, v finančnih analizah in podobno.

Način reševanja problemov v porazdeljenem sistemu je veliko bolj praktičen, manj kompleksen in zato bližji tudi običajnim uporabnikom. Ni pa univerzalen. Vrsta problema, ki se rešuje, mora biti takšna, da se ga s porazdeljenim sistemom da reševati. To so predvsem problemi, ki jih je mogoče razdeliti na več časovno in prostorsko manj zahtevnih avtonomnih podproblemov – torej takšnih, ki so med seboj ohlapno povezani. To pomeni, da ne zahtevajo veliko medsebojne komunikacije. Ključno je, da končni uporabnik porazdeljenega sistema razume obnašanje oziroma princip delovanja takšnega sistema in da je seznanjen z vsemi zahtevami in omejitvami, ki jih takšen sistem ima.

Na podlagi pregleda dveh obstoječih sistemov, BOINC in Condor, ter ustreznih poenostavitev bomo v magistrski nalogi predstavili možnost proučitve lastnosti in ovrednotili zmogljivosti poenostavljenega prostovoljnega sistema. Računalniška simulacija je koristna za sistemske zmogljivostne analize. Še posebej, ko sistema, ki ga želimo ovrednotiti, ni na voljo. Omogoča nam veliko lažje razumevanje delovanja modela, z analizo rezultatov pa lahko dovolj dobro ocenimo smernice obnašanja realnega okolja pod različnimi pogoji. V ta namen bo zasnovan simulacijski model, ki bo temeljil na prilagojeni odprti mreži strežnih enot M/M/1 s povratno zanko, implementiran pa bo v diskretnem simulacijskem orodju ns-2. Omogočal bo vpogled v lastnosti in napovedovanje obnašanja poenostavljenega prostovoljnega sistema pri parametrih, ki ga karakterizirajo: število strežnih enot, vhodna intenzivnost, zmogljivost strežnih enot, tip razvrščanja in verjetnost vračanja v zanki. V odvisnosti od sprememb omenjenih parametrov so zanimive predvsem spremembe odzivnega časa sistema, velikost zahtevanih pomnilniških kapacitet strežnih enot, interval vhodne intenzivnosti, v katerem je sistem uporaben, in spremembe prepustnosti sistema.

1.2 Metodologija

K problemu smo pristopili tako, da smo v začetku opravili klasifikacijo in pregled obstoječih tehnologij, ki se uporabljajo na področju vzporednega in porazdeljenega računalništva. Pozornost smo posvetili predvsem dvema prostovoljnima sistemoma: BOINC in Condor. Primerjali smo oba pristopa. Na podlagi poenostavitev smo zasnovali simulacijski model. Izvedbo simulacije smo opravili s pomočjo odprtokodnega orodja ns-2. S preverbo in

vrednotenjem smo zagotovili verodostojnost modela. Pri eksperimentalnem delu so bili natančno definirani scenariji, katerih analiza je privedla do ovrednotenja prostovoljnega sistema in s tem zaključka študije.

1.3 Prispevki magistrske naloge

Magistrska naloga vsebuje naslednje izvirne prispevke:

1. Zasnova generičnega konceptualnega modela poenostavljenega prostovoljnega sistema na podlagi bistvenih lastnosti obstoječih realnih sistemov BOINC in Condor;
2. Implementacija simulacijskega modela s pomočjo diskretnega simulacijskega orodja ns-2;
3. Analiza obnašanja in ovrednotenje zmogljivostnih parametrov simulacijskega modela pri spremembah njegovih karakterističnih parametrov.

2 Vzporedni in porazdeljeni sistemi

Načrtovalci računalniških sistemov vedno znova presenečajo z vse zmogljivejšimi računalniškimi sistemi. V zadnjih dveh desetletjih je to še posebno očitno. Visokozmogljivi računalniški sistemi danes omogočajo izjemno hitro izvajanje večjega števila operacij v enoti časa. Predvsem v znanstvenoraziskovalnih organizacijah je takšna računska moč več kot zaželena. Hitremu širjenju razvoja in raziskovanja sistemov, ki zagotavljajo takšne zmogljivosti, botrujejo predvsem prostorsko in časovno zahtevni problemi, želja po večji prepustnosti in hitrem izvajanju kompleksnih algoritmov. Na področju raziskav si preprosto želimo reševati izjemno zahtevne probleme, želimo jih rešiti čim več in čim hitreje. Ko poskušamo visokozmogljive sisteme klasificirati, nemudoma naletimo na dilemo o tem, kaj pravzaprav je vzporedni in kaj porazdeljeni sistem? Ali sta pristopa diametralna ali imata vendarle stične točke? Kakšne so pravzaprav razlike in sinergije med obema?

2.1 Vzporedni sistemi

Dandanes integrirana vezja (čipi) premorejo več kot dve milijardi tranzistorjev. Vse od prvih integriranih vezij je bilo povečevanje tega števila hitro in se še danes omenja skupaj s tako imenovanim Moorovim zakonom. Soustanovitelj podjetja Intel, Gordon E. Moore, je namreč leta 1965 objavil članek [40, 47], v katerem je napovedal, da se bo število tranzistorjev na čipu vsako leto podvojilo. V napovedi se je omejil na 10 let, vendar jo je kmalu, leta 1975, popravil in napovedal podvajanje števila tranzistorjev na vsaki 2 leti. Od leta 1971 se je število tranzistorjev na 2 leti tudi resnično podvojilo. Danes podjetje Intel sledi Mooreovemu zakonu z uvedbo popolnoma nove arhitekture ali pa tehnološkim krčenjem že obstoječe. Gre za tako imenovani *tick-tock* pristop.

Eksponentno hitro spreminjanje tehnologije v veliki meri vpliva na razvoj in proizvajanje računalniških sistemov. Pri strukturnih zasnovah procesorjev je zelo hitro prihajalo do tega, da so pričeli proizvajalci na integrirana vezja poleg procesorja vgrajevati tudi predpomnilnike in podenote za računanje v plavajoči vejici. S tem so bistveno zmanjšali stroške proizvodnje računalnikov, hkrati pa pospešili delovanje. Od leta 1971 pa vse do danes se je velikost katerega koli objekta v dimenziji X ali Y na integriranem vezju zmanjšala od nekaj 10 mikronov na velikost 32 nm. Po napovedih podjetja Intel se bo ta zmanjšala tudi na 11 nm. Gre za tako imenovano lastnost *feature size* oziroma vrsto tehnologije, ki določa število tranzistorjev na integriranem vezju. Objekt je lahko tranzistor ali del tranzistorja, povezava ali presledek med dvema objektoma. Povečevanje števila tranzistorjev je neposredno povezano s kvadratom zmanjševanja velikosti objekta. Tako nam prehod s tehnologije 65 nm na 32 nm pri enaki površini zagotavlja, da število tranzistorjev početverimo. Povečevanje hitrosti tranzistorjev pa je problem, ki je povsem drugačne narave in povezan s fizikalnimi lastnostmi materialov. Z zmanjševanjem dimenzij se poslabšujejo tudi lastnosti povezav elementov integriranega vezja. Zakasnitev v povezavah je odvisna od upornosti in kapacitivnosti povezave [28]. Čeprav so povezave krajše, so tudi tanjše, s tem pa se zakasnitev na enoto dolžine povečuje. Problem je bil sicer rešen z zamenjavo aluminijastih povezav z bakrenimi, ki imajo nižjo upornost, vendar se je izkazalo, da je ukrep le začasen in da je problem zakasnitve zaradi narave povezav še vedno pereč. Ena od posledic je tudi počasnejše

povečevanje frekvence sistemske ure, še posebno po letu 2000. Na to vpliva tudi poraba energije in z njo povezano sproščanje toplote na integriranem vezju. Pri preklopih tranzistorjev se namreč porablja precejšnja količina moči, ki je odvisna od kapacitivne obremenitve tranzistorja, napajalne napetosti in frekvence preklapljanja [28]. Kapacitivnosti so sicer zelo majhne, je pa zato veliko tranzistorjev in pri današnjih procesorjih se med delovanjem sprošča moč, ki presega tudi 100 W. Problematično je tudi tokovno uhajanje (izvorno: *leakage currents*), ki še dodatno povečujejo porabo energije. Odvajanje toplote s površine nekaj kvadratnih centimetrov velikega integriranega vezja je eden večjih problemov. Bil bi občutno večji, če proizvajalci ne bi hkrati zmanjševali tudi napajalne napetosti integriranega vezja. Problem odvajanja toplote s povečevanjem frekvence sistemske ure narašča in predstavlja omejitveni dejavnik za nadaljnje povečevanje hitrosti delovanja integriranih vezij oziroma procesnih enot. Pri procesnih enotah, ki so zasnovane na klasični, Von Neumannov način, je težko doseči majhen CPI. Število ukazov, ki jih CPE doseže v času 1 sekunde, definiramo s pomočjo izraza

$$IPS = \frac{f_{CPE}}{CPI}.$$

f_{CPE} je frekvenca sistemske ure, CPI pa povprečno število urinih period, ki so potrebne za izvedbo posameznega ukaza. Število izvedenih ukazov v času 1 s lahko povečamo na dva načina: z uporabo hitrejših elementov (večja frekvenca sistemske ure f_{CPE}) ali z uporabo večjega števila elementov (manjši parameter CPI). Uporaba hitrejših elementov ne omogoča občutnega povečanja hitrosti zaradi tehnoloških omejitev. Napredek v tehnologiji CMOS se bo počasi pričel ustavljati. V to so prepričani vsi večji proizvajalci procesnih enot [68]. Ostane le možnost izkoriščanja prostora na integriranem vezju. V času, ko bo cena komponente integriranega vezja (tranzistorja) še naprej padala, poraba energije pa se bo ustalila, je pričakovati še dodaten prehod. Na današnjih procesnih enotah je namreč poraba energije še sprejemljiva. Zaradi povečevanja števila komponent na integriranem vezju in s tem večji porabi energije, ne bomo utegnili uporabljati vseh komponent integriranega vezja hkrati. Manjšo porabo energije lahko sicer dosežemo z nižjo frekvenco sistemske ure, a to pomeni, da se izvajalni tok ukazov aplikacije lahko bistveno upočasni. Ta rešitev bi morebiti zahtevala večje posege v obstoječe aplikacije in zato ni zaželena. Privlačnejša rešitev za nižanje porabe energije je uporaba namenskih procesnih jeder, ki bi se aktivirala le po potrebi. Začetke tega pristopa je zaslediti pri procesni enoti Cell [36, 37], zasnovani za doseganje večjih zmogljivosti večpredstavnostnih aplikacij.

Von Neumannova arhitektura računalniškega sistema je že dolgo ena najbolj razširjenih računalniških arhitektur, vendar pa je zanjo značilno zaporedno izvajanje ukazov počasno. Vzrok je ozko grlo med procesno enoto in pomnilnikom. Prva razširitev Von Neumannove arhitekture je bila tehnika, ki omogoča izkoriščanje vzporednosti na nivoju ukazov (ILP). Predvsem v znanosti in tehnologiji se dandanes srečujemo s problemi, ki po svoji naravi dovoljujejo istočasno izvrševanje več med seboj neodvisnih operacij. Z drugačno zgradbo procesne enote (dodatnimi logičnimi elementi) se skuša doseči izvajanje več aktivnosti hkrati in s tem manjši CPI. Najbolj razširjena tehnika je uporaba cevovoda (izvorno: *pipeline*) [39]. Gre za način izvajanja več ukazov v CPE hkrati, tako da se posamezni koraki izvrševanja ukazov prekrivajo. Celotna obdelava ukaza se razdeli na več preprostejših podoperacij, za izvajanje vsake izmed teh skrbi poseben del procesne enote. Posamezna podoperacija se zato izvaja le del celotnega časa izvajanja ukaza. Vse skupaj deluje kot delo na tekočem traku, ki je pravzaprav bistvo cevovoda. Tipične dolžine cevovoda danes so blizu vrednosti 15. Mikroarhitektura Haswell ima 14 stopenj. Večje vrednosti redkeje srečujemo. Intel svojega

načrtovanega 45-stopenjskega cevovoda v jedru Tejas ni in verjetno tudi nikoli ne bo realiziral. Problematična je predvsem poraba energije in z njo povezan problem hlajenja. Pri dolžini cevovoda *N* bi moralo biti v idealnem primeru število urinih period za ukaz *N*-krat manjše kot pri necevovodni CPE. V resnici pa zaradi strukturnih, podatkovnih in kontrolnih cevovodnih nevarnosti [39] takšne pohitritve ni možno doseči. Do cevovodnih nevarnosti prihaja zaradi medsebojne odvisnosti ukazov v cevovodu. Odpravljamo jih programsko ali pa s posebno logiko v procesni enoti. S cevovodnim pristopom lahko sicer dosežemo CPI, ki je blizu vrednosti 1. Če pa želimo zmanjšati vrednost pod 1, moramo v vsaki urini periodi prevzeti in izvesti več ukazov. V tem primeru govorimo o tako imenovanih večizstavitvenih procesorjih. Poznamo dve vrsti:

1. **superskalarni procesorji** so cevovodne procesne enote, sposobne hkrati prevzemati, dekodirati, izvrševati in shranjevati več ukazov. Procesna enota, ki jo sestavlja več funkcijskih podenot, dinamično določa vrstni red prevzemanja in izstavljanja ukazov. To pomeni, da lahko prevzema ukaze v drugačnem vrstnem redu, kot so nanizani v programu. Ko je poljubna podenota prosta, se poišče ukaze, ki bi jih bilo možno na njej izvrševati. Vrstni red izstavljanja prevzetih ukazov določa logika neposredno v procesni enoti. Vzporednost se torej pri teh procesnih enotah odkriva v času izvajanja programa. Superskalarni procesorji imajo to lepo lastnost, da omogočajo povečanje zmogljivosti procesorja brez nepotrebnega poseganja v obstoječe aplikacije;
2. **procesorji VLIW** omogočajo, da se vzporednost odkriva v času prevajanja aplikacije. Torej preden aplikacijo sploh pričnemo izvajati na procesni enoti. Ideja je, da ima prevajalnik pregled nad celotnim programom in lahko zato v principu odkrije več vzporednosti. Pričakovali bi, da so procesorji vrste VLIW zaradi tega strojno bistveno manj zahtevni in bi lahko delovali tudi pri višjih frekvencah sistemske ure. Izkazalo pa se je, da so najmanj toliko zapleteni kot superskalarni, frekvenca sistemske ure pa ni višja.

Količina vzporednosti na nivoju ukazov je omejena in vsi večji proizvajalci procesorjev so mnenja, da je tehnologija na robu dosegljivega. Kljub temu lahko vzporednost nadalje izkoriščamo na višjih nivojih – vzporednost na nivoju niti [39]. Nit je pravzaprav tok ukazov in operandov, ki je povsem ločen od druge niti oziroma toka drugih ukazov in operandov. Je lahko del enega programa (program je običajno sestav več programskih procesov) ali celo povsem ločen program. Večnitnost (izvorno: *multithreading*) označuje delovanje, pri katerem si več niti deli funkcijske enote procesorja. Vsaka izmed niti ima svoje stanje, ki ga določajo programski števec, registri, tabele strani navideznega pomnilnika in programsko nevidni registri. Vse pa si delijo isti predpomnilnik in glavni pomnilnik. S stališča niti je videti, kot da je celoten procesor namenjen le eni niti.

Pri današnjih procesorjih ločimo dva načina večnitnosti [21, 39]:

1. **časovna večnitnost** (izvorno: *temporal multithreading*) − tu se nekaj časa izvršuje ena nit, nato druga, tretja, vse do zadnje in nato spet prva. Glede na preklapljanje med nitmi ločimo tudi drobnozrnato (izvorno: *fine-grain multithreading*) in grobozrnato (izvorno: *coarse-grained multithreading*) večnitnost. Pri prvi vrsti večnitnosti se naredi preklop med nitmi vsako urino periodo, pri drugi pa le takrat, ko pride pri neki niti do daljšega čakanja – na primer zgrešitev v predpomnilniku;

2. **istočasno večnitnost** (izvorno: *symultaneous multithreading* ali *hyper-threading*) lahko uporabimo le pri večizstavitvenih procesorjih z dinamičnim prevzemanjem (oziroma razvrščanjem) in izstavljanjem. Te procesne enote imajo običajno na voljo več funkcijskih enot, kot jih lahko izkoristi ena nit. Ker ukazi ene niti niso odvisni od ukazov druge niti, je procesna enota lahko poenostavljena. Tipična števila hkratnih niti so 2, 4 ali tudi 8.

Zaradi tehnoloških omejitev so pričeli večji proizvajalci procesnih enot izkoriščati razpoložljivi prostor na integriranem vezju tudi tako, da procesno enoto zasnujejo kot večjedrno [1, 33, 39, 44, 68] (izvorno: *multicore processor*). Gre za takšne procesne enote, ki imajo na istem integriranem vezju več neodvisnih procesnih podenot – jeder. Vsako jedro je enota zase in ima običajno svoj predpomnilnik (izvorno: *cache*). Na višjih nivojih (običajno 3. nivoju) predpomnilniške arhitekture pa imajo jedra v procesni enoti tudi skupen predpomnilnik. Če se niti izvršujejo vsaka na svojem jedru, dobimo pravo vzporednost. Kljub temu je zaradi skupnega pomnilnika deloma še vedno prisotna medsebojna odvisnost. Večina današnjih procesorjev je zasnovana tako, da je vsako izmed jeder tudi večnitno. Proizvajalci veliko lažje vgrajujejo več kopij iste procesne enote na integrirano vezje, kot vlagajo v sredstva za razvoj novih arhitekturnih sprememb na procesnih enotah. Mnenja so tu deljena. Vsekakor pa se odpira problem pri končnih uporabnikih, ki jih proizvajalci surovo silijo v pisanje večnitnih programov.

Uporaba večjedrnih procesnih enot je razširjena na področjih, kjer je izkoriščanje vzporednosti možno, na primer spletnih in podatkovnih strežnikih, prevajalnikih, znanstvenih aplikacijah in aplikacijah, ki uporabljajo večpredstavnost. Tipični primer večjedrne arhitekture je Intelova procesna enota z najnovejšo mikroarhitekturo Nehalem [11].

Nadaljnja razširitev Von Neumannove arhitekture je množica med seboj povezanih avtonomnih računalniških sistemov, ki je lahko med drugim tudi geografsko razpršena. Obstajajo tudi druge vzporedne arhitekture oziroma pristopi, kot so sistolični procesorji, podatkovno pretokovni modeli, nevronske mreže in podobno. V principu je razširitev Von Neumannovega modela računalnika mogoče zajeti s tako imenovano Flynnovo klasifikacijo [56]. Osnovna kriterija te klasifikacije sta število ukazov in število podatkovnih enot, ki se v določenem trenutku obravnavajo v računalniškem sistemu. Znane so tudi druge vrste klasifikacij: Dasguptova [20], Hockneyjeva [32] in Bellova [14], vendar je daleč najbolj razširjena Flynnova. Računalniške sisteme po Flynnovi klasifikaciji ločimo v 4 skupine: SISD, SIMD, MISD in MIMD.

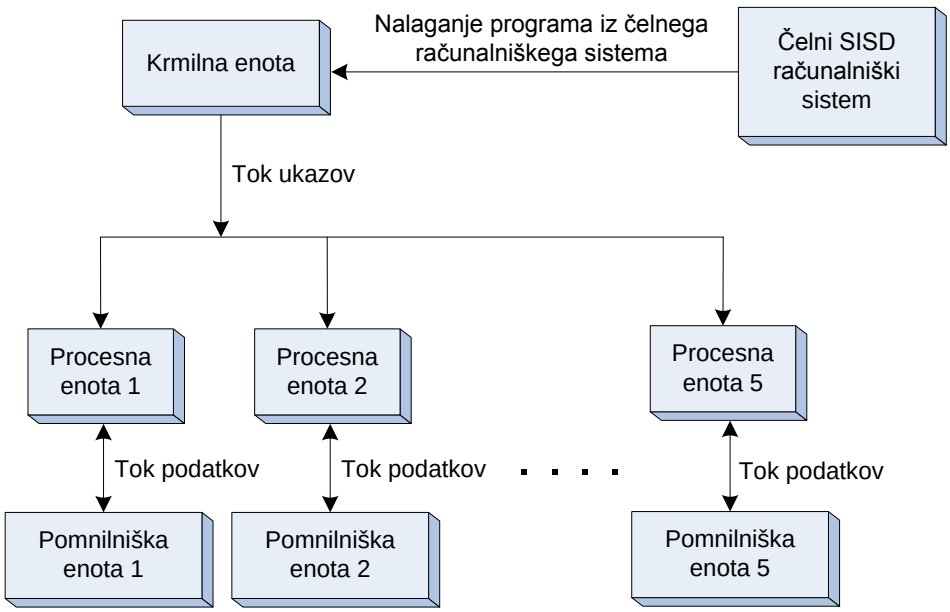
2.1.1 Sistemi SISD

Pri sistemih SISD [22, 39] gre za klasične Von Neumannove računalniške sisteme, ki imajo vgrajeno eno centralno procesno enoto (procesor). Ta bere iz pomnilnika po en ukaz naenkrat (en tok ukazov) in se izvede le nad enim podatkom hkrati (en tok podatkov) v pomnilnik. Vse arhitekture SISD imajo na voljo le en programski števec. Po branju enega ukaza iz pomnilnika se poveča tako, da kaže na naslednji ukaz in s tem omogoča zaporedno izvajanje ukazov.

2.1.2 Sistemi SIMD

SIMD sistemi [22, 39] imajo krmilno enoto, ki sprejema po en ukaz (en tok ukazov). Ta se izvaja nad več podatki hkrati (več tokov podatkov). Tak sistem ima vedno več procesnih enot. Te upravlja krmilna enota, ki proži enake krmilne signale za vse procesne enote hkrati, vsaka izmed njih pa ima svoj tok podatkov. V resnici je tak sistem sestav dvojega (slika 2.1): čelne

enote, ki je običajno kar Von Neumannov sistem oziroma sistem SISD, in polja identičnih procesnih enot s pripadajočo krmilno enoto. Naloga čelnega sistema je, da poskrbi za prenos programa v krmilno enoto. Ima svoj pomnilnik, iz katerega bere ukaze in jih izvaja zaporedno. S posebnimi ukazi lahko dostopa do pomnilnika katere koli procesne enote, podobno, kot dostopa do svojih lokalnih pomnilniških lokacij. Z ukazi pa lahko opravlja tudi prenose podatkov med posameznimi procesnimi enotami. Procesne enote hkrati izvajajo ukaze iste vrste nad različnimi podatki, ki se nahajajo v pomnilniški enoti, povezani s procesno enoto. Ukaze izvajajo v tako imenovanem načinu *lock-step*, kar pomeni, da ne izvajajo nobenega ukaza in mirujejo ali pa izvajajo enake ukaze hkrati.



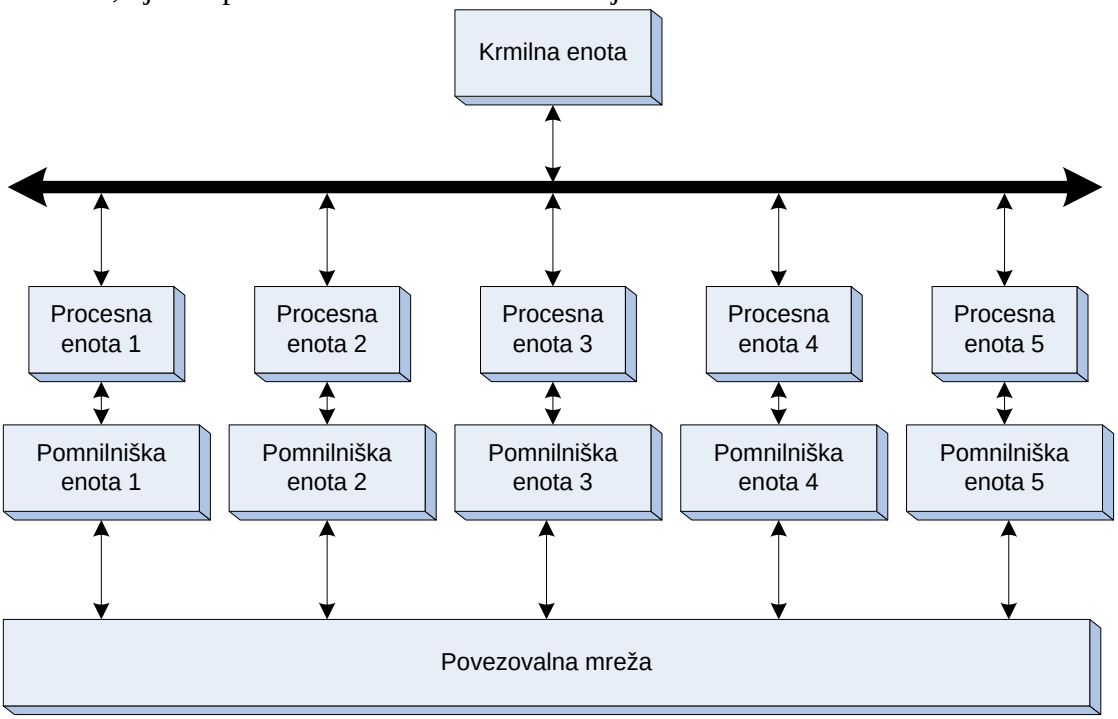
Slika 2.1: Koncept arhitekture SIMD

Prenos podatkov med procesno enoto in pomnilniško enoto lahko poteka na dva načina:

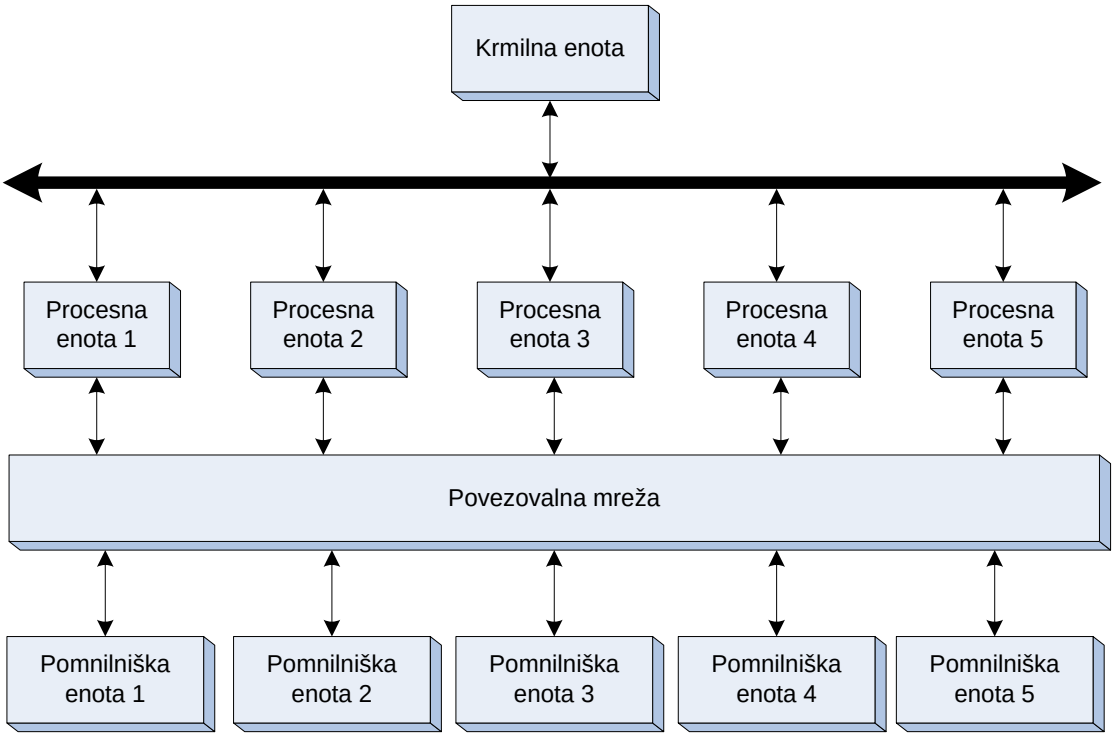
1. pri prvem načinu (slika 2.2) je vsaka procesna enota povezana neposredno s svojo pomnilniško enoto. Prenos podatkov med procesnimi enotami poteka preko posebne povezovalne mreže. Če v povezovalni mreži povezava med dvema procesnima enotama ne obstaja, lahko komunikacija med njima poteka posredno preko druge procesne enote. Ta način uporablja sistem ILLIAC IV [22, 41, 49];
2. pri drugem načinu (slika 2.3) so procesne in pomnilniške enote povezane posredno preko posebne povezovalne mreže. Prenos podatkov tako med procesnimi in pomnilniškimi enotami kot tudi med samimi procesnimi enotami poteka izključno preko povezovalne mreže. Če v povezovalni mreži povezava med dvema procesnima enotama ne obstaja, lahko komunikacija med njima poteka posredno preko druge procesne enote. Ta način uporablja sistem *Burroughs' Scientific Processor* [22, 41, 49].

Povezovalne mreže so lahko statične, pri katerih je povezava vnaprej določena, ali dinamične, kjer se pot določa šele v času komunikacije med procesnimi enotami. V prvi skupini je najpogostejša uporaba hiperkocke, pri dinamičnih mrežah pa so najbolj razširjene večstopenjske stikalne povezave [61]. Med sisteme SIMD uvrščamo tudi tako imenovana

polja procesorjev ali drugače imenovane vektorske procesorje [18, 39]. Primerni so predvsem za reševanje problemov, kjer prevladuje velika podatkovna regularnost, malo skočnih ukazov oziroma tam, kjer so podatki v veliki meri medsebojno neodvisni.



Slika 2.2: Prvi način povezovanja procesnih in pomnilniških enot v arhitekturi SIMD



Slika 2.3: Drugi način povezovanja procesnih in pomnilniških enot v arhitekturi SIMD

Tipičen primer so aplikacije za procesiranje slik in digitalnega procesiranja signalov, kjer se opravi veliko število operacij z vektorji in matrikami. Kljub temu, da so vektorski procesorji togi in jim težje povečujemo zmogljivosti, so dobro poznani in uveljavljeni. Pisanje programov je preprosto, obstoječi prevajalniki pa so zelo učinkoviti. Primera vektorskega procesorja sta CRAY C90 in NEC SX4. Zelo razširjen procesor, ki uporablja 8 128-bitnih procesorjev SIMD, je procesor Cell [36]. Uporablja se predvsem v igralnih konzolah in celo za znanstveno računanje. Pristop SIMD vgrajuje v svoje procesorje tudi podjetje Intel. Tehnologiji MMX in SSE sta primera razširitve njihovih procesorjev na pristop SIMD. Omogočata povečanje zmogljivosti pri izvajanju istih operacij nad več podatki.

2.1.3 Sistemi MISD

Računalniški sistemi vrste MISD [22, 39] lahko nad enim podatkom (en tok podatkov) izvajajo več ukazov hkrati (več tokov ukazov). Danes jih praktično ni zaslediti nikjer v svetu. Obstaja pa sicer drugačen način interpretacije tega pristopa: podatek, ki se obdeluje skozi niz procesnih enot. V to skupino spadajo cevovodne arhitekture, ki jih imenujemo sistolična polja [56]. Pri teh sistemih je pomembno, da se opravi čim več operacij nad podatkovno strukturo, dokler se ta nahaja v pomnilniku. Nad podatkovno strukturo (skupek več podatkovnih enot – običajno je to vektor ali matrika) se v vsaki stopnji cevovoda izvede ukaz. Tipični primeri uporabe so predvsem pri operacijah nad matrikami na področju obdelave slik in signalov. Od vseh arhitektur so ti računalniški sistemi najmanj razširjeni.

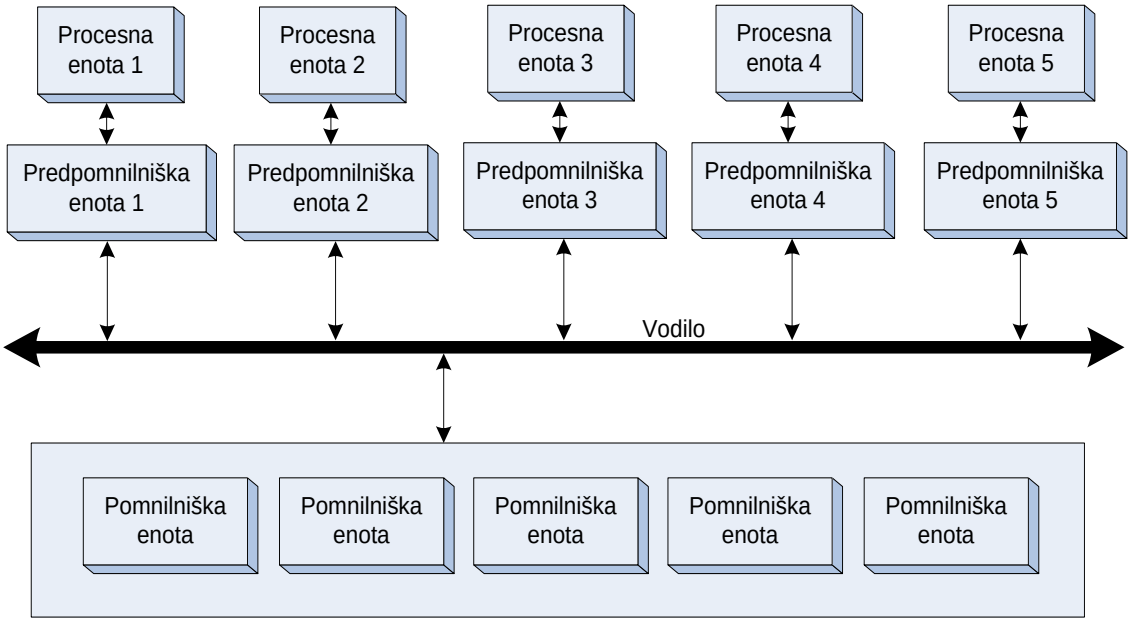
2.1.4 Sistemi MIMD

Glavna razlika med arhitekturama SIMD in MIMD [22, 39] je, da so v slednjih procesne enote povsem avtonomne. Vsaka procesna enota ima svojo krmilno podenoto in podenoto ALE. Zato lahko izvajajo programe povsem neodvisno (več tokov ukazov in podatkov). V sistemih MIMD ne obstaja skupna sistemska ura. Takšni sistemi so asinhroni. Procesne enote morajo biti programirane tako, da so med seboj usklajene. Glede na način uporabe pomnilnika sisteme MIMD ločimo v dve skupini: sisteme s souporabnim pomnilnikom (izvorno: *shared memory system*) in sisteme s porazdeljenim pomnilnikom (izvorno: *distributed memory system*). Prve imenujemo večprocesorski sistemi [13] in druge večračunalniški sistemi [13].

V sistemih s souporabnim pomnilnikom [41] procesne enote pošiljajo in sprejemajo podatke s pomočjo pisanja in branja na skupne pomnilniške lokacije. Vse procesne enote imajo lahko dodane tudi medpomnilnike (izvorno: *buffers*), registre, predpomnilnike in dodatne lokalne pomnilnike. Morebitno hkratnost dostopov do pomnilniških enot se rešuje s posebno usklajevalno enoto, ki skrbi za ustrezno posredovanje zahtev procesnih enot pomnilniškemu krmilniku. V trenutku dostopanja do poljubne pomnilniške enote krmilnik poskrbi, da so vsa nadaljnja dostopanja onemogočena, dokler se ne konča dostop, ki je v teku. V sklopu sistemov, ki uporabljajo princip souporabnega pomnilnika, ločimo dva najpogostejša načina povezovanja pomnilniških enot s procesnimi enotami: sisteme UMA in NUMA.

Souporabni pomnilnik v sistemih UMA (ali tudi SMP) tvori en naslovni prostor. Vse procesne enote dostopajo do pomnilnika na enak način (kot bi dostopale do svojega lokalnega pomnilnika) in z enakim dostopnim časom. Možnost hkratnega dostopanja do skupnega pomnilnika sicer lahko povzroči manjši padec zmogljivosti, ki pa ni kritičen. Zaradi prostorske in časovne lokalnosti se s pomočjo uporabe predpomnilnikov skuša padec omiliti. Hkrati pa se z njihovo uporabo odpira nov problem: problem skladnosti predpomnilnikov procesnih enot, ki se rešuje z vohunjenjem (izvorno: *snooping*) ali posebnimi algoritmi [39].

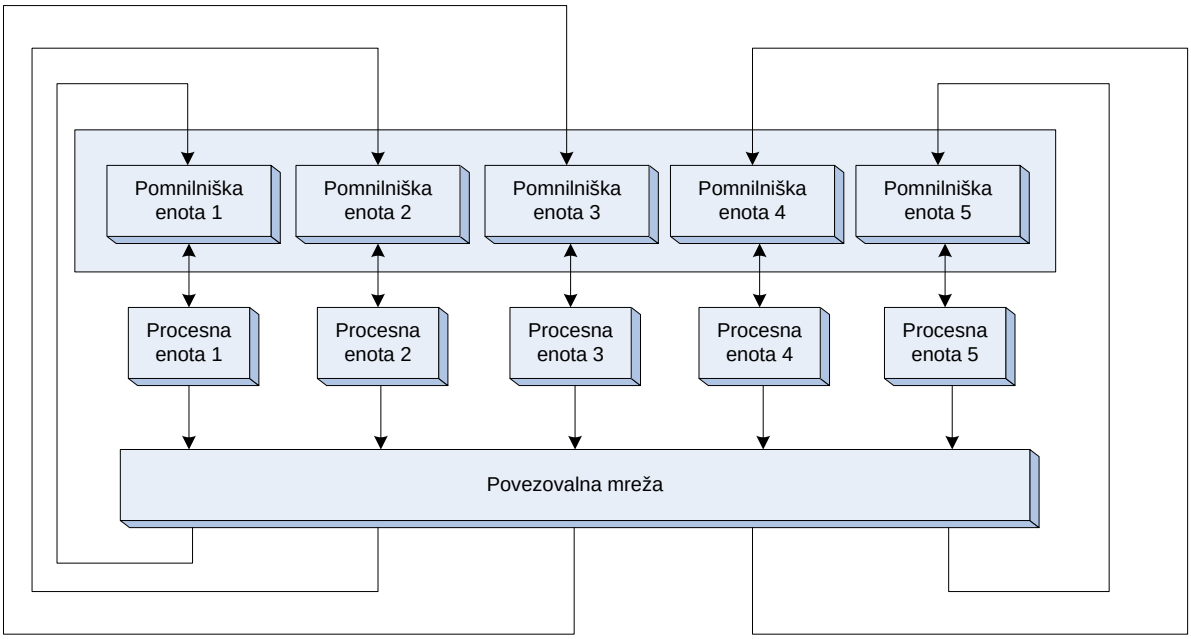
Ta pristop omogoča, da so vsi predpomnilniki vedno vsebinsko skladni. Običajna povezovalna mreža, ki se uporablja v sistemih UMA, je lahko enojno vodilo, več vodil ali uporaba prečnikov (izvorno: *crossbar switch*). Slika 2.4 prikazuje sistem UMA z uporabo vodila. Pri razširitvi sistema z večjim številom procesnih enot bi postalo vodilo ozko grlo sistema, rešitev z uporabo prečnikov pa bila bistveno dražja.



Slika 2.4: Sistem UMA z uporabo enega vodila in predpomnilniških enot

Sistemi NUMA (slika 2.5) so tesno povezani sistemi, v katerih je sleherna procesna enota povezana z delom souporabnega pomnilnika. Običajno imajo dodane tudi predpomnilnike. Tudi tu je sicer le en naslovni prostor in pomeni, da vsaka procesna enota do pomnilniške lokacije dostopa z njenim neposrednim naslovom. Čas dostopa do pomnilnika je odvisen od lokacije procesne enote in je zato za vsako procesno enoto drugačen.

Za povezovanje se pogosto uporabljajo posebne regularne in simetrične topologije, na primer mreža s povratnimi vezavami (torus) in 4-D hiperkocka. Če bi želeli število procesnih enot povečati, bi bil zaradi večje kompleksnosti povezovalne mreže in protokolov za ohranjanje skladnosti predpomnilnikov čas dostopa do pomnilniških modulov bistveno večji. Izjema je seveda lokalni pomnilnik. Porazdelitev podatkov po pomnilniških enotah v teh sistemih sicer ni kritična, je pa zaželena, saj so dostopni časi do pomnilnikov različni. Primer sistema NUMA na področju procesorjev je Intel Core i7 z mikroarhitekturo Nehalem.



Slika 2.5: V sistemih NUMA je sleherna procesna enota povezana le z delom souporabnega pomnilnika

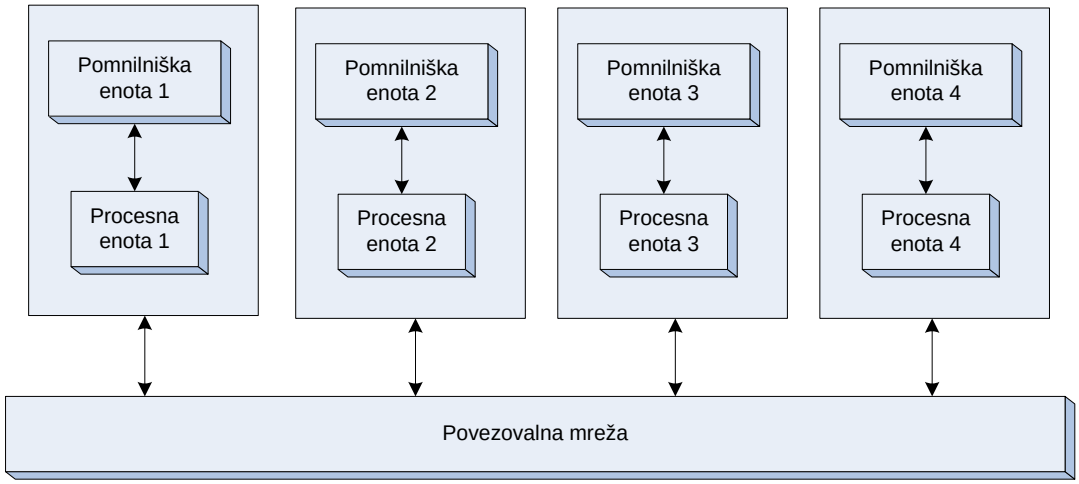
Procesne enote v sistemih, ki so zasnovani na principu pošiljanja sporočil [41], medsebojno komunicirajo tako, da si pošiljajo sporočila (izvorno: *message passing*). Vsaka procesna enota je neposredno povezana s svojo lokalno pomnilniško enoto (slika 2.6). Sistem nima skupnega pomnilnika, vsaka izmed procesnih enot pa ima svoj naslovni prostor. Procesne enote začasno shranjujejo sporočila v medpomnilne enote (izvorno: *buffers*), dokler jih ne posredujejo drugi procesni enoti ali jih sprejmejo od drugih procesnih enot. Pošiljanje in sprejemanje sta povsem ločeni aktivnosti in nista odvisni od procesiranja podatkov. Povezovalna mreža, ki zagotavlja povezljivost procesnih enot, je običajno fizično omejena na en prostor. Uporabljajo se statične in dinamične povezovalne mreže. Od statičnih so najpogostejše topologija niza procesnih enot, obroča, matrična in drevesna topologija ter topologija hiperkocke v različnih dimenzijah. Pri dinamičnih povezovalnih mrežah pa se uporabljajo stikala (eno- ali večstopenjske stikalne povezave) ali pa prečniki. Pristop s povezovalno mrežo omogoča bistveno lažjo razširljivost sistema, saj s povečevanjem števila procesnih enot ne vplivamo občutno na zmogljivost sistema. Računalniški sistemi MIMD imajo običajno manj procesnih enot, kot jih premorejo sistemi SIMD. So kompleksnejši, a ponujajo največjo učinkovitost vzporednega procesiranja.

Najzmogljivejši vzporedni računalniški sistemi so danes skorajda izključno zasnovani kot sistemi MIMD. Vsaka izmed procesnih enot je lahko tudi večjedrna. V grobem delimo dve vrsti [18]:

1. tesno povezani sistemi (sistemi SMP) − tu so vse procesne enote enake (zato simetrični) in imajo na voljo skupne sistemske vire (vmesniki V/I, pomnilnik). Tipični primer so rezine (IBM, HP, Dell), ki imajo vgrajenih več procesnih enot, vsaka izmed njih pa je večjedrna. Procesne enote krmili en sam operacijski sistem. Asimetrične sisteme [24] pa v nasprotju s simetričnimi sestavlja niz heterogenih procesnih enot. Ena izmed enot na primer skrbi za izvajanje operacijskega sistema, druga za prenos podatkov iz/v naprave V/I, spet tretja za obdelavo signalov in podobno. Povečanje

zmogljivosti sistemov SMP lahko dosežemo le s povečanjem števila procesnih enot sistema;

2. ohlapno povezani sistemi (grozdi in sistemi MPP) − te sisteme sestavljajo medsebojno povezani in neodvisni računalniški sistemi, ki imajo na voljo ločene vire (pomnilnik, vmesnike V/I) in svoj operacijski sistem. Vsak izmed sistemov je lahko sistem SMP. V tem primeru lahko povečanje zmogljivosti dosežemo s povečanjem števila sistemov ali pa s povečanjem števila procesnih enot v sistemih znotraj sistemov SMP. Grozd [18, 51] je v osnovi majhno število med seboj povezanih in običajno ekvivalentnih računalniških sistemov (rezine IBM in HP). Vsak sistem ima svojo kopijo operacijskega sistema, svojo procesno enoto, svoj pomnilnik in lahko deluje povsem avtonomno. Sistemom pravimo tudi vozlišča grozdne postavitve. Običajno so povezana z omrežno tehnologijo LAN. Postavitev je po navadi omejena na en sam fizični prostor. Navzven delujejo kot monoliten sistem (izvorno: *single system image*). Implementacija postavitve grozda je za uporabnika povsem nepomembna.



Slika 2.6: Koncept arhitekture MIMD

Grozdi omogočajo naslednje:

1. visoko zmogljivost (izvorno: *HPC cluster*) − množica sistemov za reševanje vzporednih algoritmov. Je ena od možnosti, ki lahko zamenja dražje monolitne rešitve s primerljivimi zmogljivostmi. Najobičajnejša in hkrati prva postavitev grozda je bil tako imenovan sistem Beowulf [30, 63]. Gre za grozd, ki je namenjen doseganju visokih zmogljivosti. Poleg grozda Beowulf so pretežno razširjeni tudi Rocks [57, 63], MOSIX [63, 71] in PelicanHPC [72];
2. visoko razpoložljivost (izvorno: *HA cluster*) − v primeru okvar ali skrbniških posegov se aktivirajo dodatni razpoložljivi računalniški sistemi;
3. izenačevanje obremenitve (izvorno: *LB cluster*) − v primeru večjih obremenitev omogočajo aktiviranje dodatnih razpoložljivih računalniških sistemov, ki pri običajnih obremenitvah niso aktivni;
4. podvajanje sistema (izvorno: *redundancy*) − funkcijo enega vozlišča grozda lahko v primeru okvare ali nadgradnje nadomesti drugo vozlišče, še posebno, če so vozlišča med seboj ekvivalentna.

Sistem MPP [18] je množica avtonomnih vozlišč, ki je bistveno obsežnejša in zmogljivejša od množice vozlišč v grozdu. Sega lahko od tisoč do nekaj sto tisoč vozlišč. Današnji

superračunalniki so zasnovani skoraj izključno z uporabo komercialno dostopnih serijskih procesnih enot. To je bistvena prednost v primerjavi s klasičnimi pristopi gradnje superračunalnikov, kjer se uporabljajo posebne in cenovno neugodne tehnologije GaAs. Običajno se poslužujejo nestandardne, visokozmogljive omrežne tehnologije Infiniband. Omogoča občutno manjše zakasnitve (tipično 12 μs) in bistveno večjo pasovno širino (300 Gbit/s), kot jih omogočajo klasične omrežne tehnologije LAN ali FC. Primer sistema MPP je IBM RoadRunner [9]. Zaradi heterogenih procesnih enot (AMD Opteron in IBM Cell) se obravnava kot hibridni sistem, v katerem so strežniške rezine povezane z omrežno tehnologijo Infiniband.

2.2 Porazdeljeni sistemi

Potreba po porazdeljenem računanju oziroma aplikacijah, ki potrebujejo porazdeljeno računanje, je začela rasti že v 80-ih letih prejšnjega stoletja. Zanimivi so posebno zaradi splošnega napredka v tehnologiji in občutnega padca cen strojne opreme. Kmalu so postali cenovno ugodna, visokozmogljiva in zanesljiva tehnologija pri reševanju problematike na več področjih. Že pred prehodom v novo tisočletje pa je prišlo tudi do velikega razmaha uporabnosti omrežja internet, ki je omogočalo dostop do najrazličnejših virov. Razmah spleta in nagel padec cen pasovne širine povezav sta pripomogla k priljubljenosti porazdeljenih sistemov. Razširjeni niso le na akademskem in raziskovalnem področju, ampak tudi v vladnih organizacijah in celo pri zasebnikih. Vzroke za uvajanje in uporabo teh sistemov je iskati predvsem v željah uporabnikov po souporabi virov. Čeprav je "vir" abstrakten pojem, lahko v najboljši meri karakterizira različne vrste računalniških sistemov oziroma naprav v omrežju. Označuje lahko procesno enoto, pomnilniško enoto, diskovno enoto, tiskalnike, programske entitete, podatkovne baze, avdio-/videoizvore ali pa kar namizni računalnik v splošnem. Tudi naše vsakdanje življenje spremlja vse več strojne opreme, ki je del porazdeljenega sistema (vgrajeni sistemi, tipala, naprave GPS in podobno). Porazdeljeni sistemi postajajo vse pomembnejši, še posebno sistemi širših razsežnosti [48].

Obstaja več definicij porazdeljenega sistema, vendar omenimo najbolj splošno [62]: porazdeljen sistem je množica računalniških virov (običajno računalnikov), ki si ne delijo skupnega pomnilniškega naslovnega prostora in skupne sistemske ure. Sodelujejo izključno s pomočjo pošiljanja sporočil preko komunikacijskega omrežja, ki je lahko prostrano ali lokalno (WAN, LAN). Vsaka procesna enota računalniškega vira ima svoj pomnilnik in svoj operacijski sistem. Enote so v celoti avtonomne in rahlo sklopljene. Z medsebojno komunikacijo omogočajo reševanje časovno in prostorsko zahtevnih problemov. Sistem se lahko z dodajanjem procesnih enot preprosto širi.

Porazdeljeni sistemi so ohlapno povezani sistemi. Njihovi računalniški viri so lahko geografsko ločeni, ni pa nujno. V tem primeru so komunikacijski kanali manj zanesljivi. Pogostokrat so viri heterogeni, kar pomeni, da imajo njihove procesne enote povsem drugačno arhitekturno zasnovo. V delovnem stanju porazdeljenega sistema je zaradi številnih razlogov topologija dinamična in se s časom lahko spreminja. Na primer pri napakah na komunikacijskih kanalih, dodajanju/odstranjevanju procesnih enot in pri skrbniških popravilih. Posamezni računalniški vir ima običajno na voljo svoj operacijski sistem in morebitno vmesno programsko opremo (izvorno: *middleware*). Vmesna programska oprema je pravzaprav množica programskih knjižnic. Uporabljajo se v programski kodi, ki omogoča vključevanje oziroma sodelovanje računalniškega vira v porazdeljenem sistemu. Tipična primera sta MPICH2 [69] in Open MPI [44, 70], ki sta implementaciji komunikacijskega

protokola MPI [29, 49, 63]. Protokol MPI je pravzaprav generična specifikacija vmesnikov za pošiljanje sporočil v porazdeljenih sistemih. Je skoraj v celoti nadomestila orodje PVM [27] in tako dejansko postala standard v industriji. Vmesnik MPI se med drugim uporablja tudi na superračunalniškem sistemu na inštitutu za turbinske stroje v Ljubljani. Razvrstitev porazdeljenih sistemov se v veliki meri opira na referenčno Flynnovo klasifikacijo. Porazdeljenih sistemov je več vrst: sistemi GRID, sistemi v oblakih, prostovoljni sistemi in sistemi P2P (izvorno: *peer-to-peer*). Vsi omenjeni sistemi spadajo v Flynnov razred sistemov MIMD.

2.2.1 Sistemi GRID

Beseda GRID je relativno nova beseda. Različni ljudje jo različno razumejo in sprejemajo. Marsikdaj je celo zlorabljena in poveličevana, češ da rešuje probleme prav vseh vrst. Tipičen primer je podjetje Oracle, ki naj bi zaradi takšnih pristopov povečalo ekonomsko penetracijo. Zahteve po procesni moči so in še vedno naraščajo. Novi projekti v gospodarstvu in znanosti potrebujejo ogromne količine procesne moči. Sčasoma je nastala potreba po večjem oziroma globalnem porazdeljenem sistemu, ki sega tudi preko meja ene organizacije. Govorimo o sistemih GRID [26, 50, 52], pri katerih gre za transparentno dodeljevanje oziroma souporabo računalniških virov v prostranem okolju. Izkorišča se tudi vire, ki bi bili sicer neizkoriščeni. Viri so lahko povsem heterogeni (različna strojna in programska oprema), so zanesljivi ali nezanesljivi in se lahko nahajajo na geografsko ločenih lokacijah. Dostop do virov lahko primerjamo s principom, ki je podoben dostopu do električne energije v naših domovih, kjer nam ni treba poznati izvora in načina dobave energije. Sistem GRID je praviloma množica računalniških virov, ki pripadajo več ločenim skrbniškim okoljem, imenovanim tudi navidezne organizacije. Čeprav opis deluje preprosto, so operacije, ki potekajo v ozadju, vse prej kot to.

Osnovne funkcije, ki jih mora zagotavljati sistem GRID, so naslednje:

1. funkcija overitve (vsak uporabnik sistema GRID je identificiran);
2. funkcija odobritve (odobritev dostopa za določeno množico virov v sistemu GRID);
3. omogočati dostopnost virov (oddaljeni viri sistema GRID morajo biti dostopni povsem transparentno);
4. omogočati funkcijo iskanja virov oziroma razporejanja opravil (iskanje vira za uporabnika mora biti v sistemu GRID samodejno).

Obstaja kar nekaj vrst vmesne programske opreme, ki tvori sistem GRID. Je bodisi odprtokodna ali komercialna. Najbolj razširjena odprtokodna je Globus Toolkit [80], ki vsebuje več vrst orodij (varnostne mehanizme, upravljanje z viri, komunikacijo). Večina odprtokodnih sistemov GRID temelji na orodjih, ki so del sistema Globus Toolkit. Značilna sta sistem ARC [78] in sistem gLite [73] v sklopu projekta EGEE, ki sta nadgradnji sistema Globus. Poznani pa so tudi podatkovni sistemi GRID, ki so namenjeni predvsem upravljanju in delitvi velike količine razpršenih podatkov. Tipičen primer je Oracle Database 11 g. Med sistemi GRID in grozdi je morebiti zaznati podobnost, a obstaja kar nekaj pomembnih razlik. Čeprav so tako grozdi kot sistemi GRID rahlo povezani sistemi, se pri grozdih v splošnem omejujemo na eno fizično lokacijo. Običajno so računalniški sistemi povezani z uporabo omrežne tehnologije LAN. Sistemi GRID pa so namenjeni predvsem prostranim omrežjem in imajo globalni učinek, ni pa nujno. Računalniški viri so heterogeni, avtonomni in izvajajo opravila, ki so v večji meri neodvisna. So veliko prožnejši in razširljivi. V grozdih pa bolj ali manj težimo k uporabi unificirane ali pa vsaj enakovredno zmogljive strojne opreme. Običajne postavitve obsegajo nekaj 10 računalniških sistemov, ki niso vedno avtonomni.

Obstaja predsodek, da so sistemi GRID uporabni le v okoljih z nezanesljivimi in relativno počasnimi omrežnimi tehnologijami, vendar jih lahko brez ovir namestimo tudi v lokalnih okoljih. V tem primeru so povsem primerljiva ali celo boljša postavitev od klasičnih postavitev grozdov.

Na tem mestu moramo omeniti tudi vse bolj razširjeno tehnologijo virtualizacije [25, 58], ki bo imela v prihodnjem razvoju sistemov GRID pomemben vpliv. Je dodatna plast programske opreme, ki abstrahira večino podrobnosti sistema (strojne in programske opreme) in zamenja sistemske vmesnike z navideznimi. Abstrakcijo lahko obravnavamo kot navidezni stroj nad gostujočim operacijskim sistemom oziroma nadzornikom navideznega stroja (izvorno: *hypervisor*). Navidezni stroji se v odvisnosti od razpoložljivih virov sistema lahko množijo in tako omogočijo večjo učinkovitost uporabe sistemskih računalniških virov.

Čeprav uvedba virtualizacijske tehnologije privede do manjšega padca zmogljivosti računalniškega vira, lahko bistveno učinkoviteje izkoriščamo njegove podsklope (procesor, pomnilnik, trdi disk). *Xen*, *VMware*, *Virtuozzo*, *VirtualBox* in podobne platforme podpirajo različne vrste operacijskih sistemov, ki se izvajajo na navideznih strojih. Virtualizacija pa ni edini način ločevanja uporabnikov in ponudnikov virov v sistemu GRID. Možne so tudi preprostejše rešitve, kot na primer ukaz za spremembo korenskega imenika *(*izvorno: *chroot)* v sistemih Unix ter *Jail* v sistemih FreeBSD [58]. Varnost pri teh pristopih je bistveno nižja od virtualizacije. Poleg tega so ločeni le deli, ki se nanašajo neposredno na podatkovne sisteme, ne pa celoten računalniški sistem. Podoben, a veliko bolj omejen pristop dosežemo tudi z uporabo vmesne programske opreme JVM. Je bistveno bolj omejena glede na različne vrste programske opreme, ki jo podpira. Virtualizacija je, kljub številnim rešitvam, zaradi svoje fleksibilnosti in drugih koristi primernejša, preprostejša in veliko bolj transparentna metoda abstrakcije računalniških virov v sistemih GRID.

Prednosti z uporabo virtualizacije so naslednje:

1. varnost in izolacija sta večji v okoljih z navideznimi stroji kot v klasičnih večprogramskih sistemih. Uporabniki, ki bi želeli sistemu škodovati, bi v principu morali biti sposobni prodreti izven svojega navideznega stroja, mimo varnostnih mehanizmov gostujočega sistema, nato mimo varnostnega sistema drugega navideznega stroja in dodatno skušati prodreti v sam navidezni stroj. To pa vsekakor ni trivialen postopek;
2. opravljanje skrbniških dejavnosti je hitrejše in preprostejše. Aktivnosti ne potekajo več na nivoju procesov, ampak na nivoju navideznih strojev;
3. podpora podedovane (izvorno: *legacy*) programske opreme ni več kritična;
4. zagotovljena je neodvisnost navideznega stroja od strojne opreme, na kateri se izvaja. Prenosi navideznih strojev iz ene strojne platforme na drugo so preprostejši.

Čeprav je sistem GRID s številnimi projekti deležen precejšnjega uspeha, je dostop do infrastrukture oziroma računalniških virov težavnejši. Razloge je iskati med drugim tudi v birokratskih ovirah. Sleherna raziskovalna organizacija lahko do infrastrukture dostopa le preko posebne prošnje. Ta med drugim zahteva tudi popolno dokumentacijo raziskovalne dejavnosti in projekta, ki bi izkoriščal računalniške vire v sistemu GRID. Med raziskovalnimi organizacijami lahko prihaja do ostre tekmovalnosti. Še pomembnejše pa so tehnične omejitve. Orodja in programski vmesniki aplikacij, ki uporabljajo računalniške vire sistema GRID, so specifični in vnaprej določeni. Na težave naletimo lahko že pri napačno izbranem operacijskem sistemu.

Eden izmed večjih problemov sistemov GRID pa, presenetljivo, ni tehnične narave. Čeprav tehnologija rešuje večino tegob, še vedno ostaja družben problem. Da sistem GRID sploh lahko obstaja, morajo biti vsi vključeni pripravljeni sodelovati pri souporabi računalniških virov. To pa je v nasprotju z naravo dela večine vključenih. Gotovo večji delež sodelujočih v sistemu z zadržki in previdno podaja računalniške vire v souporabo. Vsaj ne brez neposredne koristi. Težko je verjeti, da se bo v prihodnjih letih razvil globalni sistem GRID brez aktivnega sodelovanja udeležencev pri souporabi virov. Verjetno pa lahko pričakujemo, da se bodo sistemi GRID še nekaj časa gradili ločeno po individualnih organizacijah, med seboj pa se bodo povezovali po regulativnih postopkih.

2.2.2 Sistemi v oblaku

Če je bila razširjenost sistemov GRID na vrhuncu v začetku minulega desetletja, imajo danes daleč več pozornosti sistemi v oblakih [7, 31] (izvorno: c*loud computing*). Ideja izvira že iz leta 1961, ko je profesor John McCarthy [81] predlagal koncept storitvenega računalništva, a je bila takratna tehnologija preprosto premalo zmogljiva za takšne koncepte. Oživitev ideje se je zopet pojavila, a z malce drugačnim imenom: oblak. Obstaja mnogo definicij takšne vrste sistemov in nastala je precejšnja zmeda. Veliko jih izhaja predvsem iz tehnologije. Pogosto se storitveno računalništvo in oblaki preprosto enačijo. Predvsem dobavitelji definirajo računalništvo v oblaku kot storitveno računalništvo, ki temelji na tehnologiji virtualizacije. Lahko bi trdili, da je računanje v oblaku uporabniku prijazna zvrst sistema GRID. Zanimiva je predvsem definicija Buyya [17]: Sistem v oblaku je lahko vrsta distribuiranega ali vzporednega sistema, kot sestav virtualiziranih medsebojno povezanih računalnikov. Z njimi upravljamo na dinamičen in vnaprej določen način.

Računanje v oblakih omogoča uporabnikom dostop do računalniških virov ali storitev preko omrežja. Podobno kot pri sistemih GRID tudi tu najdemo primerjavo v postopku dobave električne energije po domovih. Uporabnikom je dostop do takšnih storitev povsem preprost in jim ni treba poznati vseh procesov, ki omogočajo dobavo energije. Uporabnik lahko do virov ali storitev dostopa preko prostranega omrežja internet kjer koli, kadar koli, s katero koli vrsto končne naprave. Lokacija virov ali storitev je za uporabnika povsem nepomembna. Nahajajo se nekje v "oblaku".

Podjetje Amazon je imelo odločilno vlogo pri oblikovanju sistemov v oblaku. Ko so v podjetju leta 2002 s preprostimi programskimi vmesniki (spletnimi storitvami) omogočili končnim uporabnikom storitveno računalništvo, se je v svetu informatike pričela manjša revolucija. Kmalu zatem se je pojavil nov način uporabe računalniških virov: pristop "na zahtevo". Omogoča, da se aplikacija in uporabniški podatki nahajajo na oddaljenih strežnikih oziroma oddaljenih sistemih za shranjevanje podatkov. Do njih se dostopa s katero koli vrsto končne naprave (osebni ali prenosni računalnik, pametni telefoni). Imenujemo ga tudi "programska oprema kot storitev" [54] (izvorno: *SaaS*). Gre za vrsto računalništva v oblaku, ki uporabnikom omogoča dostop do aplikacij kar preko brskalnika [31]. Ni pa nujno. Tipična podjetja, ki omogočajo te vrste storitev, sta predvsem Salesforce in Google. Malce drugačen vzorec oblike SaaS je "platforma kot storitev" [54] (izvorno: *PaaS*). Uporabnikom omogoča popolno računalniško okolje kot sestav računalniške arhitekture, operacijskega sistema in razvojnega okolja z različnimi programskimi jeziki in programskimi knjižnicami. V takšnih primerih se velikokrat uporablja nek splošen model, s pomočjo katerega uporabniki gradijo aplikacije, do njih pa običajno dostopajo preko brskalnika. Dobra stran takšnega pristopa je predvsem v svobodi razvijanja programske opreme, ki je sicer ni na voljo pri ponudniku. Slaba lastnost pa je, da so razvoj, zmogljivosti in integracija programske opreme omejeni na

okolje ponudnika (na primer Google App Engine). Tudi "infrastruktura kot storitev" [54] (izvorno: *IaaS*) je ena izmed oblik sistemov v oblaku. Uporabniku omogoča dostop do popolne infrastrukture: strežnikov, sistemske in uporabniške programske opreme, podatkovne baze, omrežne opreme (stikala, usmerjevalniki) in tudi fizičnega prostora, kjer se omenjena oprema nahaja. Za orkestracijo navedenih virov skrbi posebna programska oprema, imenovana tudi upravljalnik navideznih virov [16] (izvorno: *virtual infrastructure manager*), ki združuje vire različnih računalniških sistemov. Uporabniku in različnim aplikacijam jih predstavi kot enovito celoto.

Sistemi v oblaku so bistveno prožnejši od sistemov GRID [19]. Uporabniku so na voljo najrazličnejše vrste programske opreme, medtem ko je v sistemih GRID treba vzdrževati posebne vmesnike, ki povezujejo različne tehnologije sistemov GRID. Sisteme v oblaku odlikujejo predvsem zanesljivost, razširljivost v smislu dodatnih računalniških virov in storitev, lažje vzdrževanje, podpora uporabnikom ter enovita informacijska varnost. Oba sistema pa sicer zagotavljata podporo heterogenosti, razširljivosti in uporabi tehnologije virtualizacije.

Zagotavljanje kakovosti storitev (izvorno: *QoS*) v sistemih GRID običajno poteka kar v sklopu aplikacije, ki se izvaja nad vmesno programsko opremo. Tipični ponudniki storitev (Amazon), ki uporabljajo sisteme v oblaku, pa že od samega začetka ponujajo osnovni sporazum o zagotavljanju ravni storitve (SLA).

V splošnem imajo sistemi v oblaku in sistemi GRID enake cilje: zmanjševati stroške pri uporabi virov, povečati zanesljivost in prožnost. Lahko jih obravnavamo kot komplementarni tehnologiji. Povsem brez omejitev lahko sisteme v oblaku uporabimo kot čelni del sistema (izvorno: *front-end*), do katerega dostopa uporabnik neposredno (tu sistem GRID ni najprimernejši). Velja pa tudi obratno, sisteme GRID lahko brez omejitev zasnujemo nad računalniškimi viri, ki so del sistema v oblaku. Težko je verjeti, da bo kateri od obeh sistemov prevladal, saj gre pri vsakem za različno namembnost. Zagotovo pa lahko pričakujemo njuno konvergenco in vse pogostejšo uporabo virtualizacije.

2.2.3 Prostovoljni sistemi

Prostovoljni sistemi oziroma prostovoljno računalništvo [5] so oblika porazdeljenega sistema, imenovana tudi globalno računalništvo [12] ali tudi računalništvo z javnimi računalniškimi viri (izvorno: *public resource computing*). V tem sistemu sodelujoči (civilna družba) prostovoljno podajajo v souporabo svoje računalniške vire v omrežju (običajno prostranem omrežju internet). Vir obravnavamo kot povsem poljubno vrsto strojne opreme. To so lahko namizni osebni računalnik, zmogljivi grozdi ali zmogljivi strojni grafični vmesniki – grafične kartice, ki so za določene aplikacije izjemno uporabne [83]. Večina sodelujočih so uporabniki, ki imajo na voljo namizni osebni računalnik z operacijskimi sistemi Windows, Linux ali OS X. V prostrano omrežje so običajno povezani s povezavami DSL, kabelskimi, brezžičnimi ali tudi optičnimi povezavami. Vrste in zmogljivosti virov se med seboj lahko zelo razlikujejo, zato imajo takšni sistemi najvišjo možno stopnjo heterogenosti.

Povezljivost vseh sodelujočih v sistemu je dinamična in jih zato ne smemo ali jih ne moremo obravnavati kot zanesljive. Viri se velikokrat nahajajo tudi za požarnim zidom ali pa uporabljajo funkcionalnost NAT. To pomeni, da nimajo globalnih internetnih naslovov IP. Različica IPv6 bo imela zato v prihodnosti ključno vlogo. Viri zaradi omenjenih težav ne sprejemajo povezav, ampak jih običajno prožijo sami. Večina prostovoljnih sistemov je

zasnovana s pomočjo centralne enote (običajno strežniški sistem), na katero se povežejo vsi prosti računalniški viri sistema (na primer osebni namizni računalniki). Naloga centralne enote je, da s pomočjo prostih računalniških virov reši časovno in prostorsko zahteven problem – posel oziroma opravilo. Prostim računalniškim virom pošilja posle po izbranem postopku razvrščanja. Računalniški viri posle izvedejo in rezultate pošljejo nazaj na centralno enoto. Z zbranimi rezultati centralna enota tvori končno rešitev problema.

Če se v splošnem opravilo izvaja na nekem prostem računalniškem viru, nikakor ne moremo z gotovostjo trditi, da je rezultat izvajanja opravila pravilen oziroma verodostojen. Poleg tega centralni točki ni dovoljeno opravljati kakršnih koli sprememb na prostem viru ali povzročiti celo okvaro na njem. Dostop do zaupnih informacij o sodelujočem ali do kritičnih podatkov, ki se nahajajo na prostem viru (gesla, številke kreditnih kartic), je namreč povsem sporen. Zaradi omenjenih težav se na prostih virih običajno uporabljajo peskovniki (izvorno: *sandboxing*), ki omogočajo popolno izolacijo izvajanja opravil od ostalega okolja prostega vira. Peskovnik je pravzaprav programska oprema in kot posebna plast med operacijskim sistemom in aplikacijo prestreza sistemske klice. Vse pogostejša pa je tudi uporaba virtualizacije, kjer lahko izolacijo dosežemo na nivoju operacijskega sistema. Izvajanje opravila poteka na enem operacijskem sistemu, ostale aktivnosti okolja (aplikacije) pa v sklopu drugega operacijskega sistema. Varnost je v tem primeru bistveno večja.

Prostovoljni sistem je primeren predvsem za reševanje podatkovno vzporednih problemov, pri katerih njihova segmentacija na več manjših delov ne predstavlja večjih ovir. Deli problema se rešujejo povsem neodvisno (in ne nujno hkrati), tudi v daljšem obdobju (meseci, leta). Prepustnost takšnih okolij je lahko izjemno visoka. Problemi, ki potrebujejo zelo hiter odziv in kjer so rezultati pričakovani v najkrajšem možnem času, običajno niso primerni za reševanje na prostovoljnih sistemih.

Prostovoljni sistemi ne jamčijo nikakršnega sporazuma o ravni storitve. V te namene bi bilo treba spremeniti prostovoljni sistem v hibridni sistem, ki bi bil sestav dveh vrst virov: namenskih računalniških virov in prostih računalniških virov. Namenski računalniški viri bi morali biti zanesljivi in vedno v pripravljenosti. Prosti računalniški viri pa so lahko nezanesljivi, njihova prisotnost v prostovoljnem sistemu pa povsem sporadična. Sistem BOINC [3, 6, 74] je ena izmed platform, ki je izjemno razširjena in priljubljena na področju prostovoljnih sistemov. Uporablja se pri najrazličnejših projektih znanstvenega računanja. Trenutno je aktivnih 50 različnih projektov, vključenih pa že več kot 600000 prostih računalniških virov. Platforma s podobnimi cilji je tudi sistem Condor [66, 75], ki je sicer namenjen delovanju znotraj zaprtega okolja, a povezava z zunanjimi sistemi ni izključena [67].

Prostovoljni porazdeljeni sistemi lahko zagotovijo veliko večje število prostih virov, kot ga zagotavlja kateri koli superračunalniški sistem. Omogoča reševanje problemov, ki bi bili sicer zaradi pomanjkanja računalniških virov težje rešljivi. Skupaj zbrana procesna in pomnilniška zmogljivost pa je povsem brezplačna.

2.2.4 Sistemi "vsak z vsakim" (P2P)

Sistemi P2P [55, 59] ali sistemi "vsak z vsakim" so v minulem desetletju pritegnili veliko pozornosti. K temu sta pripomogla predvsem sistem za shranjevanje datotek Freenet [84] in že ugasli sistem Napster. Sistem Napster je omogočal prenose zvočnih datotek med uporabniki. Takšna okolja, kjer prenosi podatkov potekajo neposredno med posamezniki, obravnavamo kot porazdeljene sisteme z visoko stopnjo decentralizacije. Sodelujoči

računalniški sistemi (pravimo jim tudi vozlišča) imajo v okolju dvojno vlogo: delujejo kot strežnik in hkrati kot odjemalec. Stanje sistema določajo kar vozlišča sama in to pomeni, da izvajajo glavnino opravil. Namenskih centralnih vozlišč je malo, skrbniških posegov pa skoraj ni treba opravljati, saj so posamezna vozlišča skoraj povsem avtonomna. Lastniki vozlišč se prostovoljno vključujejo in izključujejo iz sistema P2P. Zato raste samostojno in naravno. Veliko bolj je zaščiten proti napadom, saj bi bilo za ustavitev celotnega sistema P2P treba napasti več vozlišč hkrati, kar je izjemno zahtevno ali celo nemogoče.

Uporabnost sistemov P2P je raznolika. Najbolj razširjeni so na področju souporabe datotek (izvorno: *file sharing*). Zasledimo jih tudi na področju pretočnih medijev (izvorno: *streaming media*) oziroma na področju sistemov IPTV. Tu se skuša izkoriščati predvsem obstoječo omrežno prepustnost uporabnikov namesto visoke omrežne prepustnosti centralnega strežnika. Tipičen primer uporabe sistemov P2P je na področju telefonije VoIP aplikacija Skype [76]. Uporabnikom omogoča avdio-vizualno povezljivost, neodvisno od lokacije in vrste povezave v prostrano omrežje. Ker si poleg omenjenih storitev uporabniki lahko izmenjujejo tudi računalniške zmogljivosti (na primer SETI@home), lahko takšne sisteme obravnavamo kot posebno vrsto sistemov P2P. Tako kot se uporabnik sistema P2P prostovoljno odloči za souporabo datotek, se odloči tudi za souporabo računalniških zmogljivosti. Odloči se o tem, ali želi ostalim udeležencem sistema P2P ponujati souporabo procesne enote, prostora na trdem disku, pomnilnika, omrežne prepustnosti in ostalih računalniških virov.

Arhitekturno lahko sisteme P2P razvrstimo glede na stopnjo njihove centraliziranosti:

1. **delno centralizirani sistemi P2P** imajo običajno na voljo centralno vozlišče, ki spremlja vsa ostala sodelujoča vozlišča in upravlja s celotnim sistemom. Sistem Napster je za souporabo datotek temeljil na strežniku WEB, ki je hranil podatke o uporabnikih in indeksiral njihove vsebine. Podobno velja za sisteme, ki uporabljajo protokol BitTorrent [53], in sisteme, ki temeljijo na platformi BOINC (o sistemu BOINC več v poglavju 3.1). Sistem Skype pa ima malce več centraliziranosti: omogoča prijavo uporabnikov v sistem, upravljanje z uporabniškimi imeni in plačilni promet. Prožnost teh sistemov je relativno omejena, saj centralni del, poleg kritične točke odpovedi, predstavlja ozko grlo sistema;
2. **decentralizirani sistemi P2P** so brez centralnega vozlišča in zato brez ozkega grla. Veliko bolj so zaščiteni pred vdori ali ohromitvijo storitev. Še vedno lahko obstajajo vozlišča, ki prevzemajo pomembnejše vloge, na primer shranjevanje stanja sistema in opravljanje indeksacije vsebin, a njihova prisotnost ni nujna. Lahko pa bistveno poveča učinkovitost delovanja.

Za sisteme P2P je značilno tako imenovano prekrivno omrežje [55] (izvorno: *overlay network*), ki je definirano kot usmerjen graf z množico vozlišč (v principu računalnikov) in množico povezav med njimi. V delno centraliziranih sistemih P2P se nova vozlišča v prekrivno omrežje pridružujejo tako, da se povežejo na centralno vozlišče. To pomeni, da povezave prekrivnega omrežja najprej tvorijo zvezdasto omrežno topologijo (s centralnim vozliščem v središču), kasneje pa se lahko tvorijo dodatne povezave med vozlišči. V decentraliziranih sistemih se nova vozlišča povežejo z vozliščem, ki je že del prekrivnega omrežja. Njegov naslov IP se običajno nahaja na eni od spletnih strani. Prekrivna omrežja so uporabna predvsem za učinkovito komunikacijo oziroma usmerjanje sporočil med vozlišči. Usmerjanje temelji na uporabi ključev, ki omogočajo zanesljivo in hitro lokacijo enolično označenih objektov. Objekt je lahko na primer datoteka ali del datoteke ali samo kazalec na

datoteko. Lociranje objekta običajno poteka s preplavljanjem vozlišč prekrivnega omrežja ali pa se poizvedba pošlje po naključni poti prekrivnega omrežja [55]. Izkazalo se je, da so sistemi P2P še vedno primernejši za souporabo datotek in manj za izkoriščanje računalniških virov (procesnih in pomnilniških zmogljivosti). Vzrok je predvsem oteženo zagotavljanje ravni storitve. Vse pogostejša pa so okolja, ki uporabljajo sistem P2P kot pomožno tehnologijo. Primer tega je hibridni sistem, ki ga sestavljata sistema XtremWEB in Condor [43].

2.3 Primerjava vzporednih in porazdeljenih sistemov

Čeprav so vzporedni in porazdeljeni sistemi dve ločeni veji, imajo nekaj skupnih lastnosti:

1. pri obeh pristopih se uporablja več procesnih enot, kar pomeni prisotnost vzporednega reševanja problemov. Izvaja se več vzporednih aktivnost, ne nujno hkrati, med seboj pa lahko tudi sodelujejo;
2. povezovanje procesnih enot je običajno rešeno s pomočjo uporabe omrežne tehnologije;
3. večji problem se razstavi na več manjših podproblemov, ki se rešujejo na ločenih računalniških virih.

In čeprav se ponekod izraza uporabljata izmenoma (tudi nehote), lahko razlike med obema opišemo.

Vzporedni sistemi poudarjajo naslednje:

1. problem je razdeljen na podprobleme. Rešujejo se hkrati, najpogosteje v tesni povezanosti, za katero velja, da so procesne enote zelo blizu skupaj oziroma v sklopu enega računalniškega sistema (na primer superračunalniški sistem). Povezanost je lahko tudi ohlapna. V tem primeru govorimo o grozdih in sistemih MPP. Vzporedni sistemi so drobnozrnati sistemi (izvorno: *fine-grained system*), pri katerih je čas izvajanja neposredno na procesni enoti veliko krajši od časa za komunikacijo med procesnimi enotami;
2. v vzporednem sistemu se izvaja samo en program (skupek več procesov) naenkrat, cilj pa je pohitritev tega programa;
3. vzporedni sistem je običajno zasnovan homogeno. Sestavlja ga večje število enakih procesnih enot;
4. procesi med seboj sodelujejo preko skupnega pomnilniškega prostora ali s pošiljanjem sporočil.

Porazdeljeni sistemi pa poudarjajo:

1. problem je razdeljen na podprobleme, ki se rešujejo neodvisno in ne nujno hkrati. Procesne enote so med seboj fizično oziroma geografsko ločene (ohlapna povezanost). Porazdeljeni sistemi so grobozrnati sistemi (izvorno: *coarse-grained system*), pri katerih je čas izvajanja neposredno na procesni enoti veliko daljši od časa, potrebnega za komunikacijo med procesnimi enotami;
2. v porazdeljenem sistemu se izvaja več različnih programov (kot en proces ali kot množica večih procesov), ki pripadajo več različnim uporabnikom. Izvajanje programov je lahko hkratno ali pa tudi ne. Cilj je izvesti čim več programov oziroma doseči čim večjo prepustnost;

3. porazdeljeni sistemi so zasnovani heterogeno − sestavljeni so iz večjega števila procesnih enot različnih arhitektur. Sistem je dinamičen in se s časom spreminja;
4. procesi med seboj običajno sodelujejo le s pošiljanjem sporočil. Redka je izvedba, pri kateri se komunikacija med procesi reši s pomočjo skupnega pomnilnika.

Porazdeljeni sistemi imajo kar nekaj prednosti v primerjavi z vzporednimi sistemi. Razširljivost v smislu dodatnih procesnih enot je preprostejša. So prožni in prilagodljivi naravi problema. Disipacija in potreba po večjem hlajenju na procesnih enotah sta bolj kritični pri vzporednih in manj pri porazdeljenih sistemih. Po drugi strani pa se srečujemo tudi s problematiko pri porazdeljenih sistemih − so namreč težje obvladljivi v smislu upravljanja in programiranja. So manj predvidljivi, težje jih je preverjati, odkrivanje napak in odpovedi pa je težavnejše.

Na področju vzporednih in porazdeljenih sistemov poteka trenutno veliko razvojnih in raziskovalnih aktivnosti. Znanstveniki v raziskovalnih organizacijah, pa tudi uporabniki v splošnem, so od nekdaj stremeli k reševanju kompleksnejših problemov. Šele zadnjih nekaj let smo lahko priča izjemnemu tehnološkemu napredku, ki to omogoča. In obratno, obstoj zmogljivih vzporednih računalnikov nudi obravnavanje problemov, ki v preteklosti niso bili mogoči ali zanimivi. Razvoj novih vzporednih in porazdeljenih sistemov pospešujejo tako želje po reševanju preteklih in sodobnih kompleksnih problemov kot tudi nagel tehnološki napredek. Raziskovanje najrazličnejših računalniških arhitektur bo še vedno zelo pestro: od superračunalnikov, sistemov s skupnim in ločenim naslovnim prostorom, računalniških mrež in grozdov, sistemov GRID in še posebno sistemov v oblaku. Takšna raznolikost je, kljub naprednim modelom, naprednejšim programskim orodjem in sistemom, povzročila nemalo preglavic pri razvoju programske opreme.

S pojavitvijo porazdeljenih sistemov velikih razsežnosti se sicer odpirajo nove možnosti pri iskanju vzporednosti problemov, a hkrati te prinašajo težave novih dimenzij. Takšni sistemi so običajno sestav heterogenih računalniških sistemov (različni procesni, pomnilniški in komunikacijski viri) z različnimi zmogljivostmi. Porazdeljeni sistemi morajo zato podpirati najrazličnejše vrste računalniških virov. Zagotavljati morajo geografsko razpršenost, razpoložljivost, zanesljivost, varnost in raven storitve.

Razvoj vzporednih računalnikov je zaviral predvsem napredek v tehnologiji integriranih vezij. Čas, v katerem so načrtovalci razvili vzporedni sistem, je bil preprosto predolg. V vmesnem času so se namreč pojavili enojedrni procesorji, ki so bili primerljivo zmogljivi ali celo zmogljivejši. Sodobni pristop zasnove vzporednega sistema skoraj vedno temelji na tako imenovanem serisjkem (izvorno: *of-the-shelf*) pristopu, pri katerem vzporedni sistem vsebuje prosto dostopne serijske ali komercialne gradnike (sistemi MPP). V primeru zmogljivejših komponent na trgu se te v vzporednem sistemu zamenja in s tem neposredno poveča zmogljivost. Tako zasnovan sistem postane neobčutljiv na tehnološki napredek integriranih vezij. To je tipični primer odmika od klasične zasnove vzporednih sistemov k zasnovi porazdeljenih sistemov. Razmerje cena−zmogljivost je preprosto pomembnejše.

Največji izziv načrtovalcem vzporednih in porazdeljenih sistemov zagotovo še vedno predstavlja Amdahlov zakon (39). Trdi naslednje: pohitrimo delež p od vseh operacij, ki jih izvaja program, za faktor k.

Delež $(1-p)$ operacij pa ostane nespremenjen. To pomeni, da bo v povprečju p časa delovanje programa k-krat hitrejše in $1-p$ časa nespremenjeno. Potem pohitritev celotnega izvajanja programa določa izraz

$$S = \frac{1}{(1-p)+\frac{p}{k}}.$$

Tudi če p-ti del neskončno pohitrimo, bo največja celotna pohitritev le $(1/1-p)$. Amdahlov zakon je daleč najpomembnejši dejavnik, ki omejuje pohitritev računanja. Del algoritma, ki ga ne moremo izvesti vzporedno, bo vedno predstavljal omejitev pri pohitritvi reševanja problema. Tipični primer Amdahlove omejitve srečamo pri podjetju Turboinštitut, ki s pomočjo superračunalnika LSC Adria izvaja simulacije dinamike tekočin. Predprocesiranje in poprocesiranje podatkov sta zaporedni aktivnosti in predstavljata ozko grlo pri reševanju celotnega problema. Identičen Amdahlovemu zakonu je tudi Gustafson-Barsisov zakon, ki sicer upošteva dejstvo, da se v praksi s povečevanjem števila procesnih enot povečuje tudi velikost problema. To je sicer v nasprotju s predpostavko Amdahlovega zakona, ki pravi, da je količina časa v delih programa, ki se dajo izvajati, hkrati neodvisna od števila procesnih enot. S preprostim postopkom [60] se izkaže, da sta Gustafson-Barsisov in Amdahlov zakon povsem identična.

Pri vzporednem sistemu je pomembna pohitritev, medtem ko ima v porazdeljenem sistemu večji pomen prepustnost. Kljub vsem razlikam pa se oba pristopa od 90-ih let naprej vse bolj prekrivata in konvergirata. Tehnološki napredek v omrežjih omogoča, da se v vzporednem sistemu za povezovanje velikokrat uporabljajo omrežne tehnologije (Myrinet, Infiniband), po drugi strani pa so v porazdeljenem sistemu vse bolj prisotni večjedrni procesorji. Tudi na področju raziskav se pojma "porazdeljenost" in "vzporednost" velikokrat obravnavata povsem komplementarno. Videti je, da porazdeljeni sistemi, kot so cenovno ugodni grozdi in sistemi MPP, vse bolj izpodrivajo namenske vzporedne sisteme [15]. Ali bo prišlo do popolnega zatona namenskih sistemov, bo seveda pokazal čas.

Tudi programska oprema za vzporedne in porazdeljene sisteme je problematična. Trenutno so namreč v uporabi aplikacije, ki ne izkoriščajo povsem principa vzporednega izvrševanja (izvorno: *dusty deck problem*). Vrednost te programske opreme presega milijardne vsote in njeno prilagajanje je skorajda nemogoče. Gotovo pa je pričakovati, da bodo programerji vse bolj primorani k poznavanju novih principov, tehnik in orodij pri snovanju nove programske opreme. Edina rešitev na tem področju je torej čas, ki bo potreben za sprejetje novih pristopov v širši množici uporabnikov.

V relativno kratkem času smo na področju porazdeljenih sistemov dobili več vzorcev, katerih definicije velikokrat niso popolne. Sistem GRID se marsikje in marsikdaj brezvestno izkorišča v povsem tržne namene. Zaradi netočnih definicij pogostokrat srečujemo tudi izenačevanje pomena tehnologij, zato je bistveno, da se pri pojavitvi vsakega novega vzorca točno definirajo namen in funkcionalnost sistema ter stične točke med ostalimi, že obstoječimi tehnologijami oziroma sistemi. Univerzalni porazdeljeni sistem, ki bi ustrezal vsem možnim pogojem in bil primeren za reševanje vseh vrst problematike, trenutno ne obstaja. Vsak izmed obstoječih sistemov ima svoje slabosti in prednosti v določenih okoliščinah. V prihodnosti bo lahko prihajalo do kombiniranja različnih tehnologij. Če te konvergence ne bo zaslediti, pa lahko zagotovo pričakujemo še močnejši trend razvoja vmesnikov, ki omogočajo medsebojno komunikacijo med različnimi vrstami porazdeljenih sistemov.

3 Porazdeljena sistema BOINC in Condor

3.1 Sistem BOINC

Sistem BOINC (*Berkeley Open Interface for Network Computing*) je odprtokodna vmesna programska oprema oziroma visokoprepustni prostovoljni sistem. Zasnovan je bil v sklopu projekta SETI@home [4, 82], katerega cilj je zaznati morebitni nezemeljski obstoj življenja in pokazati oziroma dokazati upravičenost uporabe prostovoljnega sistema v znanstvenih panogah. Prvotni odjemalec za projekt SETI@home ni bil skladen s programsko opremo BOINC, saj je bil povsem namenski, brez kakršnih koli varnostnih mehanizmov. Veliko sodelujočih pri projektu je pričelo namenoma poneverjati rezultate izvajanja, vdori v sistem pa so postali vsakdanji. Zaradi takšnih dogodkov in pomanjkljivosti sistema so razvijalci v laboratoriju SSL (Space Science Laboratory) na univerzi v Kaliforniji (Berkeley) razvili krovno vmesno programsko opremo BOINC [3, 74].

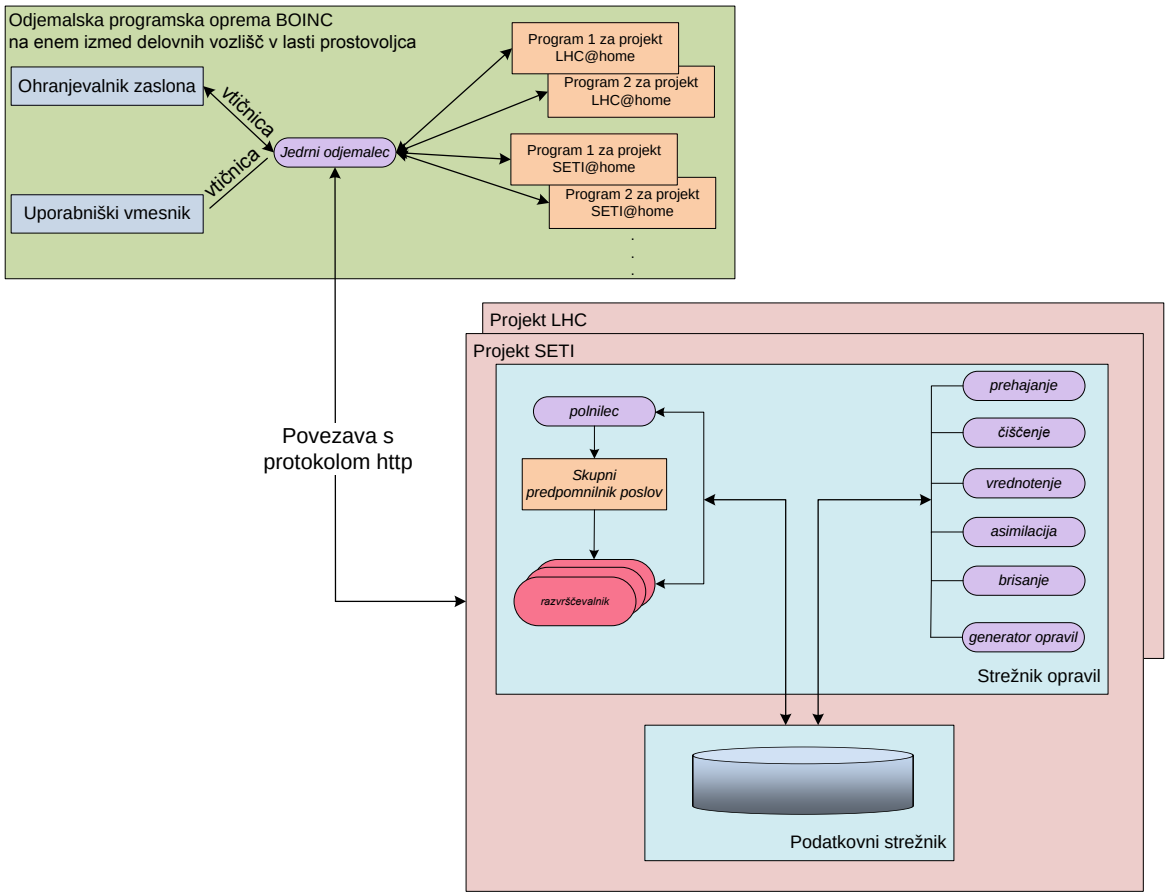
Prostovoljci (izvorno: *volunteer*), ki sodelujejo v prostovoljnem sistemu, so posamezniki ali skupine ljudi, ki so pripravljeni sodelovati pri souporabi svojih računalniških virov v koristne, pogostokrat človekoljubne namene. Iz centralnega dela sistema BOINC prenašajo opravila, jih izvedejo, rezultate pa pošljejo nazaj na centralni del. Sistem BOINC se uporablja v "projektih". Definiramo jih kot infrastrukturne postavitve (strežniki, podatkovne baze in programska oprema) organizacij (tipično akademskih), ki želijo pridobiti in uporabljati večje količine računalniških zmogljivosti. Primeri so: Einstein@home (iskanje gravitacijskih valov), Cosmology@home (iskanje najboljšega modela vesolja), LHC@home (CERN-ov hadronski trkalnik), SETI@home, PREDICTOR@home (napoved struktur beljakovin) in še mnogo drugih. Okolja, ki uporabljajo sistem BOINC, lahko s približno 600000 aktivnih vozlišč (osebnih računalnikov), zagotovijo približno 5,4 peta FLOPS procesne zmogljivosti.

Sistem BOINC je primeren predvsem za probleme, pri katerih je njihova segmentacija na manjše dele preprosta in potrebujejo ogromno procesnih in pomnilniških zmogljivosti. Prostovoljce je v javnem okolju izredno težko privabiti k sodelovanju, zato porazdeljeni sistemi, ki uporabljajo javne vire, poskušajo sodelujoče nagrajevati s pomočjo posebnih kreditnih točk, ki se objavljajo na znanih mestih [77]. Izkazalo se je, da so s tem pristopom sodelujoči bolj motivirani in še posebno navdušeni nad njihovo uvrstitvijo na točkovni lestvici v primerjavi z ostalimi. Kreditne točke predstavljajo merilo, koliko svojih računalniških virov je posameznik že daroval v souporabo in kakšna je njegova verodostojnost. Običajno se točke določajo na podlagi parametra CPE-ura, torej dela, ki ga CPE opravi v eni uri. V večini primerov se normalizira, saj so končne naprave heterogene. Trenutno ni znan noben mehanizem v sistemu BOINC, ki bi omogočal financiranje za opravljeno delo prostovoljcev. Že sama beseda "prostovoljec" izraža, da sodelujoči izvajajo aktivnosti zgolj na podlagi prostovoljne udeležbe in ne proti plačilu. Sodelujejo le v primeru, če jim projekt ne prinaša nevšečnosti, kakršnih koli tveganj na področju računalniške varnosti in drugih stroškov, na primer porabe energije. Prostovoljci, ki so pripravljeni na souporabo v sistemu BOINC, morajo namestiti ustrezno odjemalsko programsko opremo na svojem računalniškem sistemu – delovnem vozlišču. Gre za jedrni odjemalec (slika 3.1), dostopen s spletne strani projekta. Prostovoljec se mora po namestitvi odjemalca prijaviti na spletni strani izbranega projekta. S pomočjo naslova URL, ki identificira projekt, in posebne številke registrira odjemalsko

programsko opremo za izbrani projekt. Prostovoljci lahko sodelujejo na več projektih hkrati in lahko določijo, kakšen delež njihovih procesnih in pomnilniških virov bo namenjen souporabi. Imajo možnost določiti nivo porabe njihovih računalniških virov, na primer čas, ko se odjemalec na vozlišču aktivira, koliko pomnilniškega prostora ali prepustnosti povezave naj se rezervira za projekte. Določiti je mogoče celo parameter časa aktivnosti procesne enote (izvorno: *duty-cycle*), da pri izvajanju ne pride do prevelike disipacije procesorja.

3.1.1 Arhitektura sistema BOINC

Vsak projekt sistema BOINC ima prirejen namenski strežnik opravil (slika 3.1). Imenujemo ga centralno vozlišče. Odjemalska programska oprema se nahaja na delovnem vozlišču (običajno osebni računalnik, ni pa nujno) in deluje kot monoliten uporabniški program.



Slika 3.1: Arhitektura sistema BOINC

V resnici je sestavljena iz več delov [2]:

1. jedrni odjemalec je množica končnih avtomatov oziroma proces na delovnem vozlišču, ki skrbi za omrežno komunikacijo z razvrščevalnikom in podatkovnimi strežniki s pomočjo protokola HTTP. Opravlja prenos opravil s podatkovnega strežnika oziroma prenos rezultatov izvedenih opravil na podatkovni strežnik. Opravilo je sestav programa in podatkov, zato izvajanje prenesenih opravil pomeni izvajanje prenesenih programov nad prenesenimi podatki (izvajanje znanstvenega računanja). Jedrni odjemalec izvaja in upravlja s programi projekta (odloča o zaustavitvi, ponovnem zagonu in prenehanju delovanja programov). Programska koda

jedrnega odjemalca je prenosljiva na številne platforme operacijskih sistemov, vendar pa jo mora prostovoljec ročno namestiti na delovno vozlišče. Jedrni odjemalec poskrbi tudi za lokalno razvrščanje opravil (običajno po principu FIFO). Obstajajo tudi drugi pristopi razvrščanja [2]. Najpogostejši so: razvrščanje z namenom maksimizacije uporabe virov na delovnem vozlišču, razvrščanje z namenom izvedbe opravil v predvidenem časovnem roku, razvrščanje glede na prostovoljčeve rezervacije virov projektom ali razvrščanje tako, da je raznolikost prispevkov različnim projektom največja. Programi, ki se izvajajo na delovnem vozlišču, lahko namreč pripadajo več različnim projektom. Hkrati se izvaja toliko opravil, kolikor je procesorjev na delovnem vozlišču. Prostovoljec preko jedrnega odjemalca postavlja mejnike najmanjšega in največjega števila opravil, ki jih je delovno vozlišče pripravljeno sprejeti od posameznega projekta. Ko količina dela pade pod najmanjši možen nivo, jedrni odjemalec ponovno vzpostavi povezavo s strežniki projektov in prične s prenosi novih opravil. Zaključek izvajanja opravil je dvostopenjski proces: rezultati izvedenih opravil se najprej prenesejo na podatkovni strežnik projekta, nato pa jedrni odjemalec od razvrščevalnika projekta ponovno zahteva nova opravila. V slednjem koraku se razvrščevalnik obvesti tudi o vseh izvedenih opravilih;

2. uporabniški vmesnik omogoča upravljanje z jedrnim odjemalcem. Povezava z jedrnim odjemalcem je običajno lokalna, ni pa nujno. Krmiljenje jedrnega odjemalca lahko namreč poteka tudi s pomočjo oddaljenega klica (RPC). Omogoča tudi tabelarični pregled vseh projektov, s katerimi je delovno vozlišče povezano, in vseh aktivnih datotečnih prenosov;
3. ohranjevalnik zaslona se aktivira takrat, ko prostovoljec dlje časa ni aktiven na delovnem vozlišču. Takrat se aktivira izvajanje opravil projektov, jedrni odjemalec pa posreduje vse informacije o trenutnem izvajanju opravil na zaslon v obliki ohranjevalnika zaslona;
4. programi, ki jih delovno vozlišče izvaja, so s pomočjo vmesnikov API povezani s posebnimi programskimi knjižnicami, ki omogočajo upravljanje s procesi ter operacijo vmesnega shranjevanja stanja računanja (izvorno: *checkpoint*). Program lahko jedrnemu odjemalcu posreduje dvoje: trenutno porabo časa CPE in trenutek, v katerem se je izvedlo shranjevanje vmesnega stanja izvajanja opravila. V principu lahko v sistemu BOINC izvajamo programe poljubne vrste. Zagotoviti pa moramo povezavo s posebnimi programskimi knjižnicami. Možna je tudi uporaba programske opreme − tako imenovane ovojnice (izvorno: *wrapper)*, ki poskrbi za komunikacijo z jedrnim odjemalcem, program pa izvaja v obliki povsem ločenega procesa.

Iz množice procesov sestavljen strežnik opravil ima v sklopu projekta najpomembnejšo vlogo (slika 3.1). Generator opravil ustvarja nova opravila in njim pripadajoče podatkovne datoteke. V sklopu projekta SETI@home generator opravil bere digitalne podatke z magnetnih trakov, kjer so zapisani podatki z radijskih teleskopov. Po branju jih prepiše v podatkovne datoteke, jih doda k programu in ustvari opravila za delovna vozlišča. Razvrščevalnik obdeluje zahteve s strani jedrnih odjemalcev. V zahtevi (datoteka v obliki XML), ki jo jedrni odjemalec pošlje strežniku opravil, se nahaja informacija o vseh izvršenih opravilih do tistega trenutka. Pošlje pa se tudi zahteva po novih opravilih za delovno vozlišče. Zahtevi so dodane še uporabniške nastavitve jedrnega odjemalca, s pomočjo katerih se obvesti strežnik opravil o načinu dela jedrnega odjemalca.

Projekt ima lahko enega ali več programov, ki so jim prirejene ena ali več vhodnih datotek. Obstaja lahko tudi več različic programov, namenjenih različnim platformam končnih naprav (Windows, Linux OS X). Opravilo, ki se pošlje delovnemu vozlišču v izvajanje, je sestavljeno

iz programa oziroma verzije programa, množice referenc na vhodne datoteke, množice spremenljivk okolja, v katerem se program izvaja, in množice ukazov ukazne vrstice. Opravilo določa tudi omejitve, ki jim mora delovno vozlišče ustrezati. Običajno sta navedeni zahtevani procesna in pomnilniška zmogljivost.

3.1.2 Življenjski cikel opravila v sistemu BOINC

Življenjski cikel opravila, ki ga izvaja delovno vozlišče, sledi določenim korakom [3,6]:

1. poseben proces na **strežniku opravil** (generator opravil) ustvarja opravila tako, da združuje programe s podatkovnimi datotekami;
2. strežnik opravil s pomočjo razvrščevalnika opravil porazdeljuje opravila na delovna vozlišča tako, da ustrezajo omejitvam − razvrščanje. Razvrščanje se izvaja v obliki posebnega programa CGI. V vsakem trenutku je lahko aktivnih tudi več sto primerkov tega programa (za vsako delovno vozlišče en program). Postopek razvrščanja poteka tako, da se primerja zahteve in omejitve računalniških virov z zahtevami in omejitvami, ki se nahajajo v opravilih. Poteka na podlagi več omejitev:
    a. opravilo se pošlje samo tistemu **delovnemu vozlišču**, ki ima na voljo določeno vrsto platforme za izvajanje programa opravila (na primer ustrezen operacijski sistem);
    b. opravilo se pošlje na delovno vozlišče, če zadosti zahtevanim računalniškim virom (dovolj velika procesna zmogljivost, dovolj veliko pomnilniškega prostora, dovolj velika prepustnost povezave s strežnikom opravil in **podatkovnim strežnikom**);
    c. opravilo se pošlje na delovno vozlišče, če zadosti uporabniškim omejitvam (omejitev števila primerkov opravil, ki jih lahko prejme eno delovno vozlišče);
    d. opravilo se pošlje samo na točno določeno vrsto delovnega vozlišča (delovno vozlišče s procesnimi enotami Intel).

    Lahko trdimo, da je razvrščanje v sistemu BOINC proces, ki upošteva kombinacijo navedenih omejitev;
3. vsako delovno vozlišče s povezav URL prenese vse potrebne datoteke za izvajanje programa v opravilu. Prejme lahko tudi več opravil. S tem se zmanjša obremenitev strežnika opravil in hkrati zagotovi dovolj dela za delovno vozlišče tudi v primeru izpada omrežne povezave s strežnikom opravil;
4. ko delovno vozlišče izvede opravilo, prenese rezultate na podatkovni strežnik v obliki podatkovnih datotek z rezultati izvajanja programa;
5. poseben proces prehajanje na strežniku opravil spremeni stanje opravila (neaktivno, aktivno, uspešno končano, končano z napako) v podatkovni bazi. S tem se samodejno zmanjša obremenitev razvrščanja. Nato pa se sproži še proces vrednotenje, ki preveri rezultate. V procesu vrednotenje je najprej izbran kanonski rezultat (pravilen končni rezultat) in določeno število kreditnih točk, ki jih prejmejo prostovoljci. Kreditne točke se določijo glede na porabljen čas procesne enote in uporabljene pomnilniške zmogljivosti. Te informacije hrani jedrni odjemalec in jih sporoča strežniku opravil oziroma procesu vrednotenja;
6. če je rezultat pravilen oziroma verodostojen, se ločeno shrani v podatkovno bazo na podatkovnem strežniku. Za to skrbi proces asimilacija na strežniku opravil;
7. ko so vsa opravila izvedena, proces brisanje na strežniku opravil poskrbi, da se odstranijo vse datoteke, ki so pripadale opravilu. Proces čiščenje na strežniku opravil pa uredi, da se odstranijo tudi vsi dotični skrbniški vnosi.

V trenutku razvrščanja razvrščevalnik opravil posega v podatkovno bazo opravil. Podatkovna baza projekta na centralnem strežniku lahko vsebuje tudi več milijonov vnosov, razvrščevalnik pa lahko postreže le nekaj več sto zahtevam v omejenem času. Zaželena rešitev bi bila, da bi razvrščevalnik pri vsaki zahtevi opravil pregled vseh vnosov opravil v podatkovni bazi in odjemalcu poslal tisto opravilo, ki mu najbolj ustreza. V tem primeru bi bilo razvrščanje optimalno. Iz praktičnih razlogov pa je povsem neučinkovito. V resnici razvrščevalnik posega v poseben skupen in hiter medpomnilnik opravil z nekaj več tisoč vnosi. Poseben proces polnilec periodično polni medpomnilnik z novimi opravili, ki jih bere iz podatkovne baze opravil. Razvrščevalniki za vsako vozlišče posebej pregledujejo medpomnilnik in jedrnim odjemalcem na delovnih vozliščih vračajo opravila, ki ustrezajo omejitvam.

Pri pregledovanju medpomnilnika se izvajajo naslednje aktivnosti:

1. pregledovanje medpomnilnika opravil se prične na naključni točki. Za vsako opravilo se izvrši preverjanje ustreznosti, ki ne zahteva poseganja v zunanjo bazo – izven predpomnilnika opravil. Na primer: preverjanje ustrezne količine pomnilnika na delovnem vozlišču in preverjanje količine časa, ki ga delovno vozlišče potrebuje za uspešno izvrševanje opravila. Potek takšnega preverjanja je hiter;
2. če opravilo ustreza zgornjim zahtevam, se izvedejo dodatne primerjave, ki pa že zahtevajo dostop do podatkovne baze. Na primer: preverjanje, da enemu delovnemu vozlišču ni poslanih več primerkov istega opravila;
3. zgornji dve točki se ponavljata, dokler delovno vozlišče oziroma jedrni odjemalec ne prejme dovolj opravil.

Sistem BOINC s podvajanjem opravil sicer omogoča redundančno izvajanje, vendar tak pristop lahko pelje do izrazite neučinkovitosti (tudi do 50 %). V novejših različicah sistema BOINC je možna tako imenovana adaptivna replikacija, pri kateri razvrščevalnik hrani ocene delovnih vozlišč glede na njihovo preteklo pogostost pojavljanja napak. Če se opravilo pošilja na zelo zanesljivo delovno vozlišče, bo verjetnost pravilnega rezultata večja in potreba po replikaciji izvajanja opravila manj kritična. Prihranek delovnih virov lahko bistveno vpliva na učinkovitost sistema. Princip je naravnan tudi tako, da prostovoljci prejmejo dobro oceno le po daljšem časovnem obdobju, slabo oceno pa lahko že v zelo kratkem času.

3.1.3 Večjedrne in večprocesorske arhitekture v sistemu BOINC

Sistem BOINC je primeren za več vrst arhitektur na delovnih vozliščih. Uporabljajo se lahko različice programov, ki izkoriščajo tudi posebne računalniške vire, kot so na primer grafični strojni vmesniki NVIDIA ali hibridni procesorji Cell. Programi so lahko zasnovani eno- ali večnitno in s tem izkoriščajo večjedrno zasnovo procesne enote. V splošnem je jedrni odjemalec zmožen zaznati prisotnost posebnih naprav na delovnem vozlišču in njihove opise skupaj z zahtevo po opravilih posredovati strežniku opravil. Na strežniku opravil je razvrščevalnik povezan s posebno funkcijo planiranja (izvorno: *application planning function*), ki omogoča učinkovitejše prirejanje opravil delovnim vozliščem. Sleherni različici programa je prirejen planiran razred. Skupaj z opisom delovnega vozlišča se posredujeta funkciji planiranja. Ta na podlagi prejetega vrne informacijo o pričakovani zmogljivosti delovnega vozlišča, nato pa določi povprečno število procesnih enot ali jeder, ki bi lahko bila hkrati uporabljena pri izvajanju programa na delovnem vozlišču. Razvrščevalnik je tako zmožen delovnemu vozlišču dodeliti tisto različico programa, pri kateri je pričakovana zmogljivost največja.

3.1.4 Varnost v sistemu BOINC

Preden se podatkovne datoteke in različice programov pošljejo na delovna vozlišča, se na posebnih in od omrežja povsem ločenih vozliščih opravi digitalni podpis programov. S stališča projekta, ki sistem BOINC uporablja, je s tem zagotovljena overitev programov. Postopek je še posebno pomemben pri projektih, ki so javnega značaja.

Najpogostejše hekerske zlorabe pri projektih so ohromitve storitve (izvorno: *denial of service* − DoS). Zlonamerno delovno vozlišče lahko na podatkovni strežnik vrača zelo dolge rezultate opravil (velike izhodne podatkovne datoteke). Podobno preobremenitev strežnikov se doseže tudi s pogostim in namernim javljanjem napak na delovnem vozlišču. To povzroči veliko ponovnih prenosov opravil s strežnika opravil na delovno vozlišče. Prvi problem se rešuje z omejitvijo velikosti rezultatov, ki jih strežnik še dovoljuje sprejemati. Poseben program CGI skrbi, da se odstranijo vsi prenosi, pri katerih je velikost podatkov večja od vnaprej določene. Večjemu številu prenosov se izognemo z določitvijo gornje meje števila prenosov opravil v določenem časovnem obdobju (na primer v enem dnevu). V primeru prijave napake se delovnemu vozlišču število dopustnih prenosov zmanjša za 1. Torej bo vozlišče, ki pogosto javlja napake, prej ali slej pristalo na meji prenosa enega rezultata na dan. Ko pa se omenjeni problem odpravi oziroma napak ni več, bo sčasoma lahko delovno vozlišče ponovno opravljalo več prenosov na strežnik opravil. Delovna vozlišča lahko tudi namerno vračajo napačne rezultate. Tovrstni problem se rešuje s pošiljanjem več primerkov opravila večjemu številu delovnih vozlišč (izvorno: *adaptive replication*), najbolje v različne države. Vsak projekt ima definirano število delovnih vozlišč (običajno 5), ki prejmejo po eno kopijo opravila. Kvorum za določitev kanonskega rezultata določajo rezultati najmanj treh delovnih vozlišč. V primeru napačnih rezultatov vozlišče ne prejme kreditnih točk. Če je napačnih rezultatov več, se delovnemu vozlišču celo prepove sodelovati v projektu.

Če želi zlonamerni pridobiti več kreditnih točk, kot mu jih pripada, projektu ne škoduje neposredno. Gotovo pa tak pristop vpliva na zaupanje ostalih sodelujočih pri projektu. Manj ko je projekt verodostojen, manj bo sodelujočih prostovoljcev pri projektu. Tudi ta problem se rešuje s pomočjo redundančnega izvajanja opravil. Primerljivi uporabniki, ki vračajo verodostojne rezultate, lahko prejmejo enako število kreditnih točk. Če jih nekdo želi pridobiti več, je v okviru projekta že samodejno poskrbljeno, da se kreditne točke porazdelijo vsem volonterjem enakomerno in vsiljivec nič ne pridobi.

3.2 Sistem Condor

Sistem Condor [42, 66, 67, 75], katerega začetki segajo v leto 1984 na Univerzo v Wisconsin−Madison (ZDA), se deli na dva podprojekta: Condor-HTC ter Condor-G (Condor-Grid). Condor-G je podprojekt, ki se uporablja za povezavo sistema Condor z različnimi vrstami sistemov GRID (na primer EGEE, ARC) in v delu ni obravnavan. Condor-HTC je, podobno kot sistem BOINC, visokoprepustni (HTC – *High throughput computing*) sistem oziroma izpopolnjeni paketni sistem (izvorno: *batch system*). Naloga tega visokoprepustnega sistema je izkoriščanje vseh prostih oziroma v souporabo podanih računalniških virov (vozlišč). Običajno so povezani z omrežno tehnologijo (LAN). Ko je neko vozlišče nezasedeno z lokalnimi opravili, je lahko na voljo za souporabo in izvajanje zunanjih opravil. Pri takšnih sistemih je zaželena čim večja prepustnost opravil v daljšem časovnem obdobju (dnevih, mesecih in celo letih). Opravilo pojmujemo kot sestav programa in podatkov. Program izvaja aktivnosti nad podatki. Za vsako oddano opravilo se v sistemu izbere

primerno prosto vozlišče, ki opravilo izvede. Če vozlišče ni zanesljivo ali pa prihaja do pogostih napak, se lahko izvajanje opravila prenese na drugo, zanesljivejše vozlišče.

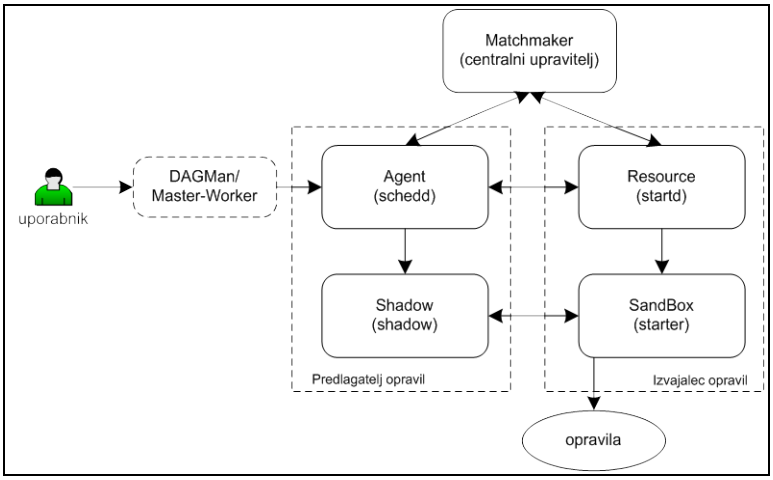
Lastnosti vozlišč in opravil so opisane s posebnim jezikom ClassAd (izvorno: *class advertising*). Opisi se v sistemu Condor pošiljajo centralnemu vozlišču (centralnemu upravitelju), ki izvede uparjevanje (izvorno: *matchmaking*) opravil z vozlišči. V principu se izvaja postopek razvrščanja. Razvrščanje je bilateralen proces, ki na podlagi podanih preferenc vozlišč in preferenc opravil preslika opravila na primerna vozlišča. Da bi bila informacija o stanju okolja sveža, se opisi vozlišč centralnemu upravitelju pošiljajo periodično. Pristop je torej podoben pristopu v sistemu BOINC. Primer opisa ClassAd za vozlišče in opravilo prikazuje Tabela 1.

Opis ClassAd za vozlišče	Opis ClassAd za opravilo

Type = "Machine";
Activity = "Idle";
DayTime = 36107 // current time
// in seconds since midnight
KeyboardIdle = 1432; // seconds
Disk = 323496; // kbytes
Memory = 64; // megabytes
State = "Unclaimed";
LoadAvg = 0.042969;
Mips = 104;
Arch = "INTEL";
OpSys = "SOLARIS251";
KFlops = 21893;
Name = "leonardo.cs.wisc.edu";
ResearchGroup = {"raman", "miron",
"solomon", "jbasney" };
Friends = {"tannenba", "wright" };
Untrusted = {"rival", "riffraff" };
Rank = member(other.Owner, ResearchGroup) * 10 + member(other.Owner, Friends);
Constraint = !member(other.Owner, Untrusted) && Rank >= 10 ? true : Rank > 0 ? LoadAvg<0.3 && KeyboardIdle>15*60 : DayTime < 8*60*60 || DayTime > 18*60*60;
	Type = "Job";
	QDate = 886799469;
	// Submit time secs. past 1/1/1970
	CompletionDate = 0;
	Owner = "raman";
	Cmd = "run_sim";
	WantRemoteSyscalls = 1;
	WantCheckpoint = 1;
	Iwd = "/usr/raman/sim2";
	Args = "-Q 17 3200 10";
	Memory = 31;
	Rank = KFlops/1E3 + other.Memory/32;
	Constraint = other.Type == "Machine" && Arch == "INTEL" && OpSys == "SOLARIS251" && Disk >= 10000 && other.Memory >= self.Memory;

Tabela 1: Primer opisa ClassAd za vozlišče in opravilo v sistemu BOINC

Parametri, ki določajo opis opravila ali vozlišča, morajo biti skladni z vnaprej specificirano obliko, ki jo določa ClassAd. Med pomembne lastnosti spada na primer *LoadAvg*, ki določa, kakšno stopnjo obremenjenosti ima lahko neko vozlišče, da sme prejeti opravilo. Sistem Condor (slika 3.2) sestavljajo vozlišča: centralni upravitelj, en ali več predlagateljev opravil in en ali več izvajalcev opravil. Vsako izmed teh ima specifične naloge. Ker imajo tako opravila kot tudi izvajalci opravil svoje zahteve, je naloga centralnega upravitelja združevati opravila z izvajalci opravil tako, da so njihovi opisi ClassAd skladni. Predlagatelji opravil so vozlišča, ki oddajajo opravila v sistem Condor. Ta se na izvajalcih opravil izvedejo.



Slika 3.2: Osnovna zasnova sistema Condor

Izvajalci opravil pa so vozlišča, ki so namenjena izključno izvajanju opravil v sistemu Condor, ali vozlišča, ki sodelujejo v sistemu Condor le, ko niso zasedena z lokalnimi opravili. Uporabniki, ki želijo s sistemom Condor izkoriščati proste vire, lahko svoja opravila oddajo v sistem neposredno ali pa z uporabo posebnih dodatnih programskih plasti *DAGMan* ali *Master-Worker* (poglavje 3.2.5.1 in 3.2.5.2). Opisi procesa *matchmaker* na centralnem upravitelju, procesa *schedd* in *shadow* na predlagatelju ter procesa *startd* in *starter* na izvajalcu so podani v nadaljevanju.

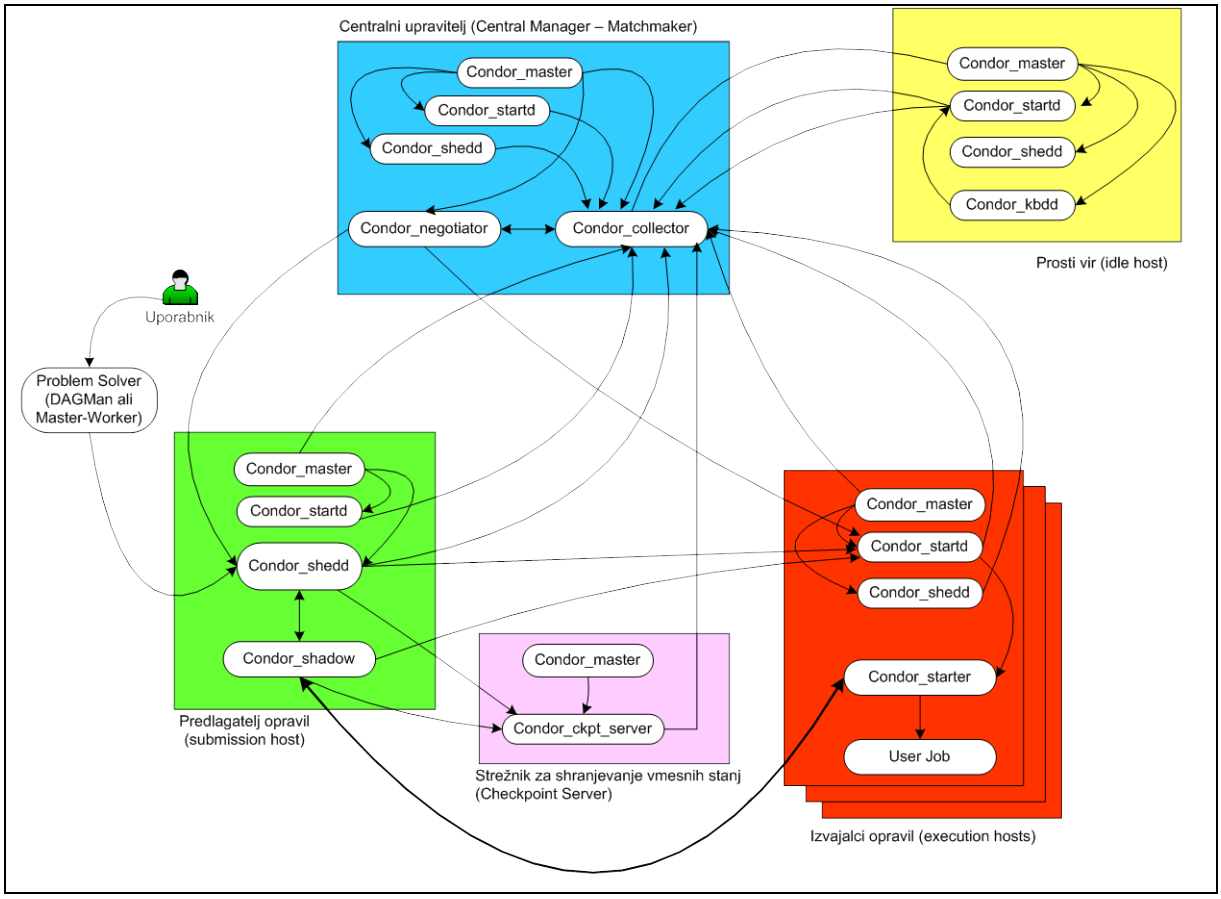
3.2.1 Arhitektura sistema Condor

Jedro sistema Condor sestavljajo posamezni programski procesi, ki so v sistemu vedno aktivni. Slika 3.3 kaže medsebojno sodelovanje med vozlišči, ki skupaj tvorijo sistem Condor. Vsako vozlišče ima določeno število aktivnih procesov. Centralni upravitelj ima globalni pregled nad vsemi vozlišči in skrbi za ustrezno preslikavo opravil na vozlišča. Predlagatelji opravil oddajajo opravila v sistem, izvajalci opravil pa jih izvajajo. Dodatno vozlišče za shranjevanje vmesnih stanj deluje kot pomožen prostor, ki omogoča shranjevanje vmesnih stanj izvajanja opravil na izvajalcih opravil. Zagotavlja večjo zanesljivost in hitrejše reševanje problemov.

Procesi so sledeči:

1. proces *condor_master* je aktiven na vseh vozliščih okolja in omogoča aktivnost vseh ostalih procesov na vozliščih. Ustvari procese *condor_startd*, *condor_schedd* in periodično preverja, če obstajajo morebitne nadgradnje teh procesov. Če kateri koli od navedenih procesov postane neaktiven (napaka ali nepravilno delovanje procesa), potem proces *condor_master* poskrbi za ustrezno obveščanje skrbnika vozlišč sistema Condor in ponovno zažene proces v okvari. Prav tako omogoča izvajanje skrbniških ukazov (zagon, ustavitev, sprememba nastavitev procesov) preko neposrednega in oddaljenega dostopa;
2. proces *condor_startd* je aktiven na vseh vozliščih sistema Condor, razen na vozlišču, ki skrbi za shranjevanje vmesnih stanj. Naloga tega procesa je, da procesu *condor_collector* na centralnem upravitelju pošilja informacijo o pogojih, ki jih izvajalec opravil zahteva, preden se na njem prične izvajati opravilo. Ko postane

proces *condor_startd* pripravljen na izvajanje opravila, ustvari nov proces, *condor_starter*;



Slika 3.3: Medsebojne interakcije med vozlišči oziroma procesi v sistemu Condor

3. zaganjalnik opravil c*ondor_starter* je proces, ki je aktiven samo na izvajalcu opravil in opravilo zažene. Pripravi okolje za nadzor in izvajanje opravila. Ko je izvajanje opravila končano, proces *condor_starter* prenese rezultat predlagatelju opravila in zaključi z aktivnostjo. Rezultat se lahko shrani tudi na skupni hrambni prostor, ki ga zagotavlja posebno vozlišče za shranjevanje vmesnih stanj. Običajno je dostopno preko protokola NFS;
4. proces *condor_schedd* je aktiven na vseh vozliščih, razen na vozlišču, ki skrbi za shranjevanje vmesnih stanj. Ko uporabnik oddaja opravila v izvajanje, se najprej postavijo v lokalno čakalno vrsto na predlagatelju opravil, ki jo upravlja proces *condor_schedd*. Posebni ukazi (*condor_submit*, *condor_q*, *condor_rm*) omogočajo upravljanje s procesom *condor_schedd*, ki poskrbi, da uporabnik lahko odda ali odstrani opravila iz čakalne vrste in nadzira njeno stanje. Če proces *condor_schedd* na predlagatelju opravil ni aktiven, noben od omenjenih ukazov ne deluje. Poleg omenjenega proces *condor_schedd* dodatno posreduje tudi informacije o zahtevah opravil po želenih virih procesu *condor_collector*. Opravilu je dodan opis v jeziku ClassAd, ki navaja, kakšne vrste podenot vozlišča (procesna in pomnilniška zmogljivost) so za njegovo izvajanje najprimernejše. V trenutku, ko se zahtevi opravila ugodi (oziroma se uspešno poišče primerno delovno vozlišče), proces *condor_schedd* ustvari nov proces, *condor_shadow*, ki poskrbi, da izvajalec opravil uspešno prevzame opravilo v izvrševanje;

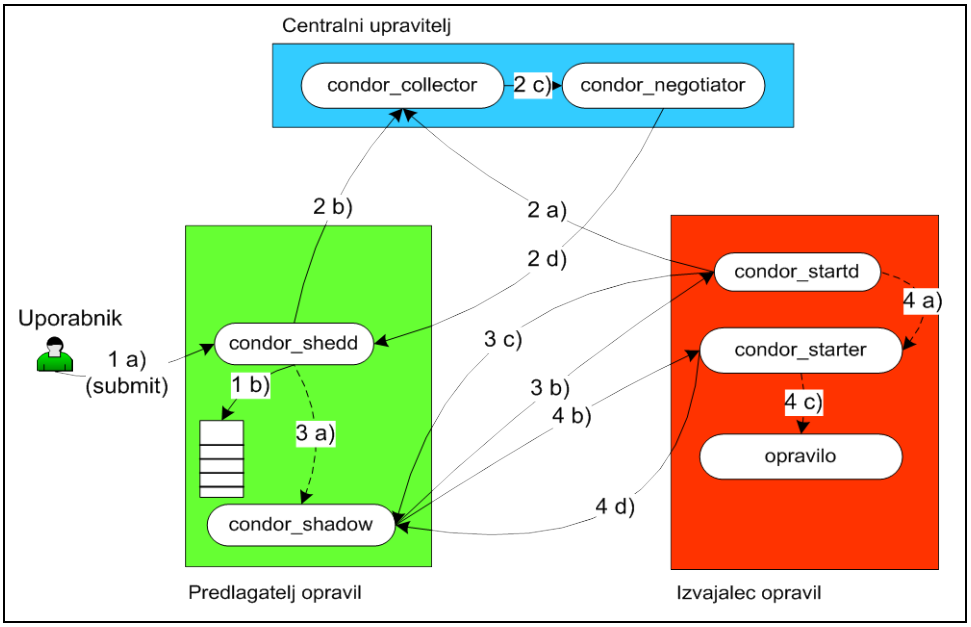
5. proces *condor_shadow* je vedno aktiven le na predlagatelju opravil. Izvaja sistemske klice preko oddaljenega dostopa, ki jih prejema od izvajalcev opravil, in skrbi za pravočasno shranjevanje vmesnega stanja izvajanja opravila. Vsi sistemski klici, ki se izvedejo na izvajalcu opravil, se s pomočjo protokola RPC posredujejo preko omrežja nazaj na predlagatelja opravil, ki klice izvede, rezultate pa posreduje nazaj na izvajalca opravil;
6. proces *condor_collector* je aktiven le na centralnem upravitelju in zbira informacije o stanju vozlišč in procesov v sistemu Condor;
7. proces *condor_negotiator* je ravno tako aktiven le na centralnem upravitelju. Opravlja preslikavo opravil na vozlišča, ki so med seboj skladna. S pomočjo opisov *ClassAd* vozlišč in opravil poišče optimalno preslikavo opravil na primernega izvajalca opravil. Če ima uporabnik z višjo prioriteto na voljo opravila za izvajanje, lahko proces *condor_negotiator* na vozlišču prekine trenutno izvajanje opravila uporabnika z nižjo prioriteto in vozlišče dodeli uporabniku z višjo prioriteto − razvrščanje s predpravico (izvorno: *preemptive scheduling*);
8. v sistemu Condor so lahko aktivna tudi vozlišča, ki niso v celoti namenska. Svoje vire ponujajo samo v primeru lastnikove neaktivnosti. V ta namen obstaja proces *condor_kbdd*, ki je aktiven na izvajalcu opravil. Proces *condor_kbdd* lahko periodično preko uporabniškega vmesnika (tipkovnica, miška) preverja, ali je izvajalec opravil prost. Informacijo o stanju pa posreduje procesu *condor_startd*;
9. velikokrat prihaja do stanj, ko izvajalec opravil ni več na voljo za izvajanje opravil. To se zgodi, če prične lastnik izvajati na njem lokalna opravila. V tem primeru proces *condor_ckpt_server* na posebnem vozlišču omogoča shranjevanje vmesnega stanja izvajanja opravil. Gre za prenos datotek, ki opisujejo stanje izvajanja opravila iz izvajalca opravila na vozlišče, to skrbi za shranjevanje vmesnih stanj. Če vozlišča te vrste v sistemu Condor ni na voljo ali ni aktivno, se vmesna stanja izvajanja lahko shranijo neposredno na izvajalcu opravila (v primeru krajše zasedenosti izvajalca) ali na predlagatelju opravila (v primeru napake ali daljše zasedenosti izvajalca). V obeh primerih velja, da je vmesno stanje izvajanja opravila pravzaprav novo opravilo. Na predlagatelju se zato zopet postavi v čakalno vrsto opravil.

3.2.2 Življenjski cikel opravila v sistemu Condor

Denimo, da so v sistemu Condor centralni upravitelj, predlagatelj opravil in izvajalec opravil. Na predlagatelju opravil je aktiven uporabnik, ki želi izvesti nek posel oziroma opravilo. Izvajalec opravil naj bo vozlišče, ki je prosto. Življenjski cikel opravila poteka po vnaprej določenem protokolu. Koraki protokola, ki jih prikazuje slika 3.4, so sledeči:

1. Oddaja opravila na predlagatelju opravil:
    a. uporabnik z ukazom *submit* na svojem vozlišču odda opravilo v sistem Condor;
    b. proces *condor_schedd* na predlagatelju opravil shrani opravilo v čakalno vrsto.
2. Preslikava opravila na ustreznega izvajalca opravil:
    a. proces *condor_startd* na izvajalcu opravil odda informacijo o lastnostih svojih virov (procesnih, pomnilniških, hrambnih) procesu *condor_collector* na centralnem upravitelju. Ta korak se izvaja periodično in neodvisno od koraka 2 b;
    b. proces *condor_schedd* na predlagatelju opravil odda procesu *condor_collector* na centralnem upravitelju informacijo o zahtevah opravila po virih, ki naj bi bili na voljo na izvajalcu opravil;

c. proces *condor_negotiator* opravi najugodnejšo preslikavo opravila na enega od razpoložljivih prostih izvajalcev opravil (uporabijo se opisi ClassAd in efektivne uporabniške prioritete);
d. proces *condor_negotiator* na centralnem upravitelju obvesti proces *condor_schedd* na predlagatelju opravil o ustrezno izbranem izvajalcu opravil, ki lahko izvede predloženo opravilo.



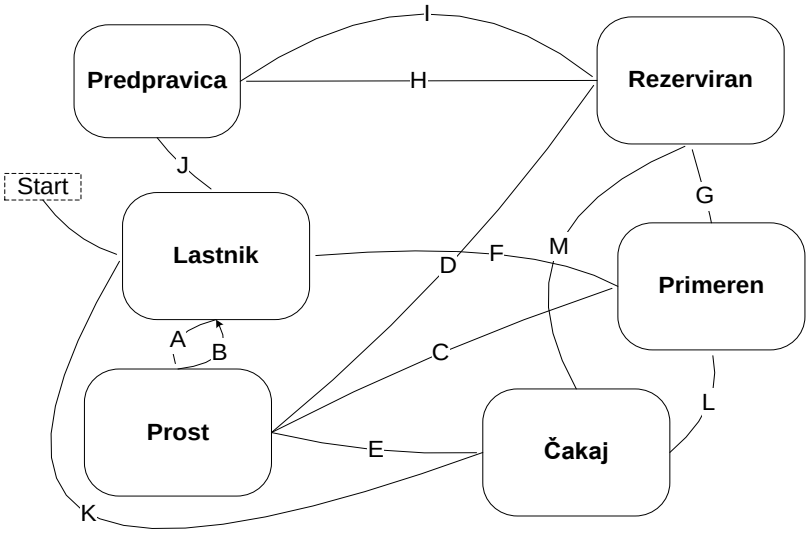
Slika 3.4: Življenjski cikel opravila v sistemu Condor

3. Vzpostavitev komunikacije med *condor_schedd* na predlagatelju opravil in *condor_startd* na izvajalcu opravil:
   a. proces *condor_schedd* na predlagatelju opravil ustvari nov proces, *condor_shadow*;
   b. proces *condor_shadow* na predlagatelju opravil prenese procesu *condor_startd* na izvajalcu opravil zahteve opravila;
   c. proces *condor_startd* na izvajalcu opravil s posebnim postopkom izvede oceno o primernosti virov za izvajanje opravila. Če so viri primerni in je izvajalec opravil še na voljo, proces *condor_startd* pošlje procesu *condor_shadow* na predlagatelju opravil potrditev, v nasprotnem primeru pa se celoten postopek prekine.
4. Izvajanje opravila na izvajalcu opravil:
   a. proces *condor_startd* na izvajalcu opravil ustvari nov proces, *condor_starter;*
   b. proces *condor_shadow* na predlagatelju opravil pošlje procesu *condor_starter* na izvajalcu opravil izvršljiv program;
   c. proces *condor_starter* na izvajalcu opravil prične izvajati prejeti program. Vsi sistemski klici (npr. branje in zapisovanje datotek) se posredujejo predlagatelju opravil (uporaba protokola RPC). Izvrši jih proces *condor_shadow*, rezultate sistemskih klicev pa posreduje nazaj na izvajalca opravil;
   d. po končanem izvajanju opravila proces *condor_strater* na izvajalcu opravil pošlje procesu *condor_shadow* na predlagatelju opravil rezultate uspešno opravljenega opravila.

3.2.3 Stanja vozlišč in prehodi med stanji v sistemu Condor

Vsako od vozlišč sistema Condor je lahko v enem od naslednjih stanj (slika 3.5):

1. stanje **lastnik** − izvajalec opravil je obremenjen z aktivnostmi lastnika in vozlišče v tem stanju ni del sistema Condor. Ko vozlišče zopet postane del sistema Condor, se praviloma njegova aktivnost prične v stanju *lastnik*;



Slika 3.5: Stanja vozlišč in prehodi v sistemu Condor

2. stanje **prost** − izvajalec opravil je na voljo za izvajanje opravil, vendar trenutno ne izvaja nobenega opravila;
3. stanje **primeren** − izvajalec opravil je na voljo za izvajanje opravil. Proces *condor_negotiator* ga je uspel povezati z ustreznim procesom *condor_schedd*, a se proces *condor_shedd* z njim še ni povezal;
4. stanje **rezerviran** − izvajalec opravil je rezerviran s strani procesa *condor_schedd*, opravilo je v stanju izvajanja;
5. stanje **predpravica** − izvajalec opravil je bil sicer rezerviran s strani procesa *condor_schedd*, vendar pa sedaj svoje izvajanje prekinja zaradi enega od naslednjih razlogov:
    a. lastnik je na izvajalcu opravil zopet pričel z lokalnim delom,
    b. na voljo so nova opravila za izvajalca opravil, ki imajo višjo prioriteto,
    c. na voljo so primernejša opravila za izvrševanje izvajalcu opravil;
6. stanje **čakaj** − izvajalca opravil ne uporablja lastnik niti ne sodeluje v sistemu Condor, zato preprosto čaka. V tem stanju se običajno izvajajo zunanja opravila (na primer preko vmesnika, ki omogoča povezavo s sistemom BOINC).

Med stanji so na voljo naslednji prehodi:

1. prehod A − minilo je določeno število časovnih enot od trenutka, ko lastnik ni več uporabljal izvajalca opravil, ki je zato lahko na voljo za izvajanje opravil v sistemu Condor;
2. prehod B − izvajalca opravil je zopet pričel uporabljati lastnik za lokalna opravila;
3. prehod C − proces *condor_negotiator* je za izvajalca opravil uspel najti primerno opravilo;

4. prehod D − neposreden preskok v stanje *rezerviran* se zgodi v primeru, ko proces *condor_negotiator* uspe najti primerno opravilo za izvajalca opravil, vendar v tem primeru proces *condor_schedd* prejme potrdilo in prične s protokolom rezervacije vozlišča (izvajalca opravil), še preden proces *condor_startd* uspe pridobiti potrdilo od procesa *condor_negotiator*;
5. prehod E − ta prehod se zgodi v primerih, ko je izvajalec opravil nameščen tako, da ne opravlja nikakršnih aktivnosti ali pa v tem stanju izvaja opravila z najnižjo prioriteto;
6. prehod F − prehod se opravi, če izvajalca opravil zopet prične uporabljati lastnik, ali v primeru poteka časovnega intervala, znotraj katerega mora proces *condor_schedd* vzpostaviti povezavo s procesom *condor_startd*. Povezava je potrebna za izmenjavo vseh informacij v povezavi z izvajanjem opravila. Če tega proces *condor_schedd* ne opravi, je treba poskrbeti, da je izvajalec opravil na voljo ostalim predlagateljem opravil v sistemu Condor. Prehod se lahko opravi tudi v primeru uspešnega začetka komunikacije med procesoma *condor_schedd* in *condor_startd*, a v času komunikacije pride do napake;
7. prehod G − prehod, ko proces *condor_schedd* uspešno opravi komunikacijo s procesom *condor_startd*. Opravilo se uspešno izvaja;
8. prehod H − prehod se opravi v več primerih:
    a. v primeru da proces *condor_schedd* nima več opravil za rezerviranega izvajalca opravil,
    b. v primeru ponovne lokalne aktivnosti lastnika na izvajalcu opravil,
    c. v primeru signala za zaustavitev vseh procesov, ki so potrebni za sodelovanje izvajalca opravil v sistemu Condor,
    d. v primeru, da želi izvajalca opravil uporabljati uporabnik z višjo prioriteto,
    e. v primeru, da je izvajalec opravil primernejši za izvajanje drugih vrst opravil;
9. prehod I − prehod se opravi, če obstaja primernejše opravilo za izvajanje na izvajalcu opravil;
10. prehod J − prehod se opravi, če je bilo opravilo dlje časa ustavljeno ali zaradi zaustavitve vseh opravil na izvajalcu opravil (proces *condor_startd* mora odstraniti vse procese na vozlišču) in izvajalec opravil ni več zmožen ali pa ne želi ponujati svojih računalniških virov za izvajanje opravil v sistemu Condor;
11. prehod K − izvajalec opravil preide v stanje *lastnik*, če preneha z izvajanjem opravil izven sistema Condor (na primer opravila sistema BOINC) ali pa če je lastnik izvajalca opravil pričel z lokalnimi aktivnostmi;
12. prehod L − prehod se izvede, ko se za izvajalca opravil uspešno najde opravilo za izvajanje. Tega zažene proces *condor_schedd* na izvajalcu opravil;
13. prehod M − podobno kot prehod D.

3.2.4 Delovna okolja v sistemu Condor

Pomemben koncept v sistemu Condor so različne vrste delovnih okolij – tako imenovani univerzumi. Vsako delovno okolje (izvorno: *run-time environment*) ima svoje lastnosti in prednosti, ki so primerne za izvajanje različnih vrst opravil. Na voljo so delovna okolja local, vanilla, standard, parallel, Globus in Java.

V okolju Local universe se opravila nemudoma izvedejo neposredno na vozlišču, kjer je bilo opravilo tudi oddano. Preslikava opravila na primernega izvajalca opravil se tu ne izvaja.

Opravila, ki se izvajajo v okolju Vanilla universe nimajo na voljo shranjevanja vmesnega stanja izvajanja in tudi nimajo možnosti izvajanja oddaljenih sistemskih klicev. To pomeni, da morajo biti datoteke, ki jih opravila uporabljajo, na voljo na nekem skupnem datotečnem

sistemu (na primer z uporabo protokola NFS) ali pa mora za prenose datotek poskrbeti sistem Condor. Ker tu prenosa opravil na druge izvajalce opravil ni na voljo, se morajo opravila v primeru napak zagnati od začetnega stanja. Okolje je primerno predvsem za paketno (izvorno: *batch*) izvajanje opravil.

Delovno okolje Standard universe daje možnost shranjevanja vmesnega stanja izvajanja opravila (izvorno: *checkpointing*) in prenosa opravil iz enega izvajalca opravil na drugega (izvorno: *process migration*). Shranjevanje vmesnega stanja se lahko izvaja:

1. periodično, zaradi možnosti napak na izvajalcih opravil;
2. s posebnim ukazom;
3. v primerih prekinitve zaradi višje prioritete drugega opravila;
4. v primerih pogoste, daljše in sporadične aktivnosti lastnika izvajalca opravil.

Vse lastnosti opravila se shrani v datoteko. Shranijo se: stanje pomnilnika, registri procesne enote, stanje uporabljenih signalov in aktivnosti V/I. S shranjevanjem vmesnega stanja se izvajanje opravil na nezanesljivih izvajalcih opravil bistveno skrajša. Standardno delovno okolje omogoča tudi izvajanje oddaljenih sistemskih klicev. Če program, ki se izvaja, potrebuje del zapisa iz večje datoteke, se vsi sistemski klici (odpiranje, branje, zapisovanje in zapiranje datotek) prestrežejo in pošljejo preko omrežja kot oddaljeni klici procedur na predlagatelja opravila. Aktivnost V/I se torej opravi na vozlišču, ki je opravilo oddalo, in ne na izvajalcu opravil. Pristop zagotavlja, da ni treba vedno izvesti časovno zahtevnih prenosov večjih datotek. Uporaba standardnega okolja zahteva, da programer upošteva naslednje omejitve:

1. povezovanje programa v opravilu s posebnimi programskimi knjižnicami standardnega delovnega okolja (potreben je dostop vsaj do objektnih datotek). Običajno pa ni treba spreminjati izvorne kode programa. Komercialno dostopni programi za to delovno okolje torej niso primerni;
2. program ne sme uporabljati funkcije *fork()*, ki omogoča ustvarjanje novih procesov;
3. program ne sme biti zasnovan večnitno;
4. program ne sme uporabljati medprocesne komunikacije (izvorno: *inter process communication*).

Delovno okolje vrste *Parallel universe* omogoča izvajanje vzporednih opravil z uporabo posebnega okolja za vzporedno programiranje (na primer MPI). Vzporedna opravila so sestavljena iz programa in podatkov, pri čemer je program zasnovan s pomočjo več procesov. Vsak izmed procesov pa se lahko izvaja na različnih izvajalcih opravil. Takšno delovanje je v sistemu Condor možno le, če namestimo izvajalce opravil tako, da so na voljo izključno izvajanju te vrste opravil. Prenosi vzporednih opravil med izvajalci opravil niso možni. Določiti je treba tudi posebno namensko vozlišče, ki opravlja razvrščanje in deli opravila izvajalcem opravil.

Globus universe je delovno okolje, ki omogoča, da se opravila predajo tudi v druge vrste porazdeljenih okolij, na primer NorduGRID [78], Globus [80], UNICORE [79].

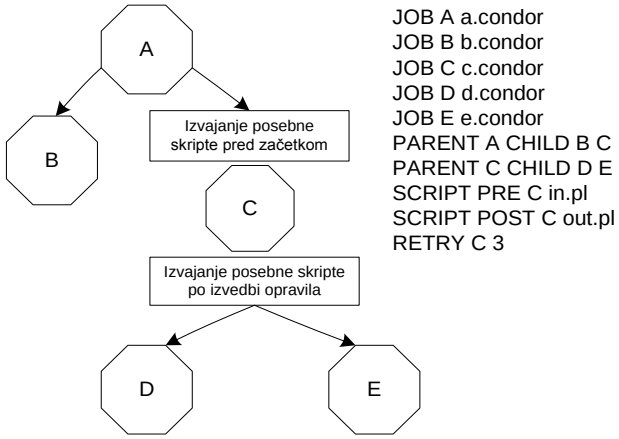
Izvajanje opravil, ki so pisana s pomočjo programskega jezika Java, se lahko izvaja v posebnem okolju *Java universe*. To pomeni, da se opravila izvajajo le na izvajalcih opravil, ki imajo nameščeno vmesno programsko opremo JVM.

3.2.5 Upravljanje z opravili s pomočjo orodij DAGMan in Master-Worker

Uporabniki v sistemu Condor pričakujejo uspešno izvajanje opravil. Zanje je sistem povsem transparenten in nimajo neposrednega dostopa do jedra sistema Condor. Pravzaprav ga niti ne potrebujejo. Na vozliščih, ki oddajajo opravila, se nad procesi, ki zagotavljajo vključenost v sistem Condor, nahaja dodatna programska plast, imenovana *problem solver*. Omogoča sledenje izvajanja opravil, aktivnih in neaktivnih izvajalcev opravil, prirejanje opravil izvajalcem opravil in ostale skrbniške aktivnosti. Uporabnik lahko posveča tudi več pozornosti reševanju problema oziroma algoritmu in ne toliko sistemu, v katerem se izvajajo opravila. Sistem Condor lahko uporablja dve dodatni programski plasti: *DAGMan* ali *Master-Worker*. Praviloma pripravita delo, primerno za izvajanje v sistemu Condor. Povezani sta neposredno s procesom *condor_schedd*, s katerim si izmenjujeta podatke. Proces *condor_schedd* namreč zagotavlja, da se delo v sistemu Condor uspešno opravi. Delo se lahko loči tudi na več manjših delov. Tak pristop je pogost predvsem pri izvajanju upodabljanja (izvorno: *rendering*) in izvajanju operacij nad različnimi parametri. Če uporabnik ne bi želel uporabljati omenjenih programskih plasti, bi moral s pomočjo programskih vmesnikov sam skrbeti za dostop do jedra sistema Condor. To pa je precej zamuden in zamotan postopek.

3.2.5.1 Orodje DAGMan

DAG je usmerjen acikličen graf vozlišč (vozlišče predstavlja opravilo), ki določa vrstni red izvajanja opravil. Vozlišča so povezana z usmerjenimi povezavami, ki določajo odvisnosti med opravili v grafu. Slika 3.6 prikazuje primer poteka izvajanja opravil. Opravili D in E se izvedeta šele, ko sta opravili A in C uspešno zaključeni. Opravili D in E se lahko izvajata hkrati, ni pa nujno.



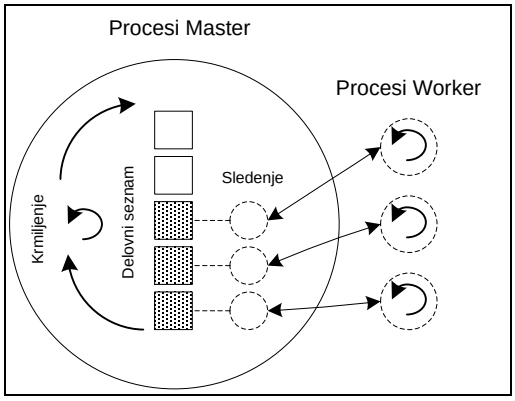
Slika 3.6: Primer poteka izvajanja opravil z orodjem DAGMan

Sistem Condor uporabniku sicer zagotavlja proste računalniške vire oziroma izvajalce opravil, ne zagotavlja pa pravilnega poteka izvajanja opravil glede na njihove medsebojne odvisnosti. V ta namen se uporablja poseben upravitelj (v principu programski proces) ***DAGMan***, ki v primeru medsebojne odvisnosti opravil zagotavlja pravilni vrstni red izvajanja. Določi ga programer s posebnimi ukazi, ki jih upravitelj DAGMan upošteva pri izvajanju (slika 3.6). Oddaja opravila v sistem Condor skladno z opisom DAG in sproti beleži napredek izvajanja. Hrani predvsem stanja preslikav med opravili in izvajalci opravil ter čase pričetka in zaključka izvajanja opravil. Če pride do napake ali nepričakovanega zrušenja upravitelja, se lahko s pomočjo beleženja brez ovir ponovno vzpostavi prejšnje stanje. Izvrševanje opravil se

prične z oddajo opravil procesu *condor_schedd*. Ena od nalog procesa *condor_schedd* je tudi zagotavljanje uspešnega izvajanja opravil. Skrbi za ponavljanje izvajanja, dokler se opravilo uspešno ne zaključi. Da bi bil upravitelj podobno zanesljiv kot ostala opravila, se tudi sam odda v izvajanje procesu *condor_schedd*. Ta obravnava upravitelja kot povsem običajno opravilo, le da se izvaja na predlagatelju opravil. Preden se opravilo prične izvajati in po njegovem zaključku, se lahko izvedejo tudi posebne ukazne datoteke. Ukazne datoteke se ne obravnavajo kot običajna opravila, ampak le kot pomožno sredstvo, ki pripravi sistem Condor za pričetek izvajanja opravil. Na primer prenos in raztegnitev datotek, ovrednotenje rezultatov po končanem opravilu in podobno. Ukazne datoteke izvede upravitelj DAGMan.

3.2.5.2 Orodje Master-Worker

Master-Worker je dodatna programska plast v obliki razredov jezika C++ za reševanje problemov poljubne velikosti na številnih nezanesljivih virih. Ko želi uporabnik rešiti računski problem, je pristop preprost. Časovno in prostorsko zahtevnejši problem razcepi na več manj kompleksnih delov – opravil, nato pa s pomočjo posebnih razredov jezika C++ zasnuje program, ki temelji na strukturi, kot je prikazano na sliki 3.7.



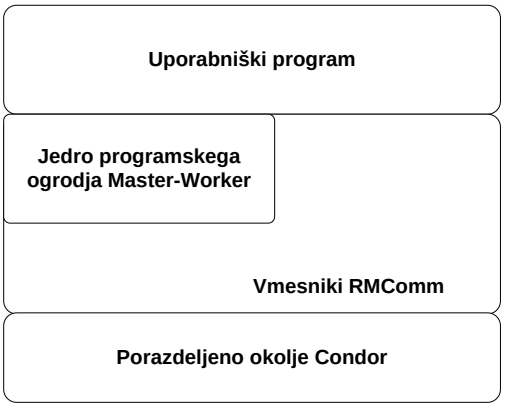
Slika 3.7: Struktura programa s pomočjo orodja Master-Worker

Program sestavlja več procesov *worker* in pa proces *master* iz treh poglavitnih delov:

1. delovnega seznama − seznama opravil, ki jih želi proces *master* izvesti v sistemu Condor;
2. modula za sledenje, ki dodeljuje opravila procesom *worker*;
3. krmilnega modula, ki upravlja in nadzira izvajanje opravil. Pregleduje rezultate, spreminja delovni seznam in opravlja komunikacijo s procesom *condor_schedd*.

Procesi *worker* odražajo razpoložljivost izvajalcev opravil v sistemu Condor in so zaradi njihove nezanesljivosti lahko zelo dinamični. Njihovo delovanje se lahko prekine takoj, ko na izvajalcih opravil pride do odpovedi ali usodne napake ali pa je izvajalca opravil pričel uporabljati lastnik. Lahko se pojavijo ponovno, ko so zopet na voljo prosti izvajalci opravil. Če proces *worker* postane neaktiven, krmilni modul poskrbi, da se opravilo, ki mu pripada, vrne na delovni seznam. Za večjo zanesljivost je na voljo tudi možnost podvajanja procesov *worker*, tako da je za eno opravilo aktivnih več različnih procesov *worker*. To pomeni, da se eno opravilo izvaja na več izvajalcih opravil v sistemu Condor. Procesi oziroma opravila se izvajajo povsem neodvisno in med seboj neposredno ne komunicirajo. Komunikacija poteka s

pomočjo protokola TCP/IP izključno s posredovanjem procesa master. Arhitektura orodja Master-Worker temelji na več plasteh (slika 3.8). Uporabniški program preko posebnih programskih vmesnikov (API) komunicira z jedrom programske plasti Master-Worker. To je povezano s programskimi vmesniki RMComm, ki zagotavljajo komunikacijo med programskim ogrodjem Master-Worker in sistemom Condor.



Slika 3.8: Arhitektura orodja Master-Worker

Vmesniki RMComm omogočajo:

1. prenos podatkov med procesom *master* in procesi *worker*;
2. zasedbo, sproščanje in spremljanje stanja izvajalcev opravil v sistemu Condor;
3. zagon opravil na izvajalcih opravil.

3.2.6 Dodeljevanje izvajalcev opravil v sistemu Condor

Izvajalci opravil, ki jih želijo uporabniki izkoriščati oziroma zasesti, so povezani z uporabniškimi prioritetami. Vsakemu izmed uporabnikov je v sistemu Condor prirejena začetna vrednost prioritete. Začetna in najvišja možna prioriteta, ki jo ima lahko uporabnik v nekem trenutku, je 0,5. Manjša vrednost pomeni višjo prioriteto in s tem večjo možnost pridobivanja večjega števila izvajalcev opravil. Če uporabnik uporablja več izvajalcev opravil, kot mu jih določa vrednost njegove prioritete, se vrednost prioritete sčasoma prične povečevati. Če pa ima uporabnik na voljo manj vozlišč, se bo sčasoma vrednost zmanjševala in uporabnik bo imel višjo prioriteto. Pri dodeljevanju vozlišč se uporabljajo posebne efektivne vrednosti prioritet, ki so enake zmnožku uporabniške prioritete in njemu pripadajoče karakteristične vrednosti. Dolgoročni cilj je, da vsi uporabniki dobijo enakomeren delež izvajalcev opravil za reševanje njihovih problemov.

Sistem Condor nima informacij o dolžini trajanja izvajanja opravil, zato ne more opravljati dodeljevanja glede na potreben čas za izvajanje. Na voljo je le informacija o številu vseh uporabnikov, ki so oddali svoja opravila v okolje, in število vseh v nekem trenutku prostih izvajalcev opravil. Glede na te informacije se lahko s pomočjo uporabniških prioritet določi število izvajalcev opravil, ki pripadajo vsakemu od uporabnikov. Ustvari se vrstni red, ki ga uporabljajo centralni upravitelj (*matchmaker*) oziroma procesa *condor_collector* in *condor_negotiator*. Proces *condor_schedd* pošlje opis opravila centralnemu upravitelju, ki nato določi vse možne izvajalce opravil, na katerih se lahko predlagano opravilo izvaja.

Pri dodeljevanju izvajalcev opravil uporabnikom lahko v trenutku vstopa uporabnika z višjo prioriteto v sistem prihaja do prekinitev opravil uporabnika z nižjo prioriteto. Trenutno izvajajoče opravilo na enem izmed zasedenih izvajalcev opravil se lahko zaradi višje prioritete drugega uporabnika ustavi, njegovo vmesno stanje shrani in po potrebi prenese na drugo delovno vozlišče. Za prekinitev izvajajočega opravila mora biti prioriteta uporabnika novega opravila vsaj za 20 % višja. Če opravilo nima določenih nikakršnih preferenc, se mu dodeli prvi prosti izvajalec opravil. Postopek je enak za vse nadaljnje uporabnike z naslednjo najvišjo prioriteto. Poleg prioritizacije uporabnikov se uporablja tudi lokalna prioritizacija opravil na izvajalcu opravil. To razvrščanje je lokalno in tako omejeno na želje oziroma omejitve lastnika izvajalca opravil. S stališča porazdeljenega sistema Condor to ni bistveno.

3.3 Primerjava in abstrakcija sistemov BOINC in Condor

Med obema sistemoma je s konceptualnega vidika velika razlika. Oba pristopa predstavljata rešitev za izkoriščanje neuporabljenih računalniških virov (procesnih ali pomnilniških). Sistem Condor je bil že od samega začetka v osnovi namenjen zaprtim okoljem znotraj ene organizacije. Vozlišča so v takšnih sistemih privzeto verodostojna in zanesljivejša, a še vedno lahko heterogena. To pomeni, da je napak malo in se jih lahko odpravlja s pomočjo vmesnega shranjevanja stanja opravil (v sklopu delovnega okolja *standard universe*). Vozlišča so lahko namenjena izključno sistemu Condor (običajno so to namenska vozlišča, organizirana v grozd) ali pa v sistemu sodelujejo le takrat, ko niso zasedena z opravili njihovih lastnikov. Sistem BOINC pa je v večji meri (vsaj v praktični uporabi) namenjen heterogenim okoljem vozlišč svetovnih razsežnosti, kjer ni popolnega zaupanja in 100 % razpoložljivosti. V sistemu Condor ni treba dodatno ovrednotiti rezultatov izvajanja poslov oziroma opravil. V sistemu BOINC pa se s pomočjo digitalnih podpisov programov lastniki projektov lahko zaščitijo pred neverodostojnimi ali napačnimi rezultati. Šifriranje je možno tudi v sistemu Condor, a se to redko uporablja.

Komunikacija pri obeh pristopih je povsem diametralna: sistem BOINC uporablja prosti pristop (metoda *pull*), pri katerem prosta vozlišča s pomočjo protokola HTTP pričnejo komunikacijo s strežnikom opravil. Protokol HTTP je za požarne zidove prehoden. Pri sistemu Condor pa moramo v požarnih zidovih poskrbeti za dodaten skrbniški poseg. Uporablja namreč pristop vsiljevanja (metoda *push*), pri kateri preko vrat protokola TCP, *Condor collector service* (številka vrat 9618) in iADT-tls (številka vrat 9614), centralni upravitelj prične komunikacijo z izvajalci opravil.

Uparjevalnik v sistemu Condor omogoča veliko prožnejše razvrščanje v primerjavi s sistemom BOINC, ki uporablja vnaprej zasnovane statične parametre v sklopu projekta. Sistem Condor ponuja celo več: uporablja namreč poseben programski jezik ClassAd za opisovanje zahtev, ki jih imajo opravila in izvajalci opravil. Uporabniške prioritete so pri postopku dodeljevanja vozlišč uporabnikom v sistemu Condor nujne, medtem ko sistem BOINC tega ali podobnega pristopa sploh ne uporablja. Pravzaprav ga tudi ne more, saj so sodelujoči prostovoljci povsem anonimni.

Oba pristopa imata na voljo operacijo shranjevanja vmesnega stanja izvajanja opravil. V sistemu BOINC za to poskrbi jedrni odjemalec oziroma program v opravilu na delovnem vozlišču. V sistemu Condor pa je takšno delovanje implicitno v sklopu standardnega delovnega okolja (izvorno: *standard universe*). Različne vrste delovnih okolij v sistemu Condor omogočajo izvajanje raznovrstnih opravil. Kot tak je veliko prilagodljivejši

uporabniškim potrebam. Izvajajo se namreč lahko opravila, ki ne potrebujejo nikakršnega povezovanja s programskimi knjižnicami (izvorno: *vanilla universe*), medtem ko je povezovanje s posebnimi knjižnicami v sistemu BOINC skorajda običajno. Izjema je morebiti le uporaba posebne programske ovojnice (izvorno: *wrapper*), ki opravilo izvaja kot ločen proces. S stališča prenosljivosti programske opreme lahko zato trdimo, da je sistem BOINC malce okornejši.

V sistemu Condor izhajamo s stališča, da so vozlišča večino časa del sistema Condor oziroma povezana centralnim upraviteljem. Vozlišča zato nova opravila od centralnega upravitelja prejmejo relativno hitro in ni potrebnih daljših čakalnih vrst opravil. V sistemu BOINC pa so dolžine čakalnih vrst na delovnih vozliščih daljše. Kot takšne so nujno potrebne, ne samo zaradi pogostega in sporadičnega izpada internetnih povezav in delovnih vozlišč, ampak tudi zaradi različnih projektov, v katerih vozlišče sodeluje. Več projektov zahteva več pomnilniških virov.

Povsem dopustno je, da sta oba sistema komplementarna. Ko v sistemu Condor ni na voljo novih opravil za izvajanje oziroma je neko vozlišče neizkoriščeno, se lahko v stanju čakaj preko posebnih vmesnikov poveže na sistem BOINC in izvaja opravila z najnižjo možno prioriteto. Ko se na vozlišče zopet vrne lastnik ali pa se vozlišču priredi novo opravilo iz sistema Condor, se vozlišče s prehodom K oziroma L (poglavje 3.2.3) zopet vrne v izvajanje opravil, ki pripadajo sistemu Condor.

Za obstoječa realna sistema BOINC in Condor lahko določimo naslednje bistvene lastnosti:

1. sistema sta sestav centralnega vozlišča in delovnih vozlišč, ki imajo omejene računalniške vire in so pripravljeni na souporabo oziroma izvajanje opravil (procesne, pomnilniške, hrambne). Povezuje jih poljubna omrežna topologija;
2. uporabniki, ki želijo izkoriščati delovna vozlišča, svoje delo razdelijo na manjše dele – posle oziroma opravila in jih oddajo na centralno vozlišče;
3. v obeh sistemih centralno vozlišče (centralni upravitelj oziroma strežnik opravil) izvaja postopek razvrščanja opravil na delovna vozlišča oziroma izvajalce opravil. Prejmejo nova neizvedena opravila s centralnega vozlišča in jih izvedejo. Rezultati se v sistemu Condor posredujejo z izvajalca na predlagatelja opravil ali na morebiti posebno rezerviran hrambni prostor (s pomočjo protokola NFS). V sistemu BOINC pa se rezultati posredujejo neposredno na strežnik opravil oziroma ločeno podatkovno bazo;
4. v obeh sistemih ima vsako izmed opravil svoj življenjski cikel, ki se zaključi z uspešnim ali neuspešnim izvajanjem;
5. opravilo je sestav programa in podatkov (podatkovne datoteke);
6. za povezovanje vozlišč se običajno uporablja zvezdna omrežna topologija.

Lastnosti, ki v realnih sistemih BOINC in Condor niso bistvene, pa so:

1. oba sistema zagotavljata zanesljivost izvajanja opravil. V sistemu BOINC je na voljo redundančno izvajanje opravil na več delovnih vozliščih hkrati. V sistemu Condor sta na voljo možnosti shranjevanja vmesnega stanja izvajanja opravila in prenosa izvajanja na drugo delovno vozlišče;
2. v sistemu Condor je na voljo dodatna programska plast, ki omogoča uporabniku prijazno izkoriščanje računalniških virov v sistemu (*Master-Worker* in *DAGMan*). V sistemu BOINC pa se dodatna plast nahaja kar v sami aplikaciji;

3. opravila in njihovi rezultati se hranijo v podatkovni bazi (Informix, MySQL);
4. sistem BOINC uporablja mehanizme proti namernim zlorabam delovnih vozlišč pri izvajanju opravil;
5. procesne enote delovnih vozlišč so lahko večjedrne;
6. možnost uporabe različnih načinov delovanja (Condor)

4 Ovrednotenje zmogljivosti prostovoljnega sistema

4.1 Postopek simulacije

V prejšnjem poglavju je podan opis dveh trenutno najbolj razširjenih prostovoljnih sistemov. Nekomu, ki se prvič srečuje s tovrstnimi sistemi, se lahko zdi delovanje na prvi pogled malce zapleteno. Jedro sistema Condor je namreč agregacija številnih programskih procesov, ki se izvajajo na centralnem upravitelju, predlagatelju opravil in izvajalcih opravil. Procesi si med seboj po vnaprej določenem protokolu pošiljajo podatke, življenjski cikel opravil pa sledi določenim korakom. Prožnost uporabe sistema je velika, saj je na voljo kar nekaj različnih delovnih okolij, izmed katerih lahko izbirajo uporabniki. Da je kompleksnost še večja, je na voljo tudi dodatna vmesna programska oprema (*DAGMan* ali *Master-Worker*), ki uporabniku omogoča prijaznejšo uporabo sistema. Podobno velja tudi za sistem BOINC. Kot celota deluje precej monolitno, vendar se v samem jedru strežnika opravil nahaja več procesov, ker je sistem namenjen skoraj izključno prostrani uporabi (v omrežju WAN) in ima varnost pomemben vpliv na delovanje sistema. Na splošno so tehnike ovrednotenja zmogljivosti takšnih sistemov ali procesov različne, po navadi pa imamo na voljo tri najpogostejše pristope: meritve, analitično modeliranje in simulacijo.

Najbolj razširjen pristop so **meritve**, ki se med drugim običajno uporabljajo tudi za preverjanje specificiranih lastnosti sistema. Meritve sistema lahko opravimo le, če ta že obstaja. Sistemov Condor in BOINC ni bilo na voljo, zato meritve v delu niso obravnavane.
Analitični model je običajno posplošen, abstrakten, matematičen opis realnega sistema. Nabor modelov, ki nam jih tak pristop nudi, je okrnjen, saj zahteva mnogo poenostavitev in predpostavk. Velikokrat dobri rezultati presenetijo celo analitike same. Ovrednotenje modelov je lahko zahtevno, časovno potratno in celo podvrženo napakam. V primeru kompleksnejših sistemov in procesov pa je lahko zaradi slabih in netočnih rezultatov tudi povsem neuporabno. Analitični model sistemov Condor in BOINC bi lahko sicer temeljil na modelu strežnih mrež, a bi bil kot tak preveč kompleksen in njegova uporabnost vprašljiva. Problem se na primer pojavi že pri abstrakciji vhodnega procesa oziroma določitvi obnašanja uporabnikov, ki pošiljajo opravila v porazdeljen sistem. Vprašljiva bi bila tudi implementacija razvrščanja opravil na vozlišča, ki ne bi temeljila na uporabi verjetnostne porazdelitve, ampak drugačnih kriterijih. Tudi povratne povezave v strežnih mrežah so lahko zamotane. V tem primeru neodvisnost vhodnega procesa od strežnega procesa ne velja več. Vrnjene zahteve namreč nosijo s seboj odvisnost od strežnega časa, kar lahko bistveno oteži analitično modeliranje. Če sistemov, ki jim želimo ovrednotiti zmogljivosti, ni na voljo ali zaradi oteženosti in časovne zahtevnosti njihova implementacija ni dopustna, potrebujemo njihovo imitacijo. Imitacijo omogoča **postopek simulacije**, ki ponazarja delovanje obravnavanega sistema. Je postopek, pri katerem zasnujemo simulacijski model in nad njim izvajamo eksperimente. S pomočjo rezultatov eksperimentov lahko sklepamo na obnašanje sistema v realnosti. Postopek simulacije omogoča ponazoritev lastnosti in delovanja obstoječega ali še neobstoječega sistema. Olajša analizo in izgradnjo sistema. Z njim lahko preverimo nove pristope in metode oziroma spremembe brez motenja ali eksperimentiranja na že obstoječem sistemu. Hitrost

izvajanja v modelu lahko povečamo ali upočasnimo in odkrivamo vzroke nepričakovanega ali problematičnega obnašanja sistema (na primer odkrivanje ozkih grl). Pomembno pa je, da poskrbimo za pravilno zasnovo simulacijskega modela in interpretacijo rezultatov. Čeprav so tudi v postopku simulacije velikokrat potrebne predpostavke in poenostavitve, se lahko bistveno bolj približamo obnašanju realnega sistema. Vsekakor je to ena od zanesljivejših metod in je bila zato izbrana za metodo ovrednotenja zmogljivosti prostovoljnega sistema.

Če bi želeli simulirati sistema BOINC in Condor tako, da upoštevamo vse njune značilnosti, komponente oziroma programske procese, ki ju definirajo, bi bil tak simulacijski model preprosto preobsežen in prezahteven [23, 65]. Zato se je treba osredotočiti le na pomembne značilnosti in zanemariti tiste, ki v postopku simulacije ne vplivajo bistveno ali vsaj ne neposredno. Na primer: podatkovno bazo sistema BOINC lahko obravnavamo povsem implicitno in je ni treba dodatno obravnavati pri postopku simulacije. S smiselnimi poenostavitvami je lahko simulacijski model gotovo veliko sprejemljivejši in obvladljiv. Določiti moramo bistvene lastnosti obeh obstoječih realnih sistemov Condor in BOINC, nato pa zasnovati simulacijski model, ki dovolj dobro imitira njune poglavitne lastnosti in obnašanje.

Z ustreznimi poenostavitvami bistvenih lastnosti realnih sistemov BOINC in Condor in z uporabo postopka simulacije želimo ugotoviti obnašanje generičnega prostovoljnega sistema oziroma ugotoviti njegove zmogljivosti. V odvisnosti od sprememb parametrov, ki simulacijski model karakterizirajo, skušamo sklepati na zmogljivost in zahteve realnih sistemov.

Če želimo izvesti simulacijo za omenjene cilje, moramo slediti določenim korakom:

1. poenostavitev realnega prostovoljnega sistema (BOINC, Condor);
2. simulacijski model kot nadaljnja poenostavitev;
3. implementacija simulacijskega modela s pomočjo orodja ns-2;
4. preverba in vrednotenje simulacijskega modela;
5. eksperimentalno delo in analiza rezultatov.

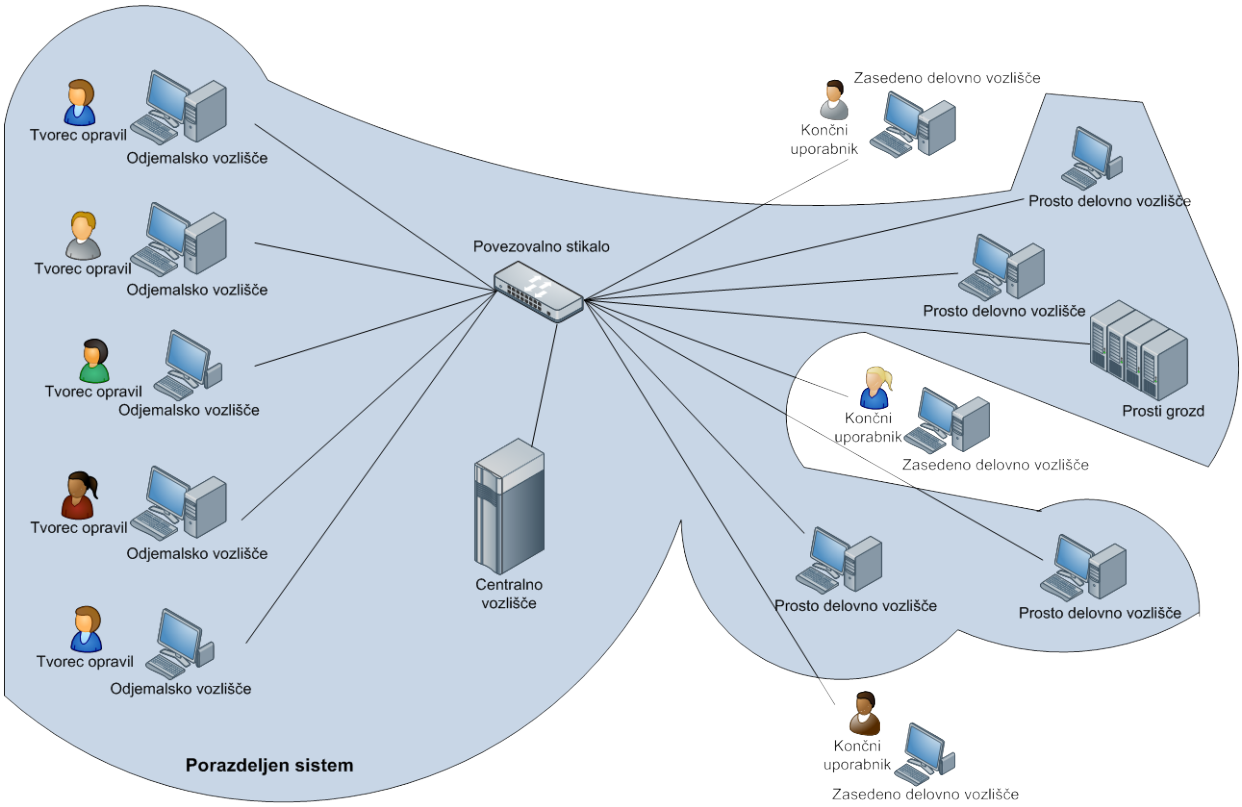
4.2 Poenostavitev realnega prostovoljnega sistema

Na podlagi bistvenih lastnosti realnih prostovoljnih sistemov BOINC in Condor lahko poenostavljeni realni prostovoljni sistem zasnujemo na sledeč način (slika 4.1):

1. sistem razdelimo na funkcijske enote (odjemalsko vozlišče, centralno vozlišče, prosta delovna vozlišča, povezovalno stikalo) in sodelujoče osebe (tvorec opravil, končni uporabnik);
2. definiramo entitete, ki prehajajo skozi sistem (opravila);
3. definiramo vloge funkcijskih enot;
4. definiramo aktivnosti oseb in diagram poteka delovanja sistema (potek entitet skozi sistem).

Tvorec opravil je oseba, ki preko odjemalskega vozlišča (običajno osebnega računalnika) in centralnega vozlišča oddaja opravila v izvajanje. Opravilo predstavlja delo (program in pripadajoče podatke), ki ga tvorec opravil želi izvesti v prostovoljnem sistemu. Centralno vozlišče je osrednji del prostovoljnega sistema. Upravlja in nadzira delovanje, tako da deluje po vnaprej določenem regularnem protokolu − s primernim uparjanjem opravil s prostimi

delovnimi vozlišči in razvrščanjem skrbi, da je prostovoljni sistem enakomerno obremenjen. Delovno vozlišče je naprava, ki se nahaja pri končnem uporabniku in je lahko v enem izmed dveh stanj: zasedena ali prosta. V času neobremenjenosti oziroma nezasedenosti je pripravljena na souporabo.

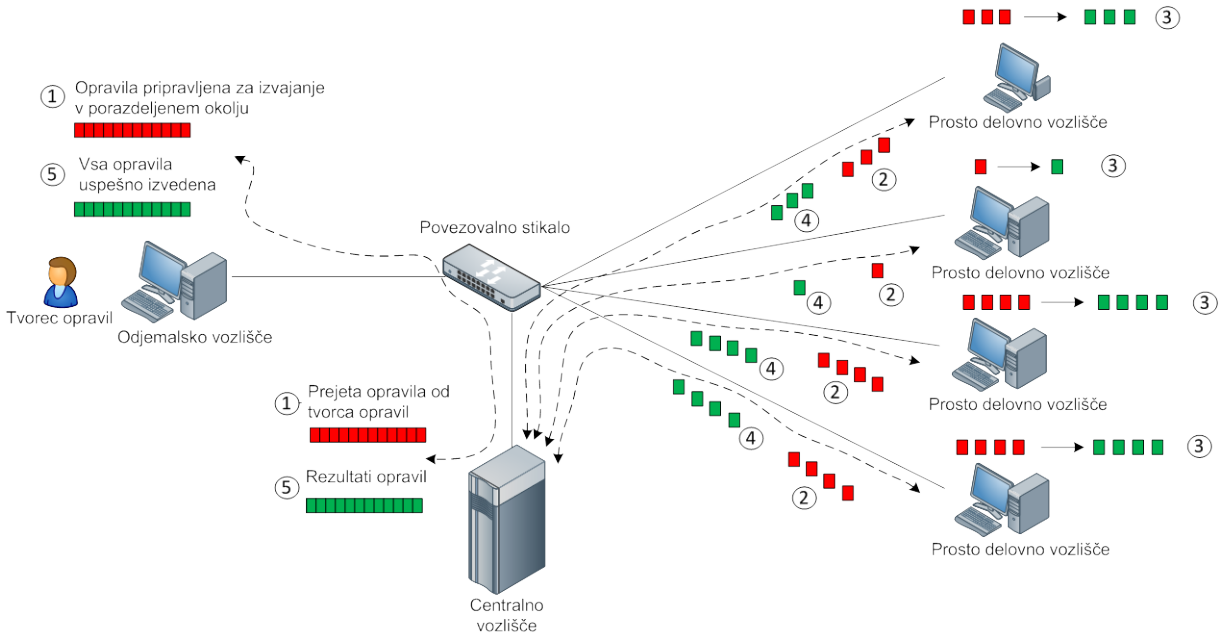


Slika 4.1: Zasnova poenostavljenega prostovoljnega sistema

Čas nezasedenosti je čas, v katerem končni uporabnik ne uporablja delovnega vozlišča oziroma delovno vozlišče ne izvaja njegovih opravil. Ko je prosto in na voljo za vključitev v prostovoljnem sistemu, izvaja opravila, ki mu jih pošilja centralni strežnik. Končne naprave so običajno namizni računalniki, igralne konzole, mobilne naprave, strežniki in grozdi. Vloga povezovalnega stikala je zagotavljanje zvezdaste omrežne topologije med vsemi vozlišči sistema.

Življenjski cikel opravila v poenostavljenem prostovoljnem sistemu poteka tako (slika 4.2):

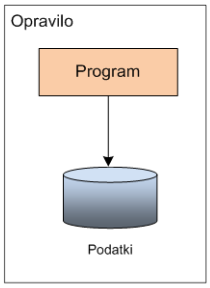
1. tvorci opravil preko odjemalske naprave in povezovalnega stikala oddajajo opravila na centralno vozlišče;
2. centralno vozlišče po vnaprej določenem protokolu (razvrščanju) odloči, katerim prostim delovnim vozliščem bo preko povezovalnega stikala poslalo opravila v izvajanje;
3. prosto delovno vozlišče prejme in prične izvajati prejeta opravila;
4. ko prosto delovno vozlišče zaključi z izvajanjem opravil, njihove rezultate preko povezovalnega stikala posreduje nazaj na centralno vozlišče;
5. ko centralno vozlišče prejme rezultate uspešno izvedenih opravil, jih preko povezovalnega stikala in odjemalskih vozlišč posreduje tvorcem opravil.



Slika 4.2: Življenjski cikel opravil v poenostavljenem porazdeljenem sistemu

Privzeta izhodišča in poenostavitve v poenostavljenem prostovoljnem sistemu so:

1. delovna vozlišča, ki izvajajo opravila končnih uporabnikov, v prostovoljnem sistemu ne sodelujejo in jih zato ne obravnavamo kot njegove gradnike;
2. opravilo je sestavljeno iz programa in podatkov (slika 4.3). Izvajanje opravila pomeni izvajanje programa nad pripadajočimi podatki;



Slika 4.3: Opravilo kot sestav progama in podatkov

3. v prostovoljnem sistemu se omejimo na podatkovno vzporedne probleme. To pomeni, da je delitev celotnega dela na opravila preprosta;
4. opravila imajo lahko različne vrste ustavljivih programov in končno dolge podatke. Časovna zahtevnost programov je identična. Opravila so zaradi različne dolžine podatkov različno zahtevna. Vedno pa zahtevajo končno mnogo procesnega časa;
5. delovno vozlišče ni omejeno na izvajanje opravil samo enega tvorca opravil, ampak lahko prejme tudi opravila drugih tvorcev opravil;
6. vsako opravilo se pošlje v samo eno od delovnih vozlišč. Centralno vozlišče nikoli ne pošilja enega opravila na več delovnih vozlišč hkrati;

7. centralno vozlišče lahko razvršča opravila na dva načina:
    a) opravilo se priredi naključnemu delovnemu vozlišču,
    b) opravilo se priredi tistemu delovnemu vozlišču, ki je najmanj obremenjeno;
8. delovna vozlišča lahko izvajajo le eno opravilo hkrati. Ko z opravilom končajo, pošljejo rezultat na centralno vozlišče in nadaljujejo z izvajanjem naslednjega opravila;
9. delovna vozlišča med seboj ne komunicirajo oziroma si ne pošiljajo podatkov. Delujejo avtonomno;
10. v prostovoljnem sistemu ne prihaja do odpovedi na kateri koli funkcijski enoti;
11. v prostovoljnem sistemu lahko prihaja do napak pri izvajanju opravil. Tako delovno vozlišče pošlje na centralno vozlišče napačne rezultate. Centralno vozlišče v tem primeru po trenutno izbranem načinu razvrščanja opravilo ponovno pošlje v izvajanje enemu od delovnih vozlišč. To ponavlja, dokler opravilo ni pravilno izvedeno;
12. delovna vozlišča so po procesnih zmogljivostih omejena. V okolju so na voljo tri vrste različno zmogljivih delovnih vozlišč, hitra, srednje hitra in počasna delovna vozlišča;
13. delovna vozlišča imajo na voljo neomejene pomnilniške zmogljivosti in lahko shranjujejo poljubno število opravil;
14. prostovoljni sistem je brezizguben, kar pomeni, da se opravila v sistemu ne izgubljajo. Kolikor opravil vstopi v prostovoljni sistem, toliko rezultatov iz prostovoljnega sistema tudi izstopi.

Tabela 2 predstavlja primerjavo med poenostavljenim prostovoljnim sistemom in obema obstoječima realnima prostovoljnima sistemoma.

Lastnost	**Poenostavljen prostovoljni sistem**	**Sistem BOINC**	**Sistem Condor**
Enota dela	Opravilo kot sestav programa in podatkov.	Opravilo, ki je sestav programa in podatkov. Delovna vozlišča se povežejo na centralno vozlišče (strežnik), ki jim nato pošlje opravila za izvajanje.	Posel oziroma opravilo, ki je sestav programa in podatkov. Program in podatki se pošiljajo na tri načine: • med delovnimi vozlišči, • preko skupnega hrambnega prostora (NFS), • s pomočjo sistemskih klicev (protokol RPC).
Shranjevanje vmesnega stanja izvajanja opravila	Ta lastnost ni bistvena in zato ni obravnavana.	Ta možnost je na voljo v sklopu programa v opravilu.	Ta možnost je na voljo le v delovnem okolju *standard universe*, izvede pa se na izvajalcu opravila.
Napake pri izvajanju opravil	Centralno vozlišče napačno izvedeno opravilo ponovno pošlje v izvajanje enemu od delovnih vozlišč (odvisno od vrste razvrščanja).	Obstaja možnost podvajanja opravil in primerjava s kanonskim rezultatom. Določa ga kvorum vseh prejetih rezultatov.	Preverjanje verodostojnosti rezultatov opravila se opravlja pri predlagatelju opravil. Podvajanje opravil je sicer možno.
Varnost	Ta lastnost ni bistvena in zato ni obravnavana.	Digitalni podpisi programov opravil so nujni ali vsaj priporočljivi. Na voljo je tudi možnost podvajanja opravil.	Možna uporaba digitalnega podpisa, a se redko uporablja, saj je sistem običajno zaprtega tipa.
Razvrščanje v porazdeljenem sistemu	2 načina: • naključno, • najkrajša čakalna vrsta najprej (parameter *LoadAvg* v sistemu Condor).	S pomočjo opisov opravil in delovnih vozlišč. Opisi so podani v obliki datotek vrste XML.	Izvaja ga centralni upravitelj, ki s pomočjo opisov ClassAd in efektivnih uporabniških prioritet uparjuje opravila in izvajalce opravil.
Lokalno razvrščanje	FIFO	FIFO. Na voljo tudi drugi pristopi.	FIFO. Na voljo tudi drugi pristopi.

Lastnost	Poenostavljen prostovoljni sistem	Sistem BOINC	Sistem Condor
Koliko opravil prejme delovno vozlišče	Delovno vozlišče lahko prejme poljubno število opravil, ki pripadajo različnim predlagateljem. Izvaja le eno opravilo hkrati.	S pomočjo opisov (datotek vrste XML) imajo delovna vozlišča vnaprej določeno število hkrati izvajajočih se opravil.	S pomočjo opisa ClassAd imajo delovna vozlišča vnaprej določeno število hkrati izvajajočih se opravil.
Odpovedi	Na delovnih vozliščih ne prihaja do odpovedi.	Do odpovedi na delovnih vozliščih lahko prihaja, vendar je s podvajanjem poskrbljeno, da se opravilo zaključi uspešno. Obstaja možnost podvajanja izvajanja opravila.	Do odpovedi na delovnih vozliščih lahko prihaja, vendar je s podvajanjem poskrbljeno, da se posel zaključi uspešno. Obstaja možnost podvajanja izvajanja opravila in prenosa izvajanja opravila na drugo delovno vozlišče.
Omejitve delovnega vozlišča	Procesni viri omejeni. Pomnilniški viri neomejeni.	Procesni viri omejeni. Pomnilniški viri omejeni.	Procesni viri omejeni. Pomnilniški viri omejeni.
Omrežna topologija	Zvezdnata	Zvezdnata	Zvezdnata

Tabela 2: Primerjava med poenostavljenim in realnim sistemom

4.3 Simulacijski model

Poenostavljeni prostovoljni sistem je treba nadalje abstrahirati v simulacijski model, ki predstavlja temelj implementacije simulacije. Definiran je s pomočjo uporabe modificiranega diskretnega stohastičnega procesa, v katerem opazujemo pretok entitet skozi funkcijske enote procesa. Za opis funkcijskih enot uporabimo klasične strežne enote M/M/1, ki jih sestavljata strežnik in čakalna vrsta. Preslikavo komponent poenostavljenega prostovoljnega sistema v komponente simulacijskega modela določa Tabela 3.

Poenostavljeni prostovoljni sistem		Simulacijski model
Porazdeljen sistem	→	Množica strežnih enot, ki so povezane v strežno mrežo.
Opravilo	→	Zahteva (klasična terminologija v teoriji vrst)
Tvorec opravil	→	Tvorec zahtev (Poissonov proces)
Odjemalsko vozlišče	→	Implicitno v vhodnem procesu
Končni uporabnik	→	Ne obravnavamo
Delovno vozlišče	→	Delovna strežna enota (sestav čakalne vrste in strežnika)
Centralni strežnik	→	Centralna strežna enota (sestav čakalne vrste in strežnika)
Povezovalno stikalo	→	Ne obravnavamo

Tabela 3: Preslikava komponent poenostavljenega prostovoljnega sistema v komponente simulacijskega modela

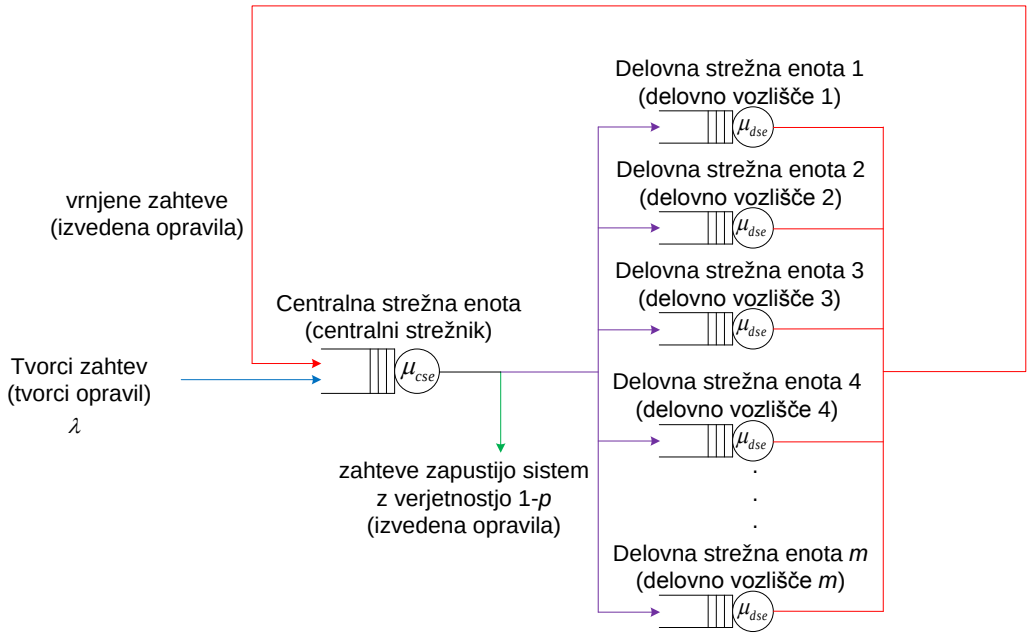
Vsak izmed tvorcev opravil v poenostavljenem prostovoljnem sistemu pošilja na centralni strežnik opravila. V simulacijskem modelu pa vsak tvorec zahtev pošilja zahteve na centralno strežno enoto po Poissonovem procesu: $\{X(t): t \geq 0\}$, kjer je $X(t)$ število zahtev, ki so prispele v centralno strežno enoto v času t [35, 38]. Proces oziroma tok dogodkov $X(t)$ je Poissonov proces z intenzivnostjo $\lambda > 0$, če:

1. velja: $X(0) = 0$;

2. je porazdelitev procesa $X(t)$ neodvisna od poljubnega časovnega zamika, število dogodkov v enem časovnem intervalu pa je neodvisno od tega, koliko dogodkov se je zgodilo v prejšnjem časovnem intervalu;
3. ima število dogodkov v katerem koli intervalu dolžine t Poissonovo porazdelitev s povprečjem λt. Za $\forall s, t > 0$ velja

$$P[X(s+t) - X(t)] = e^{-\lambda t} \frac{(\lambda t)^n}{n!}.$$

Poissonov proces je torej proces prihajanja zahtev, pri katerem je verjetnost časa med prihodoma dveh različnih zahtev porazdeljena eksponentno. Vsak tvorec opravil z intenzivnostjo λ_i po Poissonovem procesu pošilja zahteve na centralno strežno enoto. Poissonovi procesi so med seboj neodvisni in jih lahko preprosto seštejemo. To pomeni, da vse Poissonove procese, ki pripadajo posameznim tvorcem opravil, združimo v en sam Poissonov proces, ki predstavlja vse tvorce opravil. Simulacijski model prikazuje slika 4.4. V oklepajih so navedene vloge komponent v poenostavljenem prostovoljnem sistemu. Zunanje zahteve (modra barva) prihajajo v centralno strežno enoto po Poissonovem procesu. Če je čakalna vrsta centralne strežne enote prazna in strežnik ne opravlja strežbe, se zahteva takoj postreže, sicer se zahteva postavi v čakalno vrsto centralne strežne enote. Zahteva v vrsti čaka, dokler strežnik centralne strežne enote ne postane prost. Ko je prost, prevzame zahtevo v strežbo.

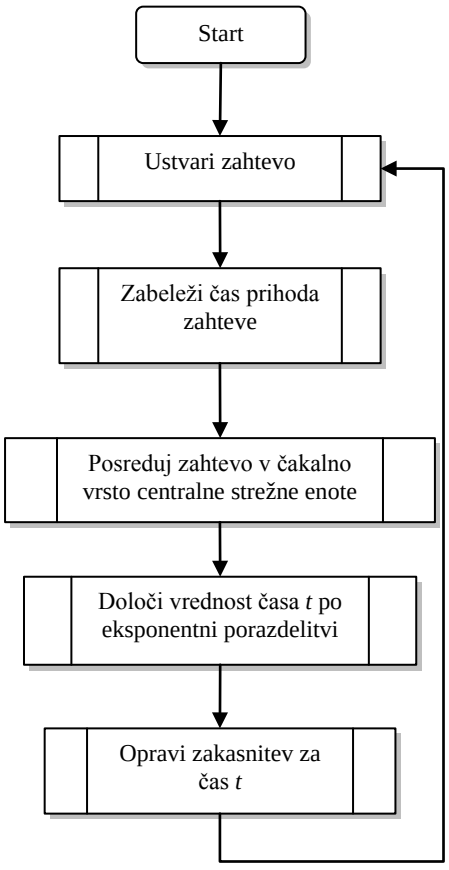


Slika 4.4: Simulacijski model poenostavljenega prostovoljnega sistema

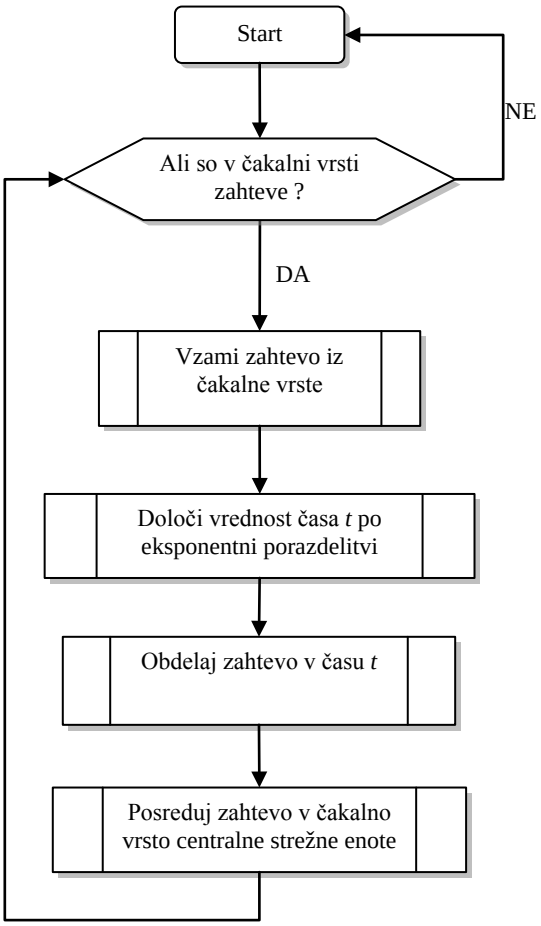
Ko v centralni strežni enoti preide zahteva iz strežbe, se vedno vsaj enkrat pošlje na eno od delovnih strežnih enot. Po vrnitvi zahteve na centralno strežno enoto ta ugotovi, ali mora biti zahteva ponovno postrežena na eni izmed delovnih strežnih enot (v primeru napake). Če je temu tako, jo na podlagi trenutno izbranega načina razvrščanja (naključno ali v najkrajšo čakalno vrsto najprej) posreduje natanko eni delovni strežni enoti. Cikel pošiljanja zahteve delovnim strežnim enotam se ponavlja, dokler zahteva z verjetnostjo $(1-p)$ ne zapusti

strežne mreže (zelena barva). V tem primeru je bila zahteva uspešno postrežena. Verjetnost p si lahko predstavljamo kot verjetnost napake pri izvajanju opravila na enem od delovnih vozlišč. Za vsa delovna vozlišča je verjetnost p enaka, kar pomeni, da lahko sleherna delovna strežna enota z verjetnostjo p povzroči ponovno pošiljanje zahteve na eno od delovnih strežnih enot. Ko delovna strežna enota prejme zahtevo (vijolična barva), jo postreže takoj, če je njena čakalna vrsta prazna in strežnik v tistem trenutku ne izvaja strežbe. Sicer se zahteva postavi v čakalno vrsto delovne strežne enote in čaka, dokler strežnik ne postane prost. Ko je prost, prevzame zahtevo v strežbo. Zahteva se nato vrne nazaj v čakalno vrsto centralne strežne enote (rdeča barva). Ta jo ponovno obdela po že omenjenem postopku. Dolžine čakalnih vrst delovnih strežnih enot in centralne strežne enote so po velikosti neomejene. To pomeni, da se zahteve v sistemu ne izgubljajo.

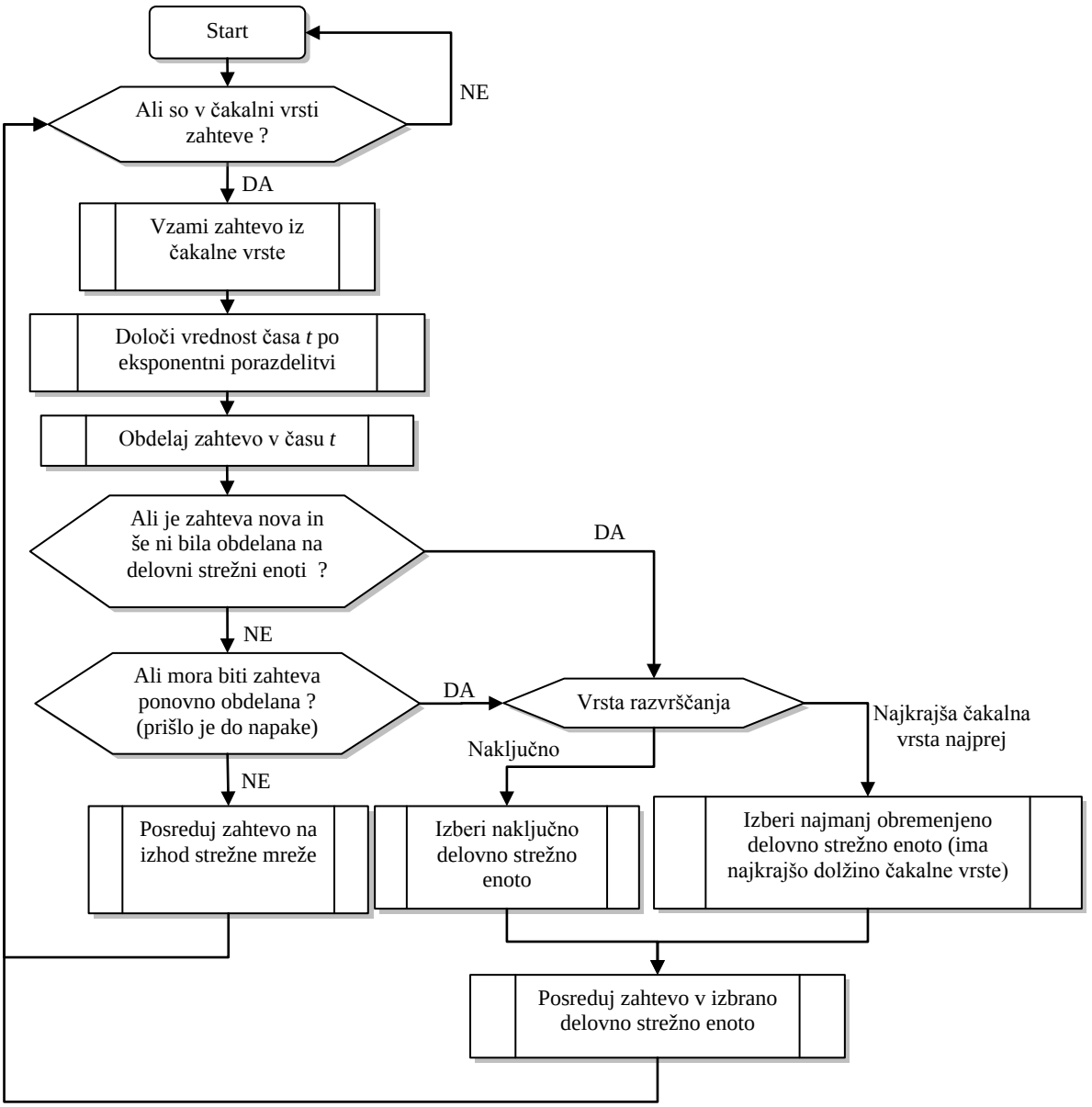
Ko zahteve vstopajo v čakalno vrsto centralne strežne enote in čakalne vrste delovnih strežnih enot, v njih ostanejo do strežbe. To pomeni, da zahteve ne zapuščajo (izvorno: *reneging*) centralne strežne enote in delovnih strežnih enot, preden se postrežejo. Prav tako zahteve ne preskakujejo (izvorno: *jockeying*) iz čakalne vrste ene delovne strežne enote v čakalno vrsto druge delovne strežne enote. Vhodni proces, proces strežbe zahtev na centralni strežni enoti in proces strežbe na delovnih strežnih enotah prikazujejo diagrami poteka – slika 4.5, slika 4.6 in slika 4.7.



Slika 4.5: Vhodni proces zahtev (Poissonov proces)



Slika 4.6: Proces streženja na delovni strežni enoti



Slika 4.7: Proces streženja na centralni strežni enoti

Formalni opisi enot simulacijskega modela so naslednji:

Centralna strežna enota:

- Kendallov zapis centralne strežne enote: (M/M/1):(FCFS/∞/∞)
- intenzivnost strežbe centralnega strežnika: μ_{cse}
- vhodna intenzivnost zahtev: λ
- tip vhodnega procesa: Poissonov proces ter proces, ki ga prispeva zanka
- tip strežbe: eksponentna porazdelitev časa strežbe
- kapaciteta čakalne vrste: ∞
- strežna disciplina: FCFS
- vrsta usmerjanja, ki ga centralna strežna enota izvaja: naključno ali najkrajša čakalna vrsta najprej

Delovna strežna enota:

- Kendallov zapis delovne strežne enote: (M/M/1):(FCFS/∞/∞)
- intenzivnost strežbe: μ_{dse}
- tip strežbe: eksponentna porazdelitev časa strežbe
- kapaciteta čakalne vrste: ∞
- strežna disciplina: FCFS

Strežna mreža:

- tip strežne mreže: enorazredni odprti model
- populacija zahtev: ∞
- število strežnih enot v mreži: 1 centralna strežna enota, *m* delovnih strežnih enot

4.4 Implementacija simulacijskega modela z orodjem ns-2

Implementacija simulacijskega modela se izvaja z diskretno simulacijo, pri kateri stanje sistema opišemo s pomočjo diskretnih spremenljivk. Dogodki so diskretni, izvajanje simulacije pa sledi kronološkemu redu. Omenjeno metodo simulacije diskretnih dogodkov uporablja orodje ns-2 [34], ki je sicer namenjeno predvsem simulacijam omrežij IP, vendar se je izkazalo kot uporabno tudi za študije drugih konceptov. Podpira različne omrežne protokole, vrste paketnega usmerjanja in povezav: *multicast*, *unicast*, žične, brezžične in satelitske povezave, splet, telnet, ftp in še mnogo drugih. Razvito je bil leta 1989 kot različica simulatorja *Real network simulator*. Orodje temelji na dualni zasnovi. Uporabljata se dva programska jezika: C++ in OTcl. Programski jezik OTcl omogoča hitro načrtovanje topologije in je primeren predvsem pri hitrih spremembah omrežnih scenarijev, jezik C++ pa omogoča natančen opis protokolov oziroma hitro izvajanje postopka simulacije. Dobra lastnost dualnosti je predvsem kompromis med enostavnim sestavljanjem omrežne topologije in hitrostjo izvajanja postopka simulacije. Slabost pa predvsem v zahtevi po poznavanju in razumevanju dveh jezikov in posledično tudi odpravljanju napak v dveh programskih jezikih. Orodje ns-2 je bilo primerno za simulacijo modela prostovoljnega sistema predvsem iz naslednjih razlogov:

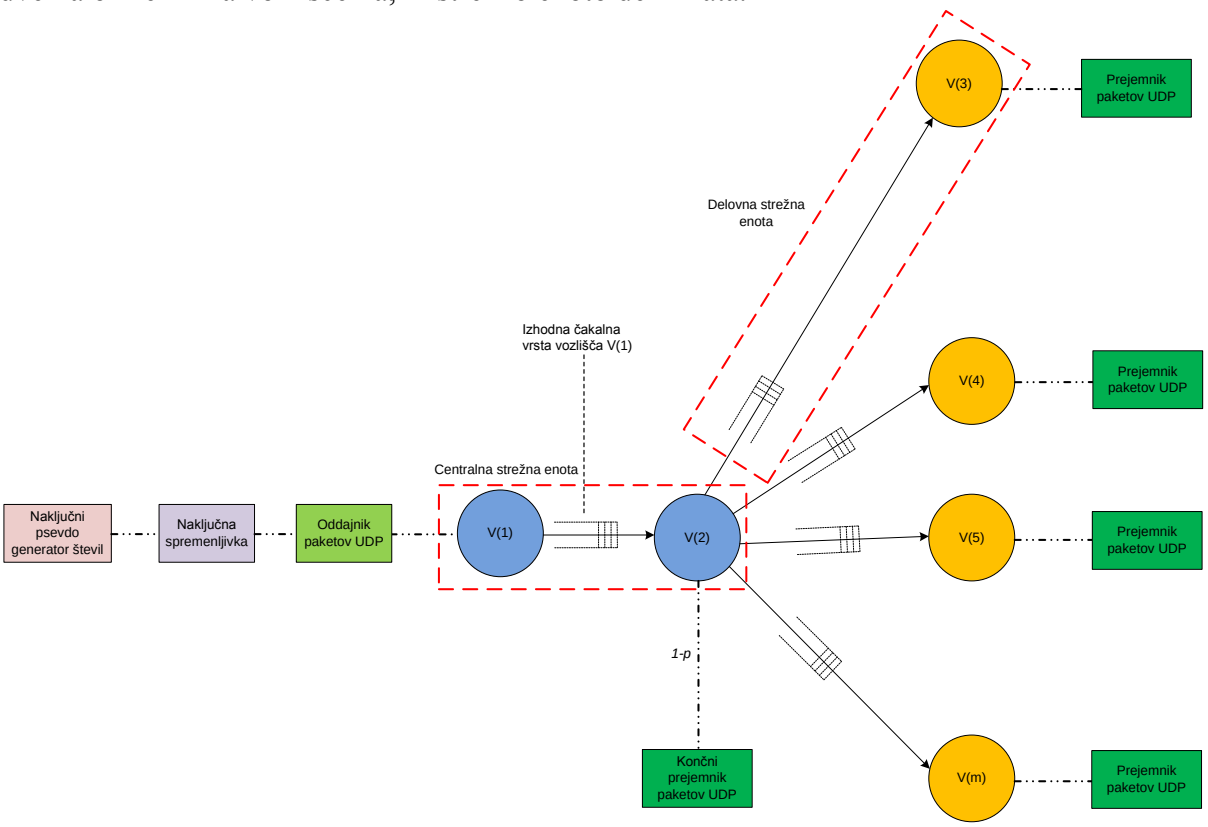
1. možnost uporabe čakalnih vrst;
2. možnost uporabe različnih vrst generatorjev prometa (naključne spremenljivke);
3. podprtost paketov najrazličnejših protokolov (vrsta in dolžina paketa);
4. dobro zasnovane datoteke za sledenje dogodkov;
5. možnost uporabe skriptnega jezika OTcl za hitro in prožno namestitev omrežne postavitve;
6. možnost uporabe animacijskega orodja NAM;
7. na voljo je dokumentacija;
8. je odprtokodno razvojno orodje.

Vsaka od komponent simulacijskega modela se mora preslikati v komponento, ki jo nudi orodje ns-2 (Tabela 4).

Simulacijski model		Orodje ns-2
Množica strežnih enot povezanih v strežno mrežo	→	Množica omrežnih vozlišč, povezanih v mrežo
Zahteva (klasična terminologija v teoriji vrst)	→	Paket protokola UDP
Tvorec zahtev	→	Kombinacija psevdonaključnega generatorja števil (L'Ecuyerjev večkratni kombinirani rekurzivni generator) in naključna spremenljivka z eksponentno porazdelitvijo. Zahteve prihajajo po Poissonovem procesu - čas med zahtevami je porazdeljen eksponentno.
Delovna strežna enota	→	Medsebojno povezan par omrežnih vozlišč
Centralna strežna enota	→	Medsebojno povezan par omrežnih vozlišč

Tabela 4: Preslikava komponent simulacijskega modela v komponente orodja ns-2

Po preslikavi dobimo omrežje komponent orodja ns-2, kot ga prikazuje slika 4.8. Omrežni vozlišči $V(1)$ in $V(2)$ ter njim pripadajoča omrežna povezava tvorijo centralno strežno enoto. Podobno velja za vse ostale omrežne povezave med omrežnim vozliščem $V(2)$ in omrežnimi vozlišči $\{V(i); 3 \leq i \leq m\}$, ki tvorijo delovne strežne enote. Vsaka izmed omrežnih povezav ima določeno prepustnost. Ko se med dvema omrežnima vozliščema vzpostavi povezava, se implicitno ustvari tudi povezavi pripadajoča izhodna čakalna vrsta. Čas strežbe zahteve v poljubni strežni enoti je zato enak času prenosa paketa UDP preko omrežne povezave med dvema omrežnima vozliščema, ki strežno enoto definirata.



Slika 4.8: Implementacija simulacijskega modela z orodjem ns-2

4.4.1 Življenjski cikel paketa UDP

Oddajnik paketov UDP s pomočjo psevdonaključnega generatorja števil in naključne spremenljivke oddaja pakete UDP po Poissonovem procesu vozlišču $V(1)$. Ko paket UDP zapusti omrežno vozlišče $V(1)$, se v primeru zasedenosti omrežne povezave postavi v izhodno čakalno vrsto omrežnega vozlišča $V(1)$. Tam čaka, da omrežna povezava postane prosta, sicer se nemudoma prične prenašati po omrežni povezavi do omrežnega vozlišča $V(2)$. V omrežnem vozlišču $V(2)$ se izvaja usmerjanje paketa UDP na eno od omrežnih vozlišč $\{V(i); 3 \leq i \leq m\}$, glede na trenutno izbrano vrsto razvrščanja. Izbere se bodisi naključno omrežno vozlišče ali omrežno vozlišče, ki pripada omrežni povezavi z najkrajšo izhodno vrsto. Paket UDP se v primeru zasedenosti izbrane omrežne povezave tudi tu postavi v izhodno čakalno vrsto omrežnega vozlišča $V(2)$. Tu čaka, da izbrana omrežna povezava postane prosta, sicer se nemudoma prične prenašati po omrežni povezavi do izbranega omrežnega vozlišča $\{V(i); 3 \leq i \leq m\}$. Ko izbrano omrežno vozlišče prejme paket UDP, ga preda prejemniku paketov UDP. Slednji povzroči, da se enak paket UDP pošlje na vhod vozlišča $V(1)$, trenutni paket UDP pa se zavrže. Za ponovno pošiljanje paketa UDP na eno izmed omrežnih vozlišč $\{V(i); 3 \leq i \leq m\}$, se vedno odloča z verjetnostjo p. Postopek se ponavlja, dokler z verjetnostjo $(1 - p)$ paket UDP ni usmerjen h končnemu prejemniku paketov UDP, ki paket UDP dokončno zavrže. Topologija omrežne zasnove je bila zasnovana s pomočjo jezika OTcl, gradniki pa s pomočjo jezika C++.

4.4.2 Opis parametrov implementacije

Parametri, ki so pri simulacijskem postopku oziroma eksperimentih nastavljivi, so sledeči:

1. vhodna intenzivnost zahtev: λ;
2. intenzivnost strežbe centralne strežne enote: μ_{cse};
3. intenzivnost strežbe delovnih strežnih enot: μ_{dse_i};
4. verjetnost vračanja paketa UDP: p;
5. tip razvrščanja paketov UDP na vozlišču $V(2)$. Možna sta dva pristopa:
    a. naključno,
    b. v najkrajšo čakalno vrsto najprej. V trenutku razvrščanja se opravi pregled dolžin čakalnih vrst vseh delovnih strežnih enot od najhitrejših pa vse do najpočasnejših povezav med vozliščem $V(2)$ in vozlišči $\{V(i); 3 \leq i \leq m\}$;
6. hitrost omrežne povezave $V(1) \rightarrow V(2)$. Privzeta vrednost je 1Mbps;
7. število delovnih strežnih enot: m. Vsaka izmed delovnih strežnih enot je lahko hitra, srednje hitra ali počasna;
8. zakasnitev povezave. Določena je privzeta vrednost $1\,ms$ (zgolj zaradi možnosti vpogleda v animacijo);
9. dolžina izhodne čakalne vrste omrežnega vozlišča $V(1)$ in $V(2)$. Čakalne vrste so privzeto neskončno dolge.

Veljati mora

$$x_{cse} = \frac{1}{\mu_{cse}} = \frac{dolžina\ podatkovnega\ paketa\ UDP}{hitrost\ omrežne\ povezave_{V(1)\rightarrow V(2)}}. \quad (1)$$

Pri čemer so:

- x_{cse} − čas strežbe na centralni strežni enoti oziroma čas prenosa paketa UDP preko povezave med vozliščema, ki tvorita centralno strežno enoto
- μ_{cse} − intenzivnost strežbe na centralni strežni enoti

Hitrost omrežne povezave $V(1) \rightarrow V(2)$ naj bo enaka konstantni vrednosti (na primer 1 Mbps). S podano intenzivnostjo procesiranja zahtev na centralnem strežniku μ_{cse}, lahko določimo potrebno povprečno dolžino paketa UDP, ki se prenaša na omrežni povezavi $V(1) \rightarrow V(2)$. Iz enačbe (1) lahko določimo povprečno dolžino paketa, ki ga oddaja oddajnik paketov UDP po Poissonu. Dobimo izraz

$$dolžina\ podatkovnega\ paketa\ UDP = \frac{hitrost\ omrežne\ povezave_{V(1)\rightarrow V(2)}}{\mu_{cse}}.$$

Glede na podano intenzivnost strežbe na centralni strežni enoti μ_{cse} in podanimi intenzivnostmi streženja delovnih strežnih enot μ_{dse_i}, lahko izračunamo razmerje med hitrostjo streženja na centralni strežni enoti in hitrostjo streženja na delovni strežni enoti i. Zato velja

$$k_i = \frac{\mu_{cse}}{\mu_{dse_i}}\ ; 3 \leq i \leq m\ . \tag{2}$$

Če velja enačba (2), velja tudi enačba

$$hitrost\ omrežne\ povezave_{V(1)\rightarrow V(2)} = k_i \cdot hitrost\ omrežne\ povezave_{V(2)\rightarrow V(i)}\ ; \\ 3 \leq i \leq m\ .$$

Potem hitrost omrežne povezave $V(2) \rightarrow V(i); 3 \leq i \leq m$ definiramo z izrazom

$$hitrost\ omrežne\ povezave_{V(2)\rightarrow V(i)} = \frac{hitrost\ omrežne\ povezave_{V(1)\rightarrow V(2)}}{k_i} =$$

$$\frac{hitrost\ omrežne\ povezave_{V(1)\rightarrow V(2)} \cdot \mu_{dse_i}}{\mu_{cse}};\ 3 \leq i \leq m\ .$$

Dolžine paketov UDP so porazdeljene eksponentno, kar zagotavlja, da je verjetnost časa obdelave zahteve na centralni strežni enoti in delovnih strežnih enotah prav tako porazdeljena eksponentno.

4.5 Preverba in vrednotenje simulacijskega modela

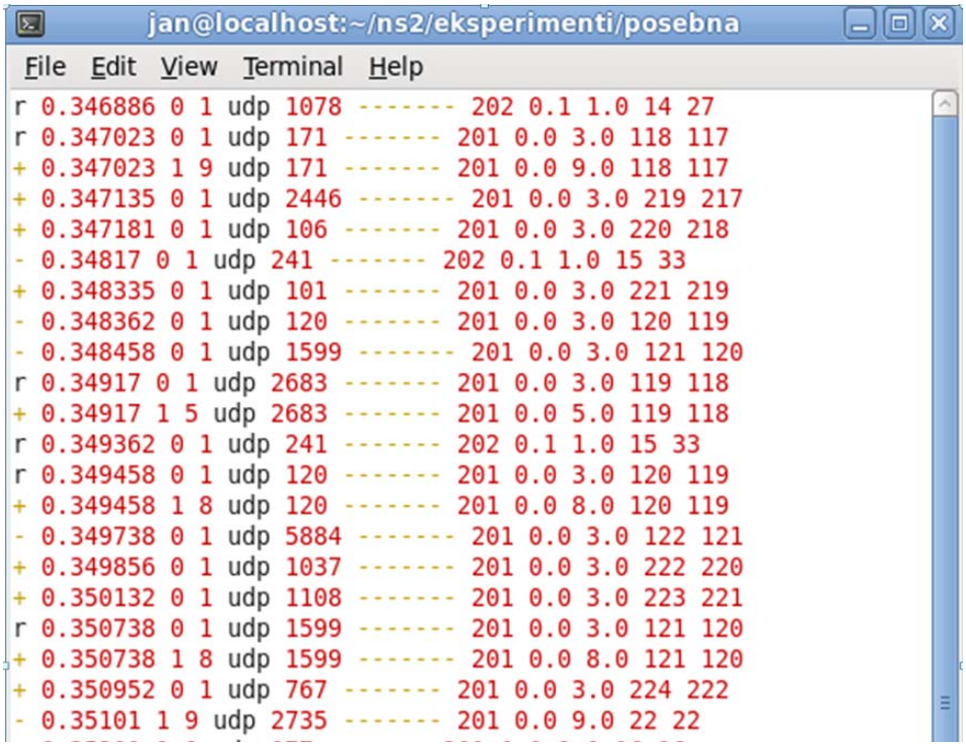
Ugotavljanje pravilne zasnove in implementacije simulacijskega modela je ključno, sicer so lahko rezultati postopka simulacije povsem neuporabni. Nič nam namreč ne koristi, če implementacija brezhibno ustreza simulacijskemu modelu, predpostavke pa so povsem napačne. In obratno, ko so predpostavke sicer pravilne, lahko napake pri implementaciji (na primer programski hrošči) ustvarijo povsem zavajajoče in napačne rezultate. Ugotoviti je

treba, v kakšni meri izhod simulacije ustreza izhodu dejanskega realnega sistema. Obstajata dva pristopa: preverba in vrednotenje [10, 35]. Pri preverbi ugotavljamo, ali je implementacija simulacijskega modela pravilna. Pri postopku vrednotenja pa skušamo ugotoviti, ali imitacija realnega sistema ustreza našim pričakovanjem oziroma so privzete predpostavke in poenostavitve pri zasnovi simulacijskega modela pravilne.

4.5.1 Preverba implementacije simulacijskega modela

Preverba implementacije je potekala sproti, med samim razvojem programske kode, in s posebno datoteko za pregled vseh dogodkov v omrežju (slika 4.9). V vsaki vrstici tabele je mogoče razbrati:

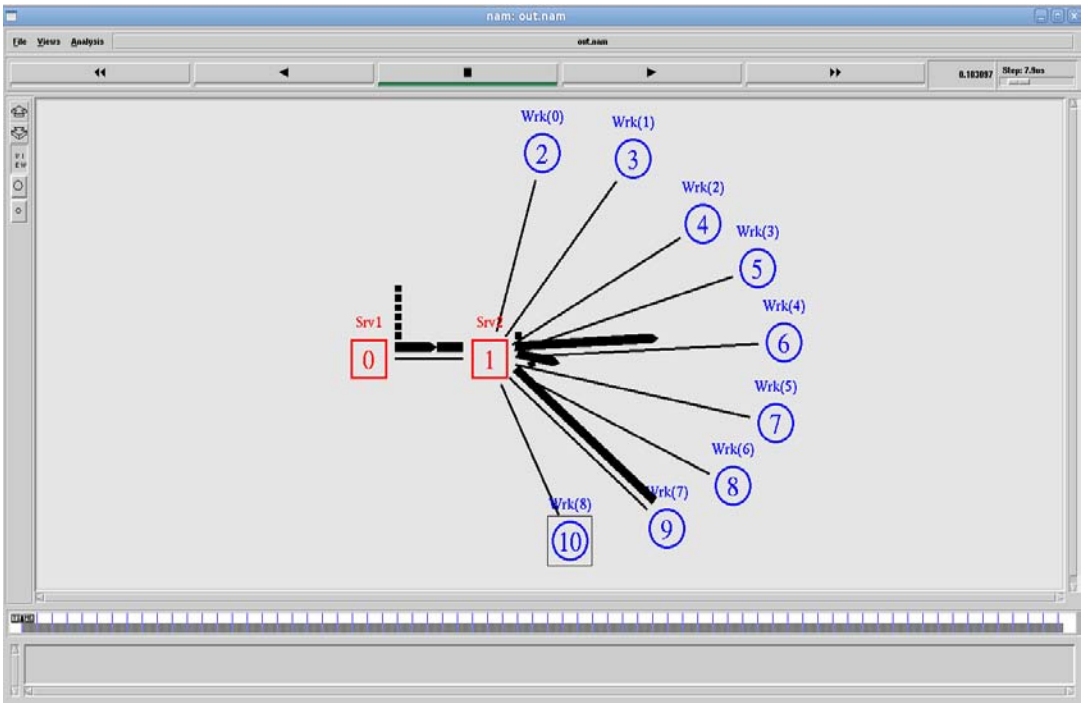
1. do kakšnega dogodka je prišlo v navedenem času;
2. izvorno in ponorno omrežno vozlišče;
3. vrsto in dolžino paketa UDP, ki se prenaša po povezavi med omrežnima vozliščema;
4. izvorni in ponorni naslov vrat v omrežnem vozlišču (vrsta prejemnika podatkovnega paketa UDP);
5. identifikacijsko številko paketa.



Slika 4.9: Vsebina datoteke sledenja pri izvajanju simulacije v orodju ns-2

V veliko pomoč je bilo tudi posebno animacijsko orodje NAM, ki je omogočalo vizualno sledenje vseh dogodkov v času izvajanja simulacije z orodjem ns-2 (slika 4.10). Preverba se je izvedla tudi s pomočjo preprostega deterministčnega vhoda. Na vhodu je bil aktiven konstanten vhodni proces paketov UDP s konstantno dolžino in hitrostjo prenosa (1 paket/s). Razvrščanje je bilo statično, kar pomeni, da je bila pot slehernega paketa UDP vnaprej določena. Izbrano je bilo vozlišče $V(3)$, verjetnost ponovnega pošiljanja paketa p, pa je bila enaka 0. Izvedba simulacije je bila pričakovana: sleherni paket UDP se je prenesel natanko dvakrat preko omrežne povezave $V(1) \rightarrow V(2)$ in natanko enkrat preko omrežne povezave

$V(2) \rightarrow V(3)$. Paket UDP se je po prihodu na končnega prejemnika paketov UDP tudi ustrezno uničil. To pomeni, da se v simulacijskem modelu paketi UDP ne zadržujejo neskončno dolgo oziroma model zapustijo v končnem času.



Slika 4.10: Orodje NAM kot del simulacijskega orodja ns-2

4.5.2 Vrednotenje simulacijskega modela

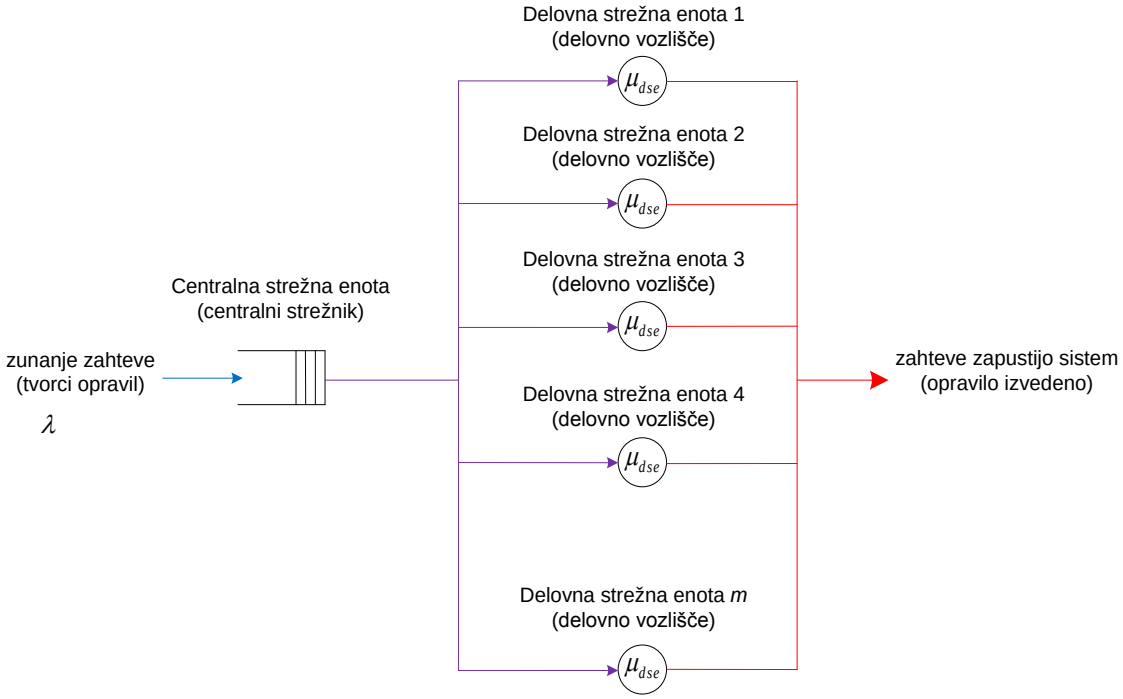
Daleč najzanesljivejša metoda vrednotenja je primerjava izhodnih rezultatov, ki jih zagotavlja realni sistem, z rezultati, ki jih dobimo z izvajanjem simulacije. Ker realnega sistema (BOINC in Condor) ni bilo na voljo, smo vrednotenje potrdili s pomočjo poenostavljenega analitičnega modela. Glede na definicijo in vse predpostavke obravnavajmo poenostavljen prostovoljni sistem kot črno komponento. Zavedamo se lahko sledečih lastnosti te komponente:

1. vanjo vstopajo opravila in jo po določenem času zapustijo;
2. v njej se nahaja več delovnih vozlišč, ki izvajajo opravila;
3. je brezizguben sistem in je zato komponenta zmožna pomnjenja.

Navedeno nam dopušča, da poenostavljen prostovoljni sistem obravnavamo kot strežno enoto: (M/M/m):(FCFS/∞/∞), kot jo prikazuje slika 4.11. V teoriji vrst je to že znan matematični model in je podoben simulacijskemu. Za opis zopet uporabimo klasično strežno enoto, ki jo sestavljajo strežniki in čakalna vrsta. Poenostavitve so sledeče:

1. delovne strežne enote se poenostavijo v strežnike strežne enote (M/M/m):(FCFS/∞/∞);
2. centralni strežnik se poenostavi v čakalno vrsto, ki sprejema vhodne zahteve (modra barva). Če so strežniki zasedeni, zahteve čakajo v vrsti. Ko postane en od strežnikov prost, takoj sprejme zahtevo iz čakalne vrste (vijolična barva);

3. predpostavimo, da do napak pri izvajanju opravil ne prihaja, zato povratne zanke tu ne uporabimo. Ko se zahteva obdela na enem od prostih strežnikov, nemudoma zapusti sistem (rdeča barva).



Slika 4.11: Nadaljnja poenostavitev simulacijskega modela
v strežno enoto vrste M/M/m

Formalni opis enot analitičnega modela je naslednji:

Centralna strežna enota:
- centralni strežnik je neskončno dolga čakalna vrsta
- vhodna intenzivnost zahtev v čakalno vrsto: λ
- tip vhodnega procesa: Poissonov proces brez prisotnosti zanke

Delovna strežna enota:
- ima vlogo strežnika strežne enote (M/M/m):(FCFS/∞/∞)
- intenzivnost strežbe strežnika: μ_{dse}
- tip strežbe: eksponentna porazdelitev časa strežbe

Strežna mreža:
- tip strežne mreže: enorazredni odprti model
- število strežnih enot v mreži: strežna enota (M/M/m):(FCFS/∞/∞) predstavlja celotno strežno mrežo

Čas zadrževanja zahteve v strežni enoti M/M/m [38] je podan z enačbo

$$T = \frac{1}{\mu}\left(1 + \frac{\vartheta}{m(1-\rho)}\right). \qquad (3)$$

Pri tem so:

$$\vartheta = p(v\ sistemu\ \geq m\ zahtev) = \frac{(m\rho)^m}{m!(1-\rho)} p_0\ ,$$

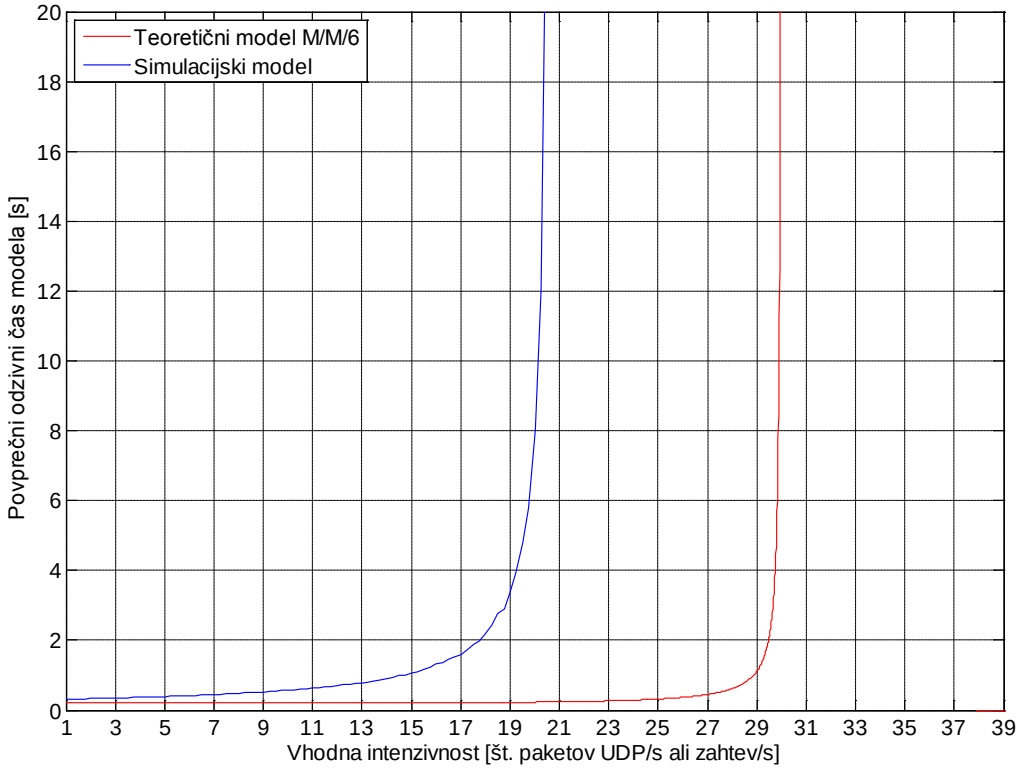
$$p_0 = \left[1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}\right]^{-1}$$ – verjetnost prazne strežne enote,

μ – intenzivnost streženja zahtev,

m – število strežnikov strežne enote,

$\rho = \frac{\lambda}{m\mu}$ – izkoriščenost strežnikov ($\rho \leq 1$).

Grafično primerjavo časa zadrževanja pri obeh pristopih prikazuje slika 4.12. Oblika obeh funkcij je zelo podobna. Odstopanja prispevajo predvsem čakalne vrste delovnih strežnih enot in dodatna zanka simulacijskega modela. Privzete parametre simulacijskega modela in vrednosti parametrov enačbe (3) določa Tabela 5. Potek pri obeh pristopih je podoben, zato lahko sklepamo, da je vrednotenje simulacijskega modela uspešno.



Slika 4.12: Odzivni čas poenostavljenega analitičnega in simulacijskega modela prostovoljnega sistema

#	Parameter	Simulacijski model	Analitični model
1	p – verjetnost vračanja zahtev	0,3	/
2	μ_{cse} – intenzivnost streženja centralnega strežnika	500 paketov UDP/s	Centralni strežnik je nadomeščen s čakalno vrsto
3	m – število delovnih vozlišč	6	Delovna vozlišča so nadomeščena s strežniki strežne enote (M/M/6):(FCFS/∞/∞)
4	μ_{dseh} – intenzivnost strežbe hitrih delovnih vozlišč	5 paketov UDP /s	Strežniki enote (M/M/6):(FCFS/∞/∞) imajo intenzivnost streženja $\mu = 5$ zahtev/s
5	μ_{dsesh} - intenzivnost strežbe srednje hitrih delovnih vozlišč	5 paketov UDP /s	
6	μ_{dsep} - intenzivnost strežbe počasnih delovnih vozlišč	5 paketov UDP /s	
7	n_{dseh} - število hitrih delovnih vozlišč	2	Vrstica 3 v Tabela 5
8	n_{dsesh} - število srednje hitrih delovnih vozlišč	2	
9	n_{dsep} - število počasnih delovnih vozlišč	2	
10	λ – vhodna intenzivnost zahtev	0 do 160 paketov UDP/s.	0 do 160 zahtev/s
11	Trajanje simulacije	20000 simulacijskih sekund	/
12	Velikost čakalnih vrst	Vse čakalne vrste so neskončno dolge.	Čakalna vrsta je neskončno dolga
13	Tip razvrščanja	Naključno	Prvi prosti strežnik

Tabela 5: Vrednosti parametrov simulacijskega in analitičnega modela za medsebojno primerjavo odzivnega časa

4.6 Eksperimentalno delo in analiza rezultatov simulacije

Pri eksperimentalnem delu želimo ugotoviti, kako spremembe parametrov simulacijskega modela vplivajo na tipične izhodne parametre strežne mreže [35, 38, 45, 46]:

1. odzivni čas strežne enote oziroma povprečni čas zadrževanja paketov UDP v strežni mreži (kako hitro porazdeljeno okolje izvede opravila);
2. dolžine čakalnih vrst delovnih strežnih enot in centralne strežne enote (kakšne pomnilniške kapacitete je treba predvideti na delovnih vozliščih);
3. interval uporabnosti strežne mreže (kdaj je porazdeljeno okolje še uporabno);
4. prepustnost (kakšen je pretok opravil skozi porazdeljeno okolje).

Označimo čas vstopa paketa UDP i v strežno mrežo s $t_{i_{vhod}}$ in čas, ko jo zapusti s $t_{i_{izhod}}$, potem je čas zadrževanja paketa UDP i v strežni mreži enak

$$t_i = t_{i_{izhod}} - t_{i_{vhod}}.$$

Če je N število vseh paketov UDP, ki so v času izvajanja simulacije zapustili strežno mrežo, potem enačba (4) določa povprečni čas zadrževanja paketov UDP v strežni mreži oziroma odzivni čas strežne mreže pri podanih parametrih.

$$\overline{T} = \frac{1}{N}\sum_{i=1}^{N} t_i \tag{4}$$

Dolžino čakalnih vrst centralne strežne enote in delovnih strežnih enot določa trenutno število paketov UDP, ki se nahajajo v čakalni vrsti in čakajo na strežbo. Povprečno dolžino čakalne vrste določa povprečno število paketov UDP v čakalni vrsti v celotnem obdobju simulacije. Interval uporabnosti strežne mreže je interval vhodnih intenzivnosti, pri katerih je strežna mreža še uporabna. Prepustnost strežne mreže je določena kot razmerje med številom vseh paketov UDP, ki so uspešno zapustili strežno mrežo, in časom simulacije.

Zanimajo nas izhodni parametri pri spremembah karakterističnih parametrov simulacijskega modela oziroma strežne mreže. Tabela 6 podaja spremembe parametrov realnega prostovoljnega sistema in k temu pripadajoče spremembe simulacijskega modela, ki utegnejo vplivati na obnašanje modela oziroma njene izhodne parametre.

	Spremembe v realnem prostovoljnem sistemu	**Spremembe v simulacijskem modelu**
1	Uporabniki v prostovoljni sistem oddajajo različno zahtevna opravila z različno intenzivnostjo.	V strežno mrežo vstopajo pri različnih povprečnih vrednostih Poissonovega procesa različno dolgi paketi UDP.
2	Število prostih delovnih vozlišč v prostovoljnem sistemu je lahko manjše ali večje.	V strežni mreži je na voljo več ali manj omrežnih vozlišč, ki tvorijo delovne strežne enote.
3	Intenzivnost strežbe na prostih delovnih vozliščih in centralnem vozlišču je lahko večja ali manjša. Centralno vozlišče in delovna vozlišča so bodisi bolj zmogljiva in postrežejo več zahtev v časovni enoti ali manj zmogljiva in postrežejo manj zahtev v časovni enoti.	Omrežne povezave med omrežnimi vozlišči modela imajo lahko večjo prepustnost in prenesejo več podatkov ali manjšo prepustnost in prenesejo manj podatkov v enoti časa.
4	Centralno vozlišče uporablja različne vrste razvrščanja opravil na delovna vozlišča.	Vozlišče $V(2)$ usmerja pakete UDP naključno ali glede na zasedenost izhodnih čakalnih vrst povezav.
5	Pri izvajanju opravil na prostih delovnih vozliščih lahko prihaja do napak. Opravilo se mora ponovno poslati na prosto delovno vozlišče.	Vozlišče $V(2)$ v odvisnosti od verjetnosti p ponovno pošlje paket UDP na povezavo, ki jo določa tip razvrščanja.

Tabela 6: Spremembe realnega prostovoljnega sistema in simulacijskega modela, ki utegnejo vplivati na izhodne parametre

Pri eksperimentalnem delu so se upoštevali vhodni in izhodni parametri z oznakami in opisi, ki jih povzema Tabela 7.

Vhodni parametri strežne mreže	
Oznaka vhodnega parametra	*Pomen vhodnega parametra*
λ	Vhodna intenzivnost paketov UDP na vhodu strežne mreže (število paketov UDP/s)
μ_{cse}	Intenzivnost streženja centralne strežne enote (število paketov UDP/s)
μ_{dseh}	Intenzivnost strežbe hitrih delovnih strežnih enot (število paketov UDP/s)
μ_{dsesh}	Intenzivnost strežbe srednje hitrih delovnih strežnih enot (število paketov UDP/s)
μ_{dsep}	Intenzivnost strežbe počasnih delovnih strežnih enot (število paketov UDP/s)
p	Verjetnost vračanja paketa UDP na vhod strežne mreže
Razvrščanje na centralni strežni enoti (cse)	Tip razvrščanja paketov UDP na centralnem strežni enoti (možni dve vrsti razvrščanja *naključno* (parameter je enak vrednosti 0) in *najkrajša čakalna vrsta najprej* (parameter je enak vrednosti 1)
s_{cse}	Hitrost omrežne povezave $V(1) \rightarrow V(2)$
τ	Zakasnitev povezav – zaradi vizualnega opazovanja izvajanja simulacije (orodje NAM)
n_{dseh}	Število hitrih delovnih strežnih enot
n_{dsesh}	Število srednje hitrih delovnih strežnih enot
n_{dsep}	Število počasnih delovnih strežnih enot
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	Število vseh delovnih strežnih enot
$L_{čv}$	Največja dolžina čakalne vrste delovnih strežnih enot in centralne strežne enote
Izhodni parametri strežne mreže	
Oznaka izhodnega parametra	*Pomen izhodnega parametra*
$\overline{T}$	Povprečni odzivni čas strežne mreže
$\overline{L_{čv_i}}$ in $\overline{L_{cse}}$	Povprečna dolžina čakalne vrste i-te delovne strežne enote in centralne strežne enote
Interval uporabnosti	Interval vhodnih intenzivnosti, pri katerih je strežna mreža uporabna in ni v nasičenju
X	Prepustnost strežne mreže

Tabela 7: Vhodni in izhodni parametri, pomembni za eksperimentalno delo

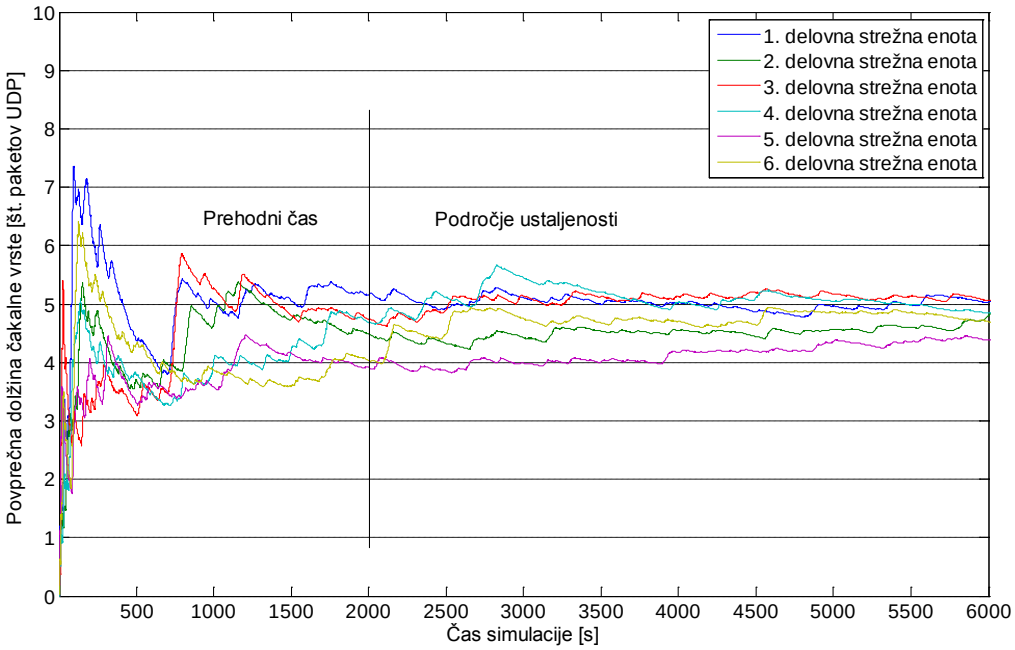
4.6.1 Določitev časa izvajanja simulacije

Simulacija običajno zahteva izhodišča, ki zagotavljajo verodostojno interpretacijo rezultatov postopka simulacije. Izvajanje simulacije mora biti časovno omejeno. Treba je določiti še sprejemljivo mejo časa izvajanja simulacije. V tem času je treba upoštevati stanja, ki so stabilna. Čakalne vrste centralne strežne enote in delovnih strežnih enot so tik pred začetkom simulacije prazne. Kot takšne vnašajo v končne rezultate določene napake. Vpliv napak se zmanjša, če se postopek simulacije izvaja dovolj dolgo. Začetne nestabilnosti potem ne pridejo do izraza oziroma je njihov vpliv zanemarljiv. Problematična pri tem pa je določitev časa trajanja simulacije, v katerem nestabilnosti nimajo več bistvenega vpliva na končne rezultate simulacije. Določiti je treba dvoje: čas izvajanja simulacije posameznega eksperimenta in čas, ki je potreben za doseganje stabilnega stanja strežne mreže. Le stabilno stanje lahko poda pravilne rezultate eksperimentov. Stanje, v katerem čakalne vrste centralnega strežne enote in čakalne vrste delovnih strežnih enot niso stabilne, imenujemo prehodni čas. Rezultatov v tem času pri analizi rezultatov ne upoštevamo in jih zanemarimo. Privzemimo vrednosti vhodnih parametrov simulacijskega modela, kot jih prikazuje Tabela 8.

Parameter strežne mreže	Vrednost parametra
λ	10
μ_{cse}	500
μ_{dseh}	5
μ_{dsesh}	5
μ_{dsep}	5
p	0,6
Razvrščanje na centralni strežni enoti	0
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	2
n_{dsesh}	2
n_{dsep}	2
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	6
$L_{čv}$	∞

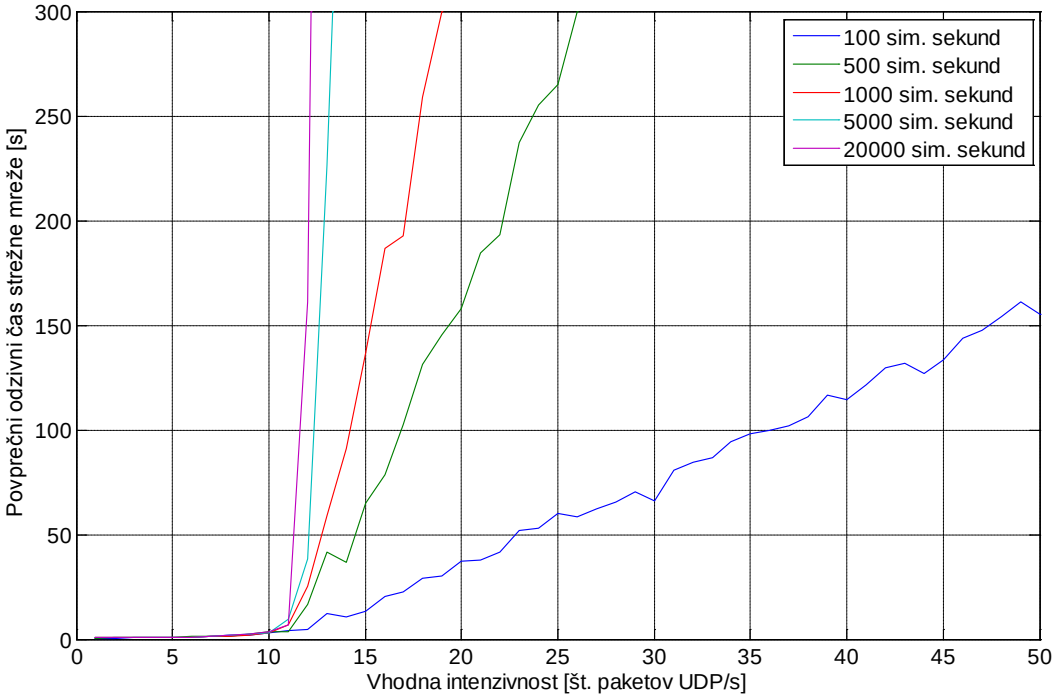
Tabela 8: Vrednosti parametrov simulacijskega modela za določanje prehodnega obdobja

Pri tako izbranih parametrih izvedemo postopek simulacije in opazujemo povprečno dolžino čakalnih vrst delovnih strežnih enot. S pomočjo rezultatov lahko nato ugotovimo, koliko časa je potrebno, da se čakalne vrste delovnih vozlišč ustalijo. Seveda je izbrana vhodna intenzivnost takšna, da je celotna strežna mreža v povprečju vedno stabilna. V nasprotnem primeru lahko čakalne vrste strežnih enot narastejo preko vseh meja, s tem pa bi bila določitev prehodnega časa onemogočena. Ugotovljeni prehodni čas in področje ustaljenosti prikazuje slika 4.13.



Slika 4.13: Določitev prehodnega časa in področja ustaljenosti pri simulaciji

Upoštevajmo parametre, ki jih določa Tabela 8, in spremenimo vhodno intenzivnost λ tako, da zavzame vrednosti 0 – 50 paketov UDP/s. Izvedemo postopke simulacije, ki trajajo različno dolgo. Potek povprečnega odzivnega časa strežne mreže prikazuje slika 4.14. Glede na dobljene rezultate sklepamo, da izvajanje simulacije, ki traja 22000 simulacijskih sekund (z upoštevanjem prehodnega časa 2000 simulacijskih sekund), podaja dovolj dobre rezultate. Efektivno trajanje postopka simulacije znaša torej 20000 simulacijskih sekund.



Slika 4.14: Potek povprečnega odzivnega časa pri različnih časih simulacije

4.6.2 Eksperimentalno delo in rezultati

V nadaljevanju želimo z eksperimenti prikazati obnašanje simulacijskega modela pri različnih vhodnih parametrih. Z eksperimentom I želimo ugotoviti vpliv vhodne intenzivnosti in različne zmogljivosti centralne strežne enote na izhodne parametre. Pri eksperimentih II in III ugotavljamo vpliv verjetnosti vračanja paketov UDP oziroma vpliv različnega števila delovnih strežnih enot na izhodne parametre. V eksperimentu IV ugotavljamo kako tip razvrščanja paketov UDP vpliva na izhodne parametre. Eksperimenta V in VI pa poleg tipa razvrščanja, dodatno upoštevata še različne zmogljivosti delovnih strežnih enot. Za boljšo ponazoritev uporabe strežne mreže, kot modela prostovoljnega sistema, besedo "zahteva" enačimo z besedo "paket UDP".

4.6.2.1 Eksperiment I

Vhodne parametre eksperimenta prikazuje Tabela 9.

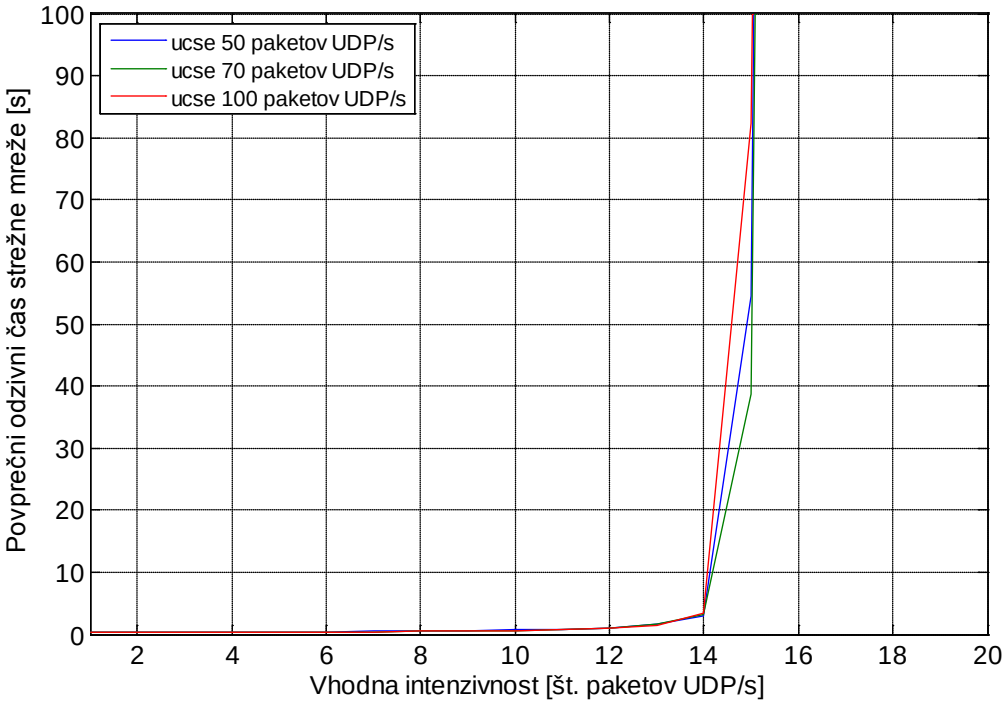
Vhodni parameter	Vrednost parametra
λ	1-1000
μ_{cse}	50 70 100
μ_{dseh}	5
μ_{dsesh}	0
μ_{dsep}	0
p	0,0
Razvrščanje na cse	0
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	3
n_{dsesh}	0
n_{dsep}	0
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	3
$L_{čv}$	∞
Čas simulacije	20 k sim. s

Tabela 9: Tabela vhodnih parametrov za simulacijski model eksperimenta I

Želimo ugotoviti:

1. vpliv vhodne intenzivnosti in različno zmogljivost centralne strežne enote na potek povprečnega odzivnega časa strežne mreže;
2. vpliv vhodne intenzivnosti na potek povprečne dolžine čakalnih vrst centralne strežne enote in delovnih strežnih enot pri $\mu_{cse} = 100{,}0\ zahtev/s$;
3. potek povprečne prepustnosti strežne mreže, centralne strežne enote in delovnih strežnih enot v odvisnosti od vhodne intenzivnosti pri $\mu_{cse} = 100{,}0\ zahtev/s$.

Pričakovati je bilo, da bo odziv strežne mreže podoben odzivu teoretičnega modela M/M/m. Različne zmogljivosti centralne strežne enote nimajo vpliva na potek odzivnega časa strežne mreže (slika 4.15). Do večjih razlik prihaja v okolici nasičenja strežne mreže, ki pa je zgolj posledica drugačnega naključnega razvrščanja pri vseh treh poskusih simulacije.

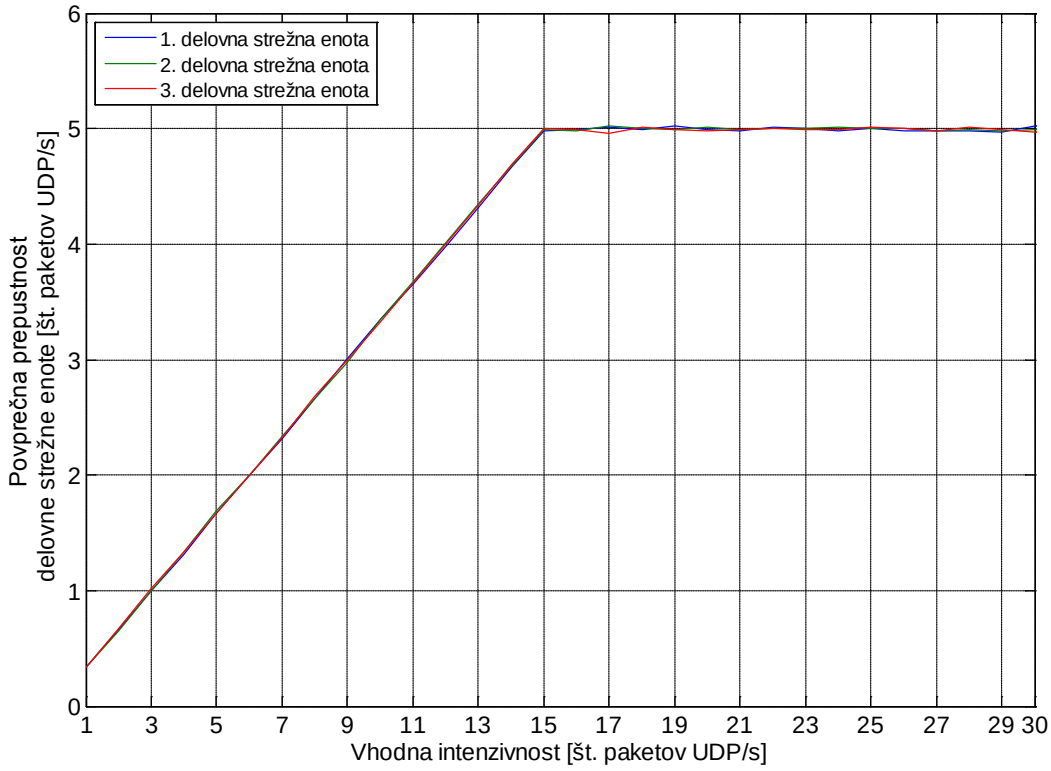


Slika 4.15: Potek povprečnega odzivnega časa strežne mreže eksperimenta I

Poskok povprečnega odzivnega časa strežne mreže pri zmogljivosti centralne strežne enote $\mu_{cse} = 100{,}0\ zahtev/s$, je zaslediti blizu vhodne intenzivnosti $\lambda = 15{,}0\ zahtev/s$. Do vzpona odzivnega časa strežne mreže pride zaradi omejene zmogljivosti vseh treh delovnih strežnih enot z zmogljivostjo $\mu_{dse_i} = 5{,}0\ zahtev/s$. Povprečna prepustnost delovnih strežnih enot se povečuje vse do vrednosti 5,0 zahtev/s (slika 4.16). To se zgodi blizu vrednosti

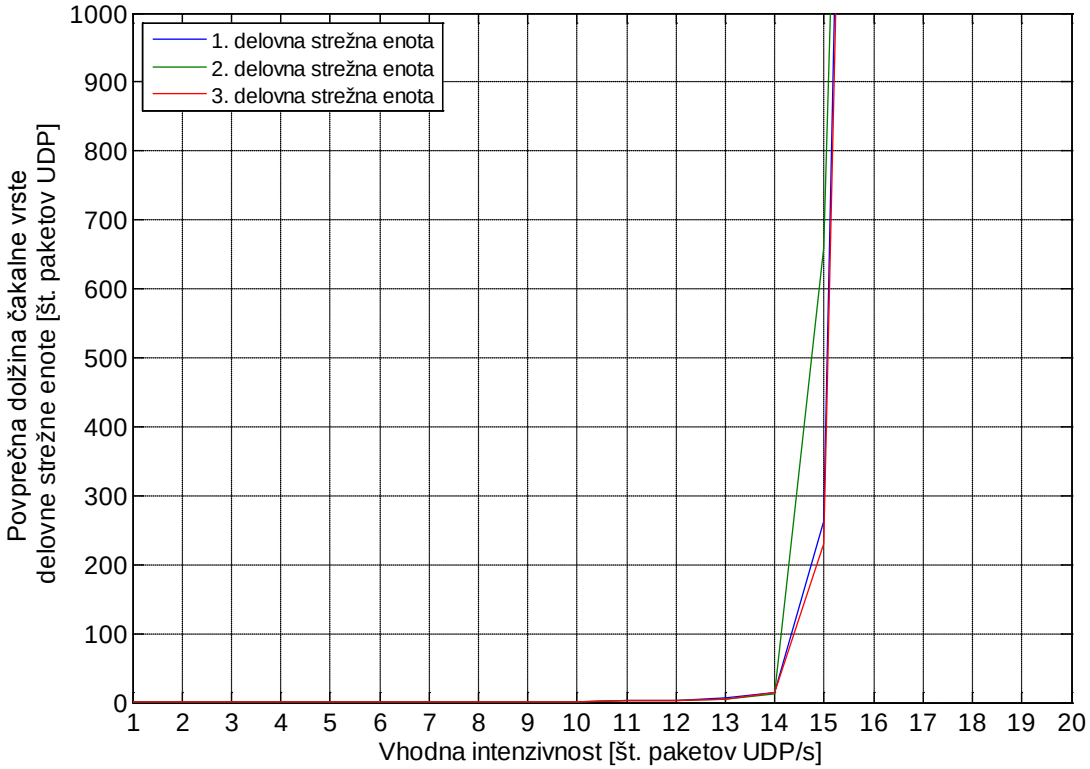
$$\lambda = m\mu_{dse_i} = 3 \cdot 5 = 15{,}0\ zahtev/s.$$

Prepustnost delovnih strežnih enot je seveda neodvisna od zmogljivosti centralne strežne enote. Po vhodni intenzivnosti $\lambda = 15{,}0\ zahtev/s$, delovne strežne enote ne morejo več obdelovati vseh nadaljnjih novih zahtev, zato se te pričnejo shranjevati v njihove čakalne vrste. Tam čakajo, dokler jih delovne strežne enote ne prevzamejo v izvajanje. Predpostavimo, da na odziv strežne mreže čakamo največ 60 sekund. Časi zadrževanja v strežni mreži so v tem primeru od vrednosti $\lambda = 15{,}0\ zahtev/s$ dalje nesprejemljivi. Od te vrednosti naprej strežna mreža ni več uporabna.



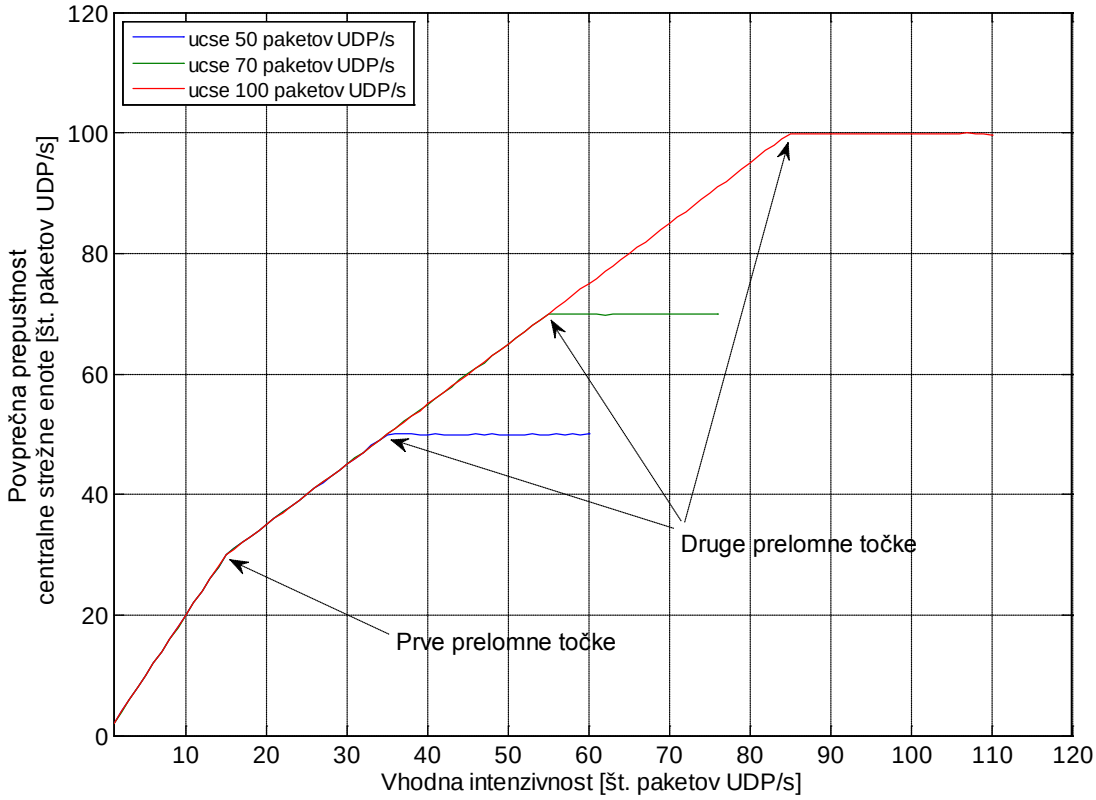
Slika 4.16: Potek povprečne prepustnosti delovnih strežnih enot eksperimenta I

Če se vhodna intenzivnost še naprej povečuje, postajajo čakalne vrste delovnih strežnih enot vse daljše (slika 4.17). Temu primerno daljši je tudi odzivni čas strežne mreže. Psevdonaključno razvrščanje povzroča favorizacijo druge delovne strežne enote, ki prejme več zahtev kot preostali delovni strežni enoti. V polno obremenitev zato ne preidejo povsem hkrati. Gotovo pa je, da njihove čakalne vrste od intenzivnosti $\lambda = 15{,}0\ zahtev/s$ naprej pričnejo hitro naraščati.



Slika 4.17: Potek povprečnih dolžin čakalnih vrst delovnih strežnih enot eksperimenta I

Slika 4.18 prikazuje potek povprečne prepustnosti centralne strežne enote. Pri zmogljivosti strežne enote $\mu_{cse} = 100{,}0\ zahtev/s$, povprečna prepustnost strežne mreže na intervalu $\lambda = 1{,}0\ zahtev/s$ - $\lambda = 15{,}0\ zahtev/s$ narašča enkrat hitreje kot na intervalu $\lambda = 15{,}0\ zahtev/s$ - $\lambda = 85{,}0\ zahtev/s$. Do hitrejšega naraščanja prihaja zaradi dveh virov zahtev na vhodu centralne strežne enote: vhodne intenzivnosti in povratne zanke, ki vrača pravilno obdelane zahteve. Intenzivnost, ki jo prejme centralna strežna enota, je torej enaka $2\ \lambda$. Pri vhodni intenzivnosti $\lambda = 15{,}0\ zahtev/s$ pa se pojavi prva prelomna točka. Od tu dalje je prispevek povratne zanke konstanten, saj delovne strežne enote v tej točki dosežejo svojo največjo prepustnost, njihove čakalne vrste pa pričnejo naraščati. Povečevanje povprečne prepustnosti povzroča le še vhodna intenzivnost λ. Centralna strežna enota je še vedno dovolj zmogljiva, da streže tako zahteve vhodne intenzivnosti kot tudi konstantno intenzivnost zahtev povratne zanke.



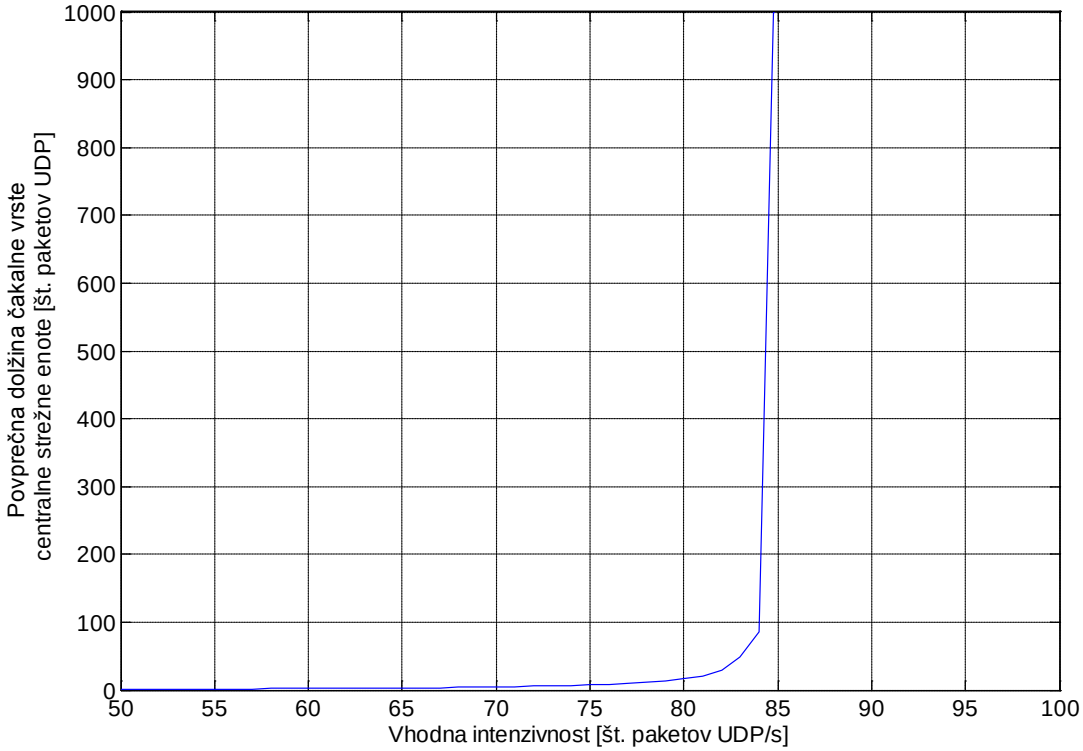
Slika 4.18: Potek povprečne prepustnosti centralne strežne enote eksperimenta I

Druga prelomna točka (v primeru $\mu_{cse} = 100$ zahtev/s) se nahaja blizu vrednosti, ki jo določa izraz

$$\lambda = \mu_{cse} - \sum_{i=1}^{m} \mu_{dse_i} = \mu_{cse} - \sum_{i=1}^{3} \mu_{dse_i} = 100 - 3 \cdot 5 = 85{,}0\ zahtev/s.$$

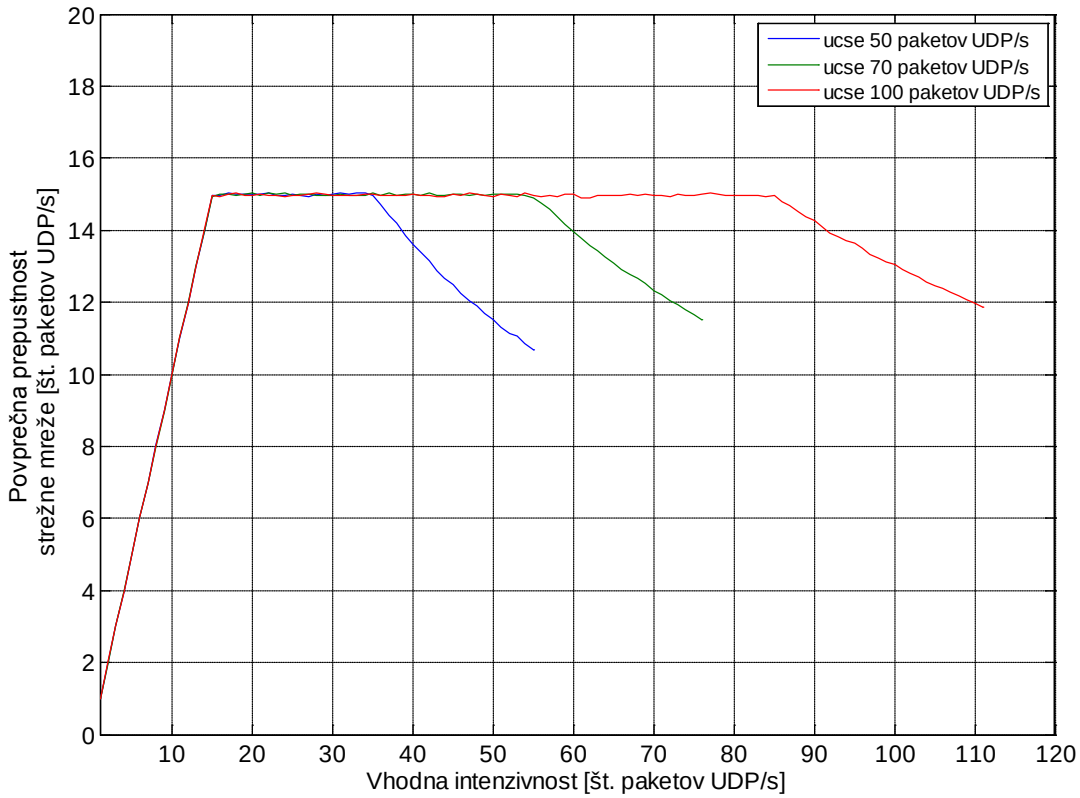
Vhodna intenzivnost mora narasti toliko, da preseže zmogljivost centralne strežne enote. Takrat postane njena prepustnost ustaljena ($X = 100{,}0\ zahtev/s$), čakalna vrsta pa prične naraščati (slika 4.19). Skupno zmogljivost delovnih strežnih enot določa izraz (5). Drugi prelomni točki se zato pojavita blizu vhodnih intenzivnosti $\lambda = 35{,}0\ zahtev/s$ (za $\mu_{cse} = 50$ zahtev/s) in $\lambda = 55{,}0\ zahtev/s$ (za $\mu_{cse} = 70$ zahtev/s). Manj ko je centralna strežna enota zmogljiva, prej doseže drugo prelomno točko pri poteku prepustnosti.

$$\sum_{i=1}^{m} \mu_{dse_i} = 15{,}0 \text{ zahtev/s} \tag{5}$$



Slika 4.19: Potek povprečne dolžine čakalne vrste centralne strežne enote eksperimenta I

Prepustnost strežne mreže (slika 4.20) je omejena z zmogljivostjo delovnih strežnih enot in narašča vse do vrednosti blizu $\lambda = 15{,}0\ zahtev/s$. Po tej vrednosti preidejo vse delovne strežne enote v nasičenje, njihove čakalne vrste pričnejo naraščati, prepustnost strežne mreže pa postane ustaljena. Takšna ostaja vse do vrednosti blizu $\lambda = 85{,}0\ zahtev/s$ (oziroma $\lambda = 35{,}0\ zahtev/s$ pri $\mu_{cse} = 50{,}0\ \ zahtev/s$ in $\lambda = 55{,}0\ zahtev/s$ pri $\mu_{cse} = 70{,}0\ zahtev/s$). Po tej vrednosti prične naraščati tudi čakalna vrsta centralne strežne enote. To pa je vzrok pričetka padanja prepustnosti strežne mreže. To pomeni, da obdelane zahteve redkeje zapuščajo strežno mrežo, saj so bolj pomešane z neobdelanimi zahtevami. Prepustnost strežne mreže doseže vrednost 0 le, kadar vhodno intenzivnost povečujemo v neskončnost.



Slika 4.20: Potek povprečne prepustnosti strežne mreže eksperimenta I

Rezultate za strežno mrežo eksperimenta I prikazuje Tabela 10.

Izhodni parameter	**Vrednost**
$\overline{T}$	60 sekund
Interval uporabnosti	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paket UDP/s do $\lambda = 15{,}0$ paket UDP/s
$\overline{L_{cse}}$ (pri $\mu_{cse} = 100$)	Od 0,10 do 0,12 paketov UDP
$\overline{L_{čv_1}}$ (pri $\mu_{cse} = 100$)	Od 14,9 do 262,2 paketov UDP
$\overline{L_{čv_2}}$ (pri $\mu_{cse} = 100$)	Od 13,2 do 660,8 paketov UDP
$\overline{L_{čv_3}}$ (pri $\mu_{cse} = 100$)	Od 14,2 do 229,8 paketov UDP
$\bar{X}$ (pri $\mu_{cse} = 100$)	Do 15 paketov UDP/s

Tabela 10: Rezultati eksperimenta I

Pričakovati je, da bo prostovoljni sistem s psevdonaključnim razvrščanjem in delovnimi vozlišči enake zmogljivosti približno uravnoteženo obremenjen. Prepustnost sistema je pri dovolj zmogljivem centralnem strežniku pogojena izključno z zmogljivostjo delovnih vozlišč. Bolj ko so ta zmogljiva, krajši bo odzivni čas in večja prepustnost celotnega sistema.

4.6.2.2 Eksperiment II

Vhodne parametre eksperimenta prikazuje Tabela 11.

Vhodni parameter	Vrednost parametra
λ	1-1000
μ_{cse}	100
μ_{dseh}	5
μ_{dsesh}	0
μ_{dsep}	0
p	0,0 (EKSP. I) 0,3 0,6 0,9
Razvrščanje na cse	0
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	3
n_{dsesh}	0
n_{dsep}	0
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	3
$L_{čv}$	∞
Čas simulacije	20 k sim. s

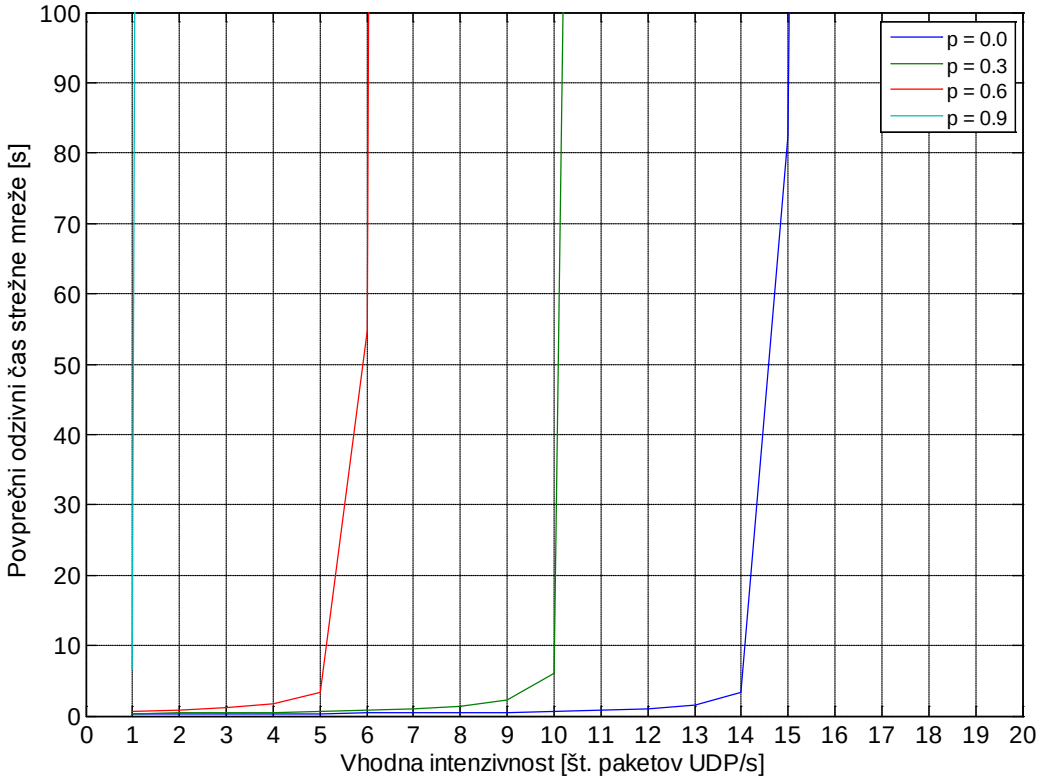
Tabela 11: Tabela vhodnih parametrov za simulacijski model eksperimenta II

Ugotoviti želimo:

1. vpliv vhodne intenzivnosti in različne verjetnosti vračanja zahtev na potek povprečnega odzivnega časa strežne mreže;
2. vpliv vhodne intenzivnosti in različne verjetnosti vračanja na potek povprečne prepustnosti centralne strežne enote, prve delovne strežne enote in strežne mreže.

Večja ko je verjetnost vračanja, hitreje preide strežna mreža v nasičenje (slika 4.21). Če je verjetnost ponovnega pošiljanja na delovne strežne enote velika, se njihove čakalne vrste zaradi cikličnega vračanja ustrezno hitreje povečujejo. Strežna mreža je tu poleg vhodne intenzivnosti in povratne intenzivnosti zanke obdelanih zahtev obremenjena tudi s povratno intenzivnostjo zanke neobdelanih zahtev. Ker je zmogljivost delovnih strežnih enot omejena, je omejena tudi skupna povratna intenzivnost obdelanih zahtev s strani delovnih strežnih enot. Če zopet predpostavimo, da je odzivni čas 60 sekund še sprejemljiv, lahko za posamezno verjetnost vračanja zahtev razberemo naslednje rezultate povprečnega odzivnega časa strežne mreže:

- $p = 0{,}0 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s - 15{,}0\ zahtev/s$,
- $p = 0{,}3 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s - 11{,}0\ zahtev/s$,
- $p = 0{,}6 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s - 7{,}0\ zahtev/s$,
- $p = 0{,}9 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s - 2{,}0\ zahtev/s$.



Slika 4.21: Potek povprečnega odzivnega časa strežne mreže eksperimenta II

Iz numeričnih vrednosti simulacije lahko sklepamo, da konstantno povečevanje verjetnosti vračanja zahtev na centralno strežno enoto povzroča konstantno krčenje intervala vhodnih intenzivnosti, pri katerih je strežna mreža še uporabna. Večja ko je verjetnost vračanja, prej postanejo delovne strežne enote nasičene. Ko dosežejo svojo največjo možno prepustnost, se pričnejo njihove čakalne vrste polniti z zahtevami. Slika 4.22 prikazuje povprečno prepustnost prve delovne strežne mreže v odvisnosti od vhodne intenzivnosti in različnih verjetnosti vračanja. Potek povprečne prepustnosti ostalih delovnih strežnih enot je podoben. Za vsako od verjetnosti vračanja je opaziti različne prelomne točke. Od teh točk dalje postane prepustnost delovnih strežnih enot največja, njihove čakalne vrste pa pričnejo naraščati. Verjetnost $p = 1{,}0$ povzroči neskončen odzivni čas, saj nobena zahteva nikoli ne zapusti strežne mreže.

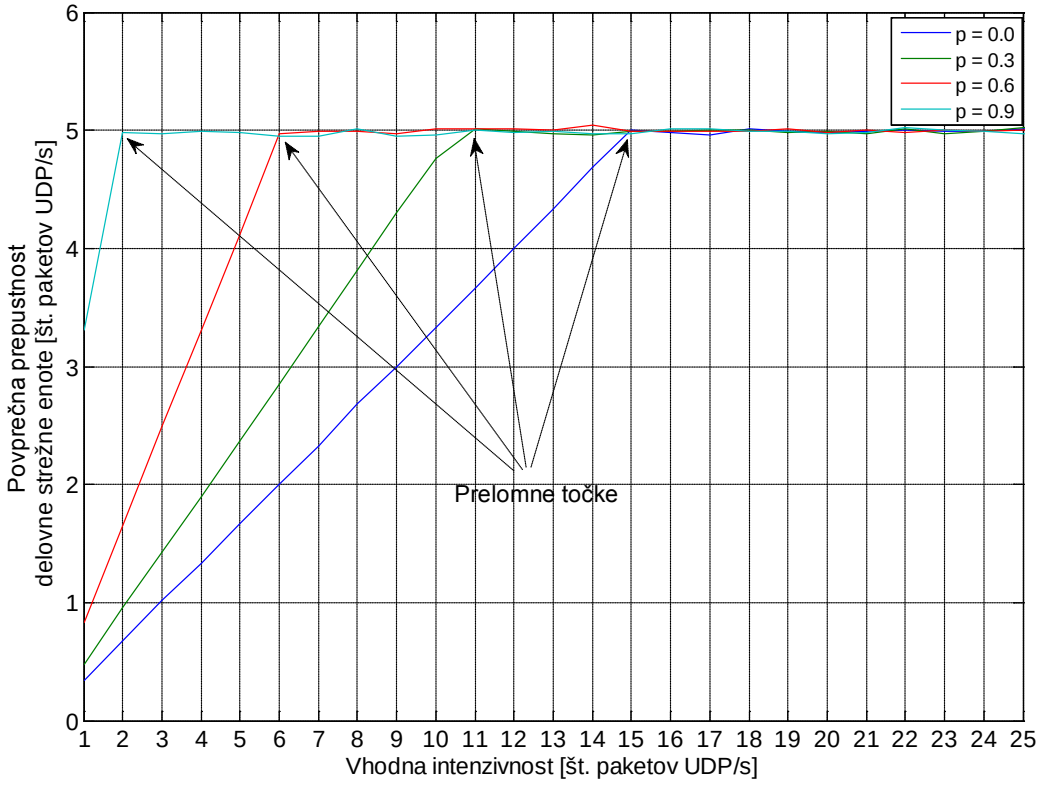
V strežni mreži so aktivne tri delovne strežne enote enakih zmogljivosti. Če predpostavimo, da psevdonaključno razvrščanje povzroči približno enakomerno obremenitev delovnih strežnih enot, mora do prelomne točke (nasičenja delovne strežne enote) veljati: zmogljivost poljubne delovne strežne enote mora biti enaka vsoti m-tega dela vhodne intenzivnosti, m-tega dela povratne intenzivnosti z verjetnostjo p, m-tega dela povratne intenzivnosti z verjetnostjo p^2, m-tega dela povratne intenzivnosti z verjetnostjo p^3, ... Vse skupaj opišemo z enačbo (6).

$$\mu_{dse_i} = \frac{\lambda}{m} + \sum_{i=1}^{\infty}\left(\frac{\lambda}{m}p^i\right) = \frac{\lambda}{m}\left(1 + \sum_{i=1}^{\infty}p^i\right) = \frac{\lambda}{m}\left(1 + \frac{p}{1-p}\right) = \frac{\lambda}{m}\left(\frac{1}{1-p}\right) \qquad (6)$$

Iz enačbe (6) sledi enačba (7), ki podaja vhodno intenzivnost nasičenja delovne strežne enote.

$$\lambda = m\mu_{dse_i}(1-p) \tag{7}$$

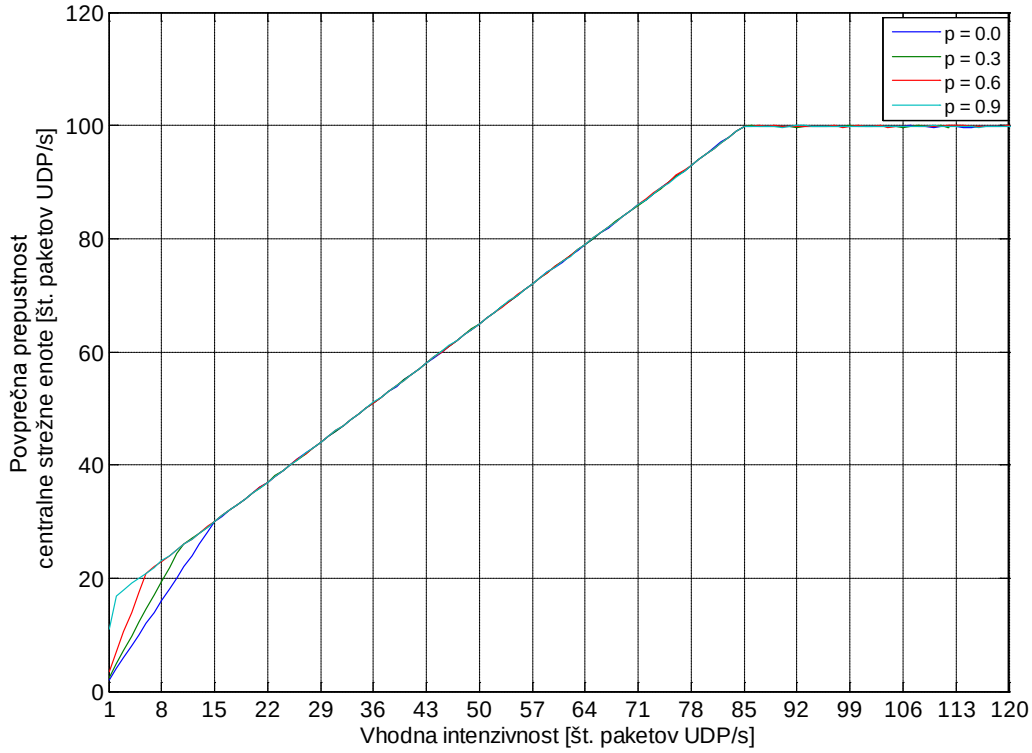
Prelomna točka se pri verjetnosti $p = 0{,}0$ pojavi blizu vhodne intenzivnosti $\lambda = 15{,}0\ zahtev/s$, pri verjetnosti $p = 0{,}3$ blizu vhodne intenzivnosti $\lambda = 10{,}5\ zahtev/s$, pri verjetnosti $p = 0{,}6$ blizu vhodne intenzivnosti $\lambda = 6{,}0\ zahtev/s$ in pri verjetnosti $p = 0{,}9$ blizu vhodne intenzivnosti $\lambda = 1{,}5\ zahtev/s$. Izračunani rezultati so skladni z numeričnimi vrednostmi pri izvajanju simulacije in grafu, ki ga prikazuje slika 4.22.



Slika 4.22: Potek povprečne prepustnosti prve delovne strežne enote eksperimenta II

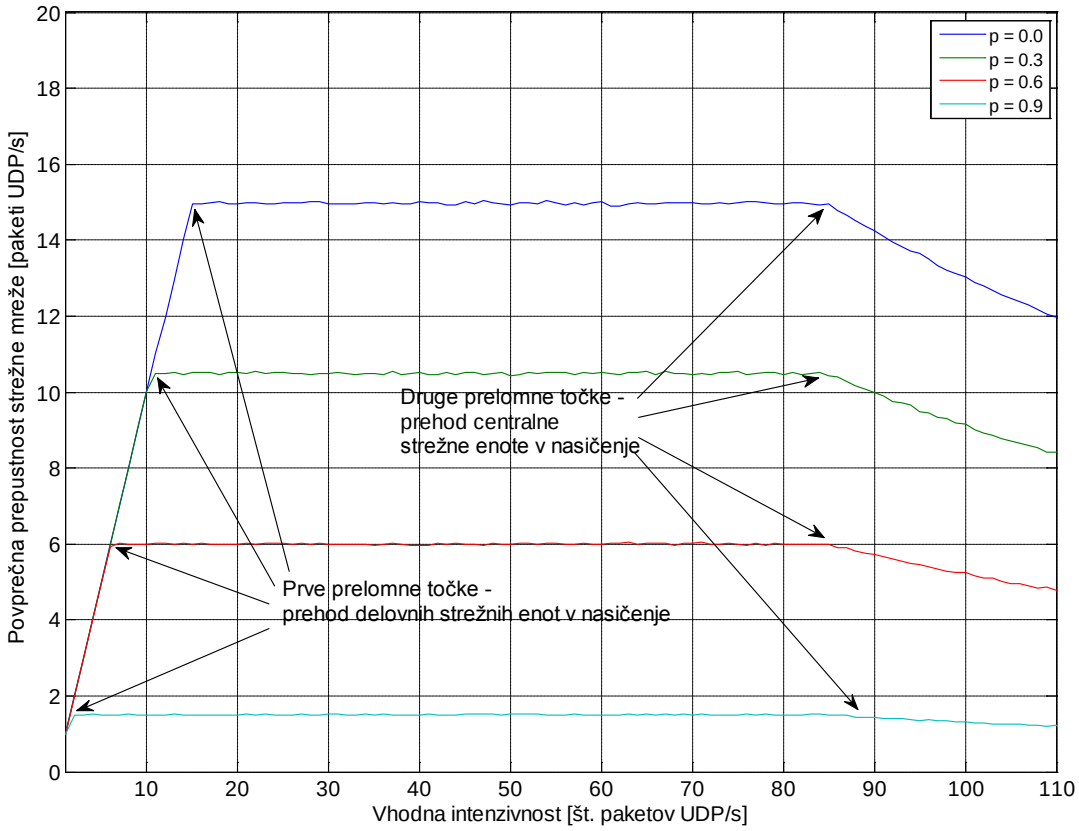
Na grafu je sicer opaziti, da se največja prepustnost delovnega vozlišča pri verjetnosti vračanja zahtev $p = 0{,}9$, pojavi blizu vrednosti $\lambda = 2{,}0\ zahtev/s$, izračunana vrednost pa je enaka $\lambda = 1{,}5\ zahtev/s$. V resnici se pojavi nekje med vrednostjo $\lambda = 1{,}0\ zahtev/s$ in $\lambda = 2{,}0\ zahtev/s$. Graf točne vrednosti žal ne prikazuje, saj je izbrana ločljivost vhodne intenzivnosti prevelika. Postopek simulacije je bil namreč opravljen le na vsako celo število vhodne intenzivnosti.

Že pri eksperimentu I smo ugotovili, da zmogljivosti delovnih strežnih enot vplivajo na potek povprečne prepustnosti centralne strežne enote (slika 4.18). Tu pa poleg delovnih strežnih enot vpliva tudi verjetnost vračanja neobdelanih zahtev. Pri večji verjetnosti vračanja povprečna prepustnost centralne strežne enote narašča hitreje (slika 4.23) in skladno s tem tudi pojav prelomnih točk. Nadaljnji potek povprečne prepustnosti centralne strežne enote je podoben poteku pri eksperimentu I.



Slika 4.23: Potek povprečne prepustnosti centralne strežne enote eksperimenta II

Največja možna prepustnost strežne mreže pri verjetnosti vračanja $p = 0{,}0$ je omejena na vrednost $X = 15{,}0\ zahtev/s$ (slika 4.24). Drugače tudi ne more biti, saj to vrednost določajo zmogljivosti delovnih strežnih enot. Prve prelomne točke so skladne s pojavom nasičenja delovnih strežnih enot (slika 4.22), druge prelomne točke pa z nasičenjem centralne strežne enote.



Slika 4.24: Potek povprečne prepustnosti strežne mreže eksperimenta II

Rezultate za strežno mrežo eksperimenta II prikazuje Tabela 12.

Izhodni parameter	Vrednost
$\overline{T}$	60 sekund
Interval uporabnosti pri $p = 0{,}0$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paket UDP/s do $\lambda = 15{,}0$ paket UDP/s
Interval uporabnosti pri $p = 0{,}3$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paket UDP/s do $\lambda = 11{,}0$ paket UDP/s
Interval uporabnosti pri $p = 0{,}6$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paket UDP/s do $\lambda = 7{,}0$ paket UDP/s
Interval uporabnosti pri $p = 0{,}9$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paket UDP/s do $\lambda = 2{,}0$ paket UDP/s
$\bar{X}$ pri $p = 0{,}0$	Do 15,0 paketov UDP/s
$\bar{X}$ pri $p = 0{,}3$	Do 10,5 paketov UDP/s
$\bar{X}$ pri $p = 0{,}6$	Do 6,0 paketov UDP/s
$\bar{X}$ pri $p = 0{,}9$	Do 1,5 paketov UDP/s

Tabela 12: Rezultati eksperimenta II

Če so v prostovoljnem sistemu delovna vozlišča nezanesljiva, postane takšen sistem zelo hitro neodziven. Večino časa se namreč nameni ponovnemu izvajanju napačno izvedenih opravil, s tem pa se zanemarjajo nova, še neizvedena opravila.

4.6.2.3 Eksperiment III

Vhodne parametre eksperimenta prikazuje Tabela 13.

Vhodni parameter	Vrednost parametra
λ	1-1000
μ_{cse}	100
μ_{dseh}	5
μ_{dsesh}	0
μ_{dsep}	0
p	0,3
Razvrščanje na cse	0
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	3 (EKSP. II) 6 9 12
n_{dsesh}	0
n_{dsep}	0
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	3 6 9 12
$L_{čv}$	∞
Čas simulacije	20 k sim. s

Tabela 13: Tabela vhodnih parametrov za simulacijski model eksperimenta III

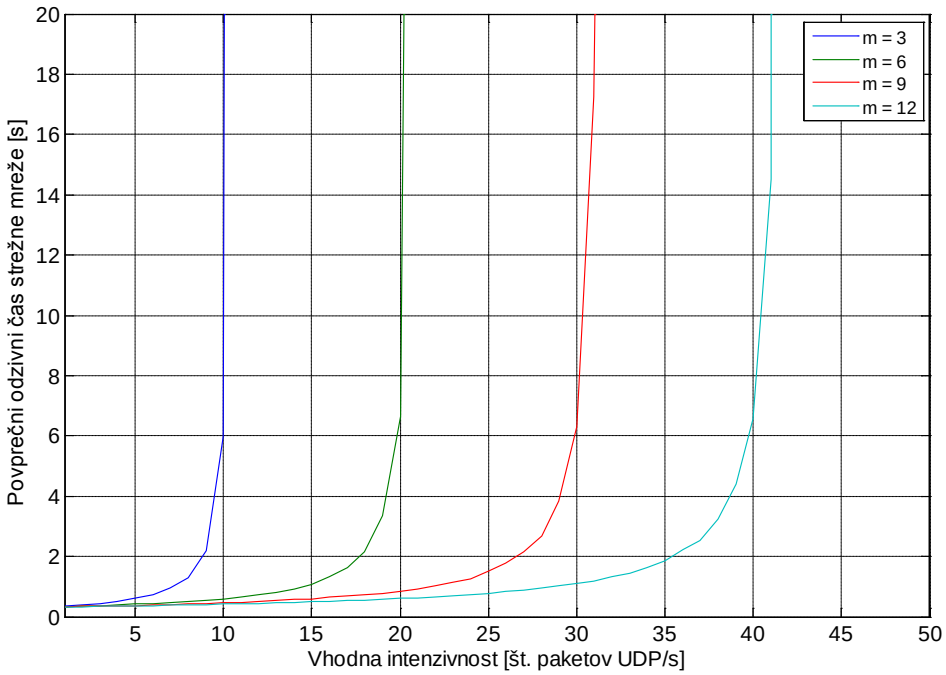
Ugotoviti želimo:

1. vpliv vhodne intenzivnosti in različnega števila delovnih strežnih enot na potek povprečnega odzivnega časa strežne mreže;
2. vpliv vhodne intenzivnosti na potek povprečne dolžine čakalnih vrst šestih delovnih strežnih enot ($m = 6$);
3. vpliv vhodne intenzivnosti in različnega števila delovnih strežnih enot na potek povprečne prepustnosti strežne mreže, centralne strežne enote in prve delovne strežne enote.

Večje število delovnih strežnih enot pomeni, da je strežna mreža sposobna obdelati več zahtev v enakem času. Če je odzivni čas 60 s še sprejemljiv, potem smo s pomočjo numeričnih rezultatov postopka simulacije ugotovili sledeče:

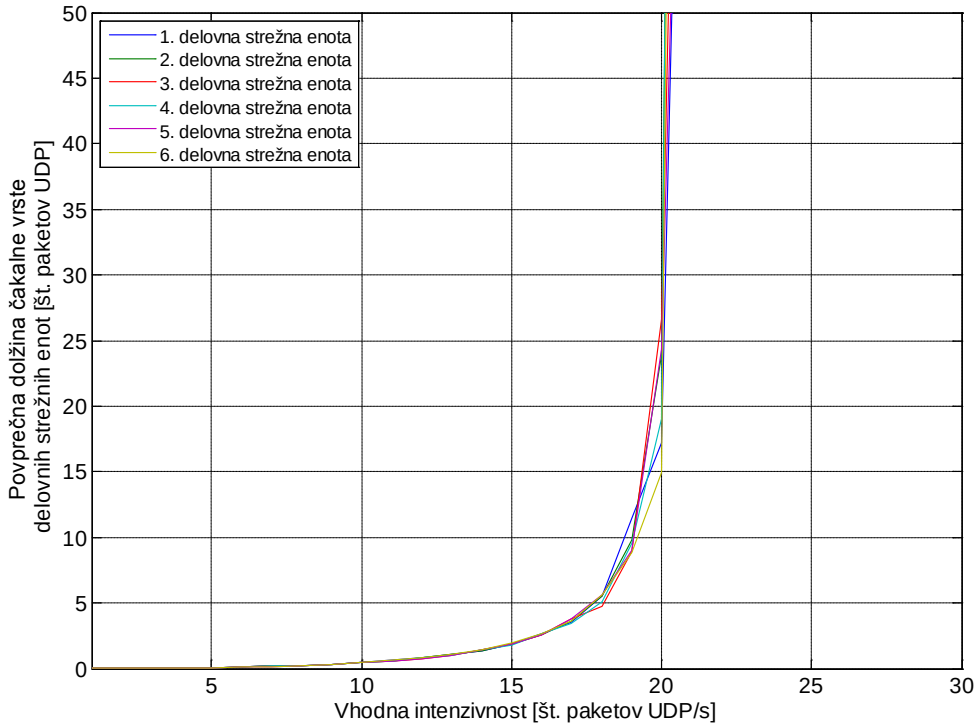
- $m = 3 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s -$ 11,0 $zahtev/s$,
- $m = 6 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s -$ 21,0 $zahtev/s$,
- $m = 9 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s -$ 32,0 $zahtev/s$,
- $m = 12 \rightarrow$ strežna mreža uporabna na intervalu 1,0 $zahtev/s -$ 42,0 $zahtev/s$.

Navedene numerične vrednosti in slika 4.25 kažejo, da konstantno povečevanje števila delovnih strežnih enot povzroči približno konstantno širjenje intervala, v katerem je strežna enota še uporabna.



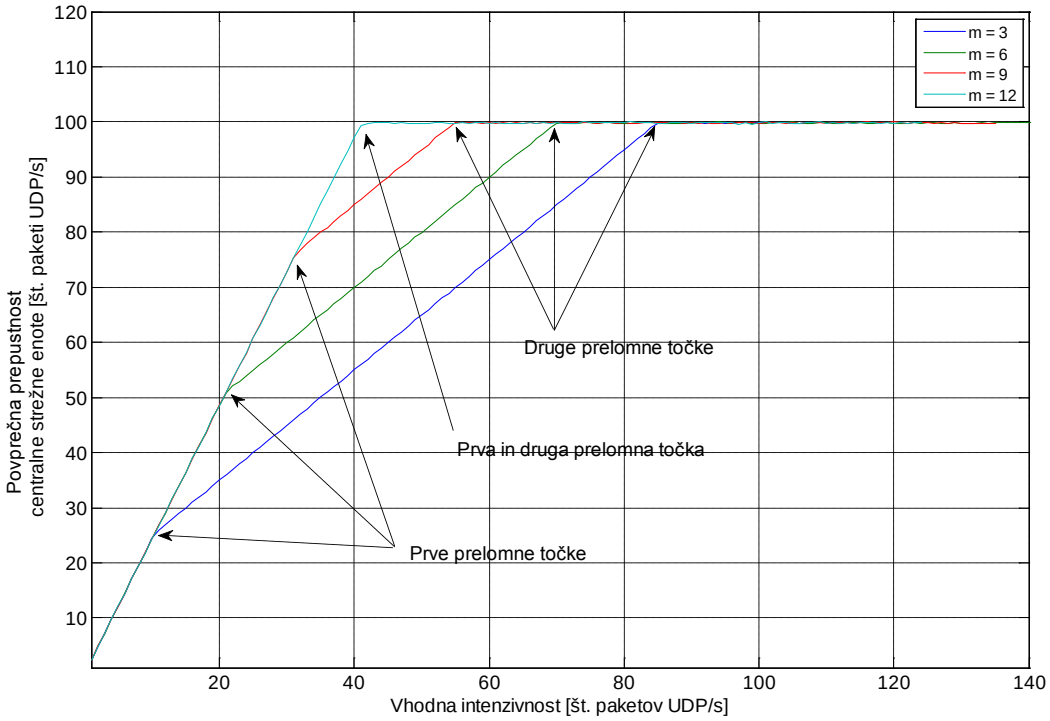
Slika 4.25: Potek povprečnega odzivnega časa strežne mreže eksperimenta III

Potek $m = 6$ čakalnih vrst delovnih strežnih enot prikazuje slika 4.26. Podobno kot pri eksperimentu I lahko zaradi uporabe psevdonaključnega razvrščanja opazimo favorizacije. Nekatere delovne strežne enote prejmejo več zahtev, druge manj.



Slika 4.26: Potek povprečnih dolžin čakalnih vrst delovnih strežnih enot eksperimenta III

Prepustnost centralne strežne enote prikazuje slika 4.27. Do prvih prelomnih točk prihaja pri vhodnih intenzivnostih, kjer delovne strežne enote dosežejo svojo največjo prepustnost oziroma preidejo v nasičenje.



Slika 4.27: Potek povprečne prepustnosti centralne strežne enote eksperimenta III

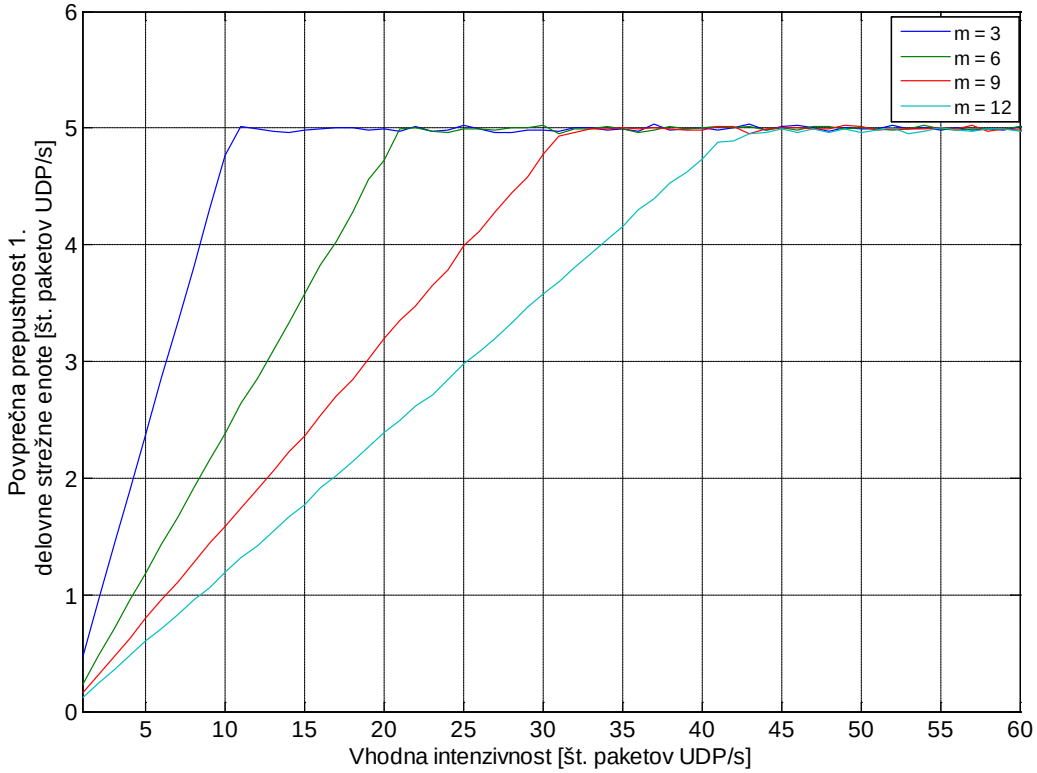
Zopet predpostavimo, da psevdo naključno razvrščanje povzroči približno enakomerno obremenitev delovnih strežnih enot. Podobno kot pri eksperimentu II mora veljati enačba $\lambda = m\mu_{dse_i}(1-p)$. Delovne strežne enote preidejo v nasičenje blizu vhodne intenzivnosti $\lambda = 11{,}5\, zahtev/s$ za $m = 3$ delovne strežne enote, $\lambda = 23{,}0\, zahtev/s$ za $m = 6$, $\lambda = 34{,}6\, zahtev/s$ za $m = 9$ in $\lambda = 42{,}0\, zahtev/s$ za $m = 12$ delovnih strežnih enot. Numerični rezultati postopka simulacije so skladni z navedenimi vrednostmi, približne vrednosti pa nam podaja slika 4.28.

Vzrok drugih prelomnih točk je pojav nasičenosti centralne strežne enote, ki se pojavi takrat, ko velja izraz

$$\mu_{cse} = \lambda + \sum_{i=1}^{m} \mu_{dse_i}.$$

Nadalje lahko izrazimo vhodno intenzivnost, blizu katere centralna strežna enota preide v nasičenje

$$\lambda = \mu_{cse} - \sum_{i=1}^{m} \mu_{dse_i}.$$



Slika 4.28: Potek povprečne prepustnosti prve delovne strežne enote eksperimenta III

Pri $m = 3$ delovnih strežnih enotah se nasičenje pojavi blizu vrednosti vhodne intenzivnosti $\lambda = 85{,}0\ zahtev/s$, pri $m = 6$ blizu vrednosti $\lambda = 70{,}0\ zahtev/s$, pri $m = 9$ pa blizu vrednosti $\lambda = 55{,}0\ zahtev/s$. Numerične vrednosti so skladne z analitičnimi.

V primeru $m = 12$ delovnih strežnih enot se pojavi le ena prelomna točka, kot kaže slika 4.27. Vzrok za to je, da se nasičenje centralne strežne enote pojavi pred pojavom nasičenja delovnih strežnih enot. Skupna vhodna intenzivnost centralne strežne enote tik pred nastopom nasičenja mora biti zato enaka izrazu

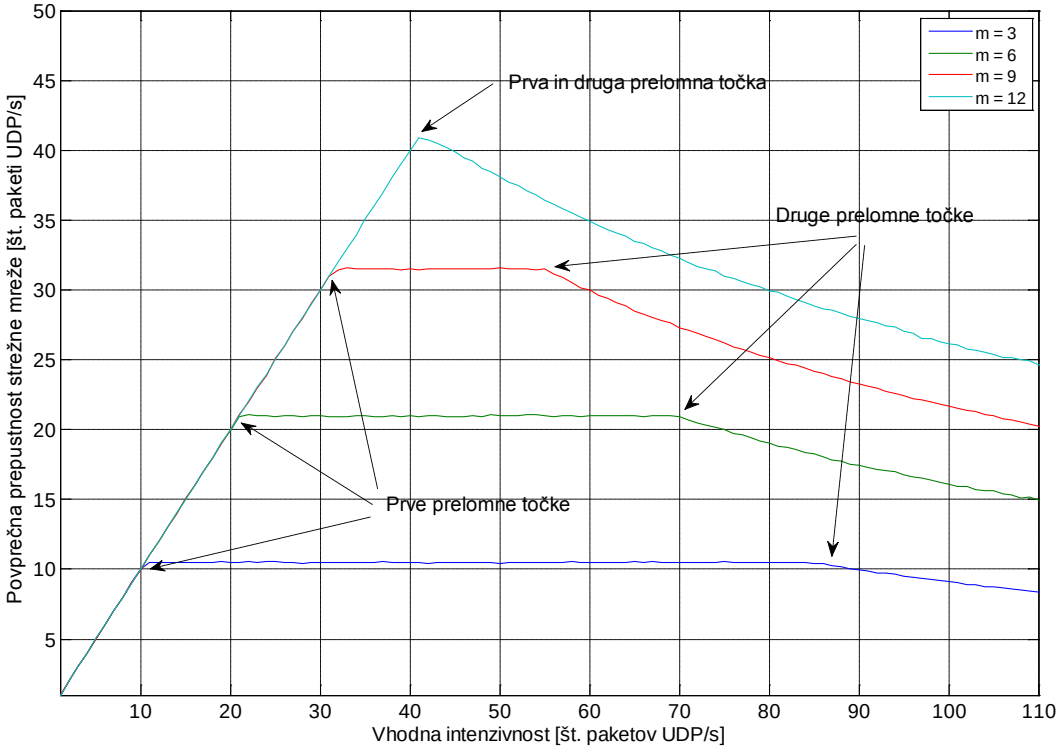
$$\mu_{cse} = \lambda + \frac{\lambda}{m}m + \frac{\lambda}{m}m\sum_{i=1}^{\infty}(p^i) = 2\lambda + \lambda\left(\frac{p}{1-p}\right).$$

Centralna strežna enota je torej obremenjena z vhodno intenzivnostjo, povratno izhodno intenzivnostjo delovnih strežnih enot uspešno obdelanih zahtev in povratno izhodno intenzivnostjo delovnih strežnih enot, ki jo povzročajo napačno obdelane zahteve. Nasičenje se zato nahaja blizu vrednosti, ki jo določa izraz

$$\lambda = \frac{\mu_{cse}}{2 + \frac{p}{1-p}} = \left(\frac{\mu_{cse}(1-p)}{2(1-p)+p}\right) = \left(\frac{100 \cdot (1-0{,}3)}{2(1-0{,}3)+0{,}3}\right) = 41{,}17 zahtev/s.$$

Centralna strežna enota naj bi blizu vrednosti vhodne intenzivnosti $\lambda = 41{,}17\ zahtev/s$ prešla v nasičenje. Opazimo, da se to zgodi še pred nasičenjem vseh $m = 12$ delovnih strežnih enot, do katerega pa pride blizu vrednosti vhodne intenzivnosti $\lambda = 42{,}0\ zahtev/s$. Vrednosti $\lambda = 41{,}17\ zahtev/s$ in $\lambda = 42{,}0\ zahtev/s$ sta tako blizu skupaj, da prehod centralne strežne enote v nasičenje skoraj sovpada s prehodom delovnih strežnih enot v nasičenje. Slika 4.28 sicer prikazuje potek prepustnosti le ene izmed $m = 12$ delovnih strežnih enot, a je potek podoben tudi pri ostalih delovnih strežnih enotah strežne mreže.

Potek povprečne prepustnosti strežne mreže prikazuje slika 4.29. Prve in druge prelomne točke sovpadajo s prelomnimi točkami, ki ji kaže slika 4.27. Prepustnost strežne mreže prične padati (druge prelomne točke), ko centralna strežna enota preide v nasičenje.



Slika 4.29: Potek povprečne prepustnosti strežne mreže eksperimenta III

Rezultate za strežno mrežo eksperimenta III prikazuje Tabela 14.

Izhodni parameter	Vrednost
$\overline{T}$	60 sekund
Interval uporabnosti pri $m = 3$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paketov UDP/s do $\lambda = 11{,}0$ paketov UDP /s
Interval uporabnosti pri $m = 6$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paketov UDP /s do $\lambda = 21{,}0$ paketov UDP /s
Interval uporabnosti pri $m = 9$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paketov UDP /s do $\lambda = 32{,}0$ paketov UDP /s
Interval uporabnosti pri $m = 12$	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paketov UDP /s do $\lambda = 42{,}0$ paketov UDP /s
$\overline{L_{cse}}$	Do 0,4 paketov UDP
$\overline{L_{čv_1}}$ pri $m = 6$	Do 111,3 paketov UDP/s
$\overline{L_{čv_2}}$ pri $m = 6$	Do 254,4 paketov UDP/s
$\overline{L_{čv_3}}$ pri $m = 6$	Do 137,0 paketov UDP/s
$\overline{L_{čv_4}}$ pri $m = 6$	Do 278,4 paketov UDP/s
$\overline{L_{čv_5}}$ pri $m = 6$	Do 106,4 paketov UDP/s
$\overline{L_{čv_6}}$ pri $m = 6$	Do 269,9 paketov UDP/s
$\bar{X}$ pri $m = 3$	Do 11,5 paketov UDP/s
$\bar{X}$ pri $m = 6$	Do 21,0 paketov UDP/s
$\bar{X}$ pri $m = 9$	Do 32,0 paketov UDP/s
$\bar{X}$ pri $m = 12$	Do 42,0 paketov UDP/s

Tabela 14: Rezultati eksperimenta III

Večje število delovnih vozlišč prostovoljnega sistema bo zagotovo omogočalo, da sistem prenese večje obremenitve oziroma ima večjo prepustnost. Treba pa je poudariti, da lahko z večjim številom delovnih vozlišč centralno vozlišče postane potencialno ozko grlo sistema. Kot takšno prispeva k povečanemu odzivnemu času in manjši prepustnosti sistema. Zato je treba pri večjem številu delovnih vozlišč zagotoviti tudi dovolj zmogljiv centralni strežnik. Pomembna ugotovitev je tudi ta, da v takšnem sistemu preidejo delovna vozlišča v nasičenje približno hkrati. To pomeni, da so v nasičenju vsa delovna vozlišča ali pa nobeno od njih.

4.6.2.4 Eksperiment IV

Vhodne parametre eksperimenta prikazuje Tabela 15.

Vhodni parameter	Vrednost parametra
λ	1-1000
μ_{cse}	100
μ_{dseh}	5
μ_{dsesh}	0
μ_{dsep}	0
p	0,3
Razvrščanje na cse	1
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	6
n_{dsesh}	0
n_{dsep}	0
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	6
$L_{čv}$	∞
Čas simulacije	20 k sim. s

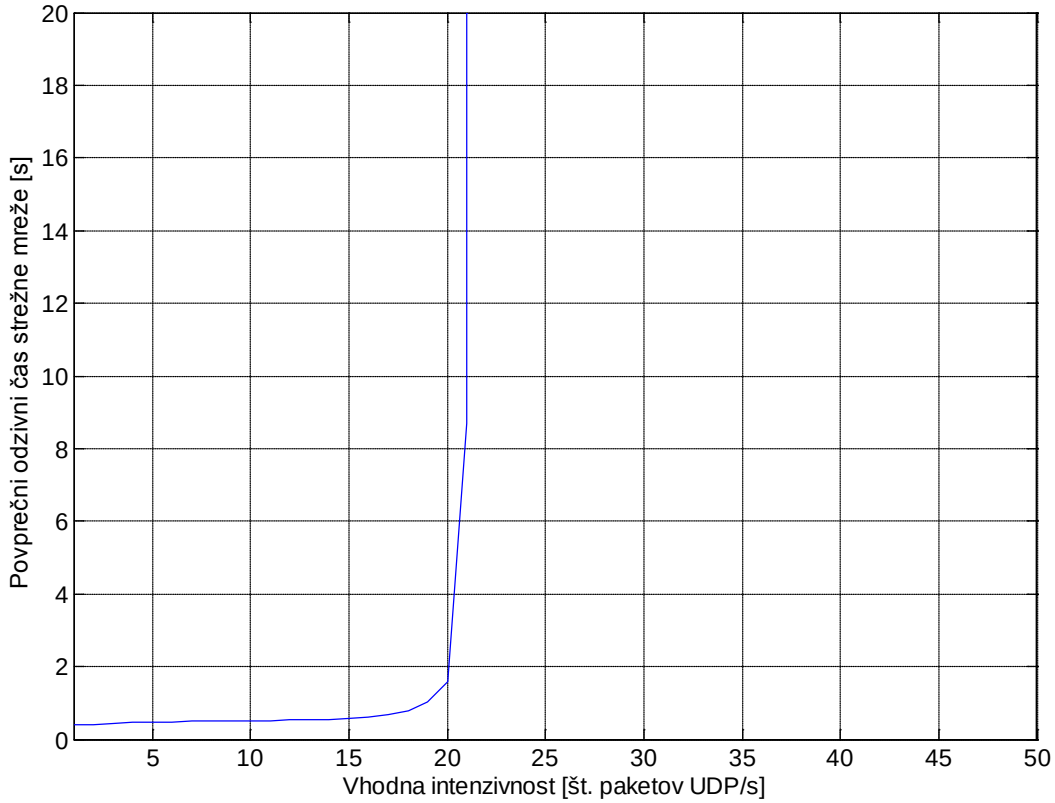
Tabela 15: Tabela vhodnih parametrov za simulacijski model eksperimenta IV

Ugotoviti želimo:

1. vpliv vhodne intenzivnosti in vpliv načina razvrščanja po principu najkrajša čakalna vrsta najprej na potek povprečnega odzivnega časa strežne mreže;
2. vpliv vhodne intenzivnosti na potek povprečne dolžine čakalnih vrst centralne strežne enote in delovnih strežnih enot;
3. potek dolžine čakalnih vrst delovnih strežnih enot v času simulacije pri vhodni intenzivnosti $\lambda = 15{,}0\ zahtev/s$.

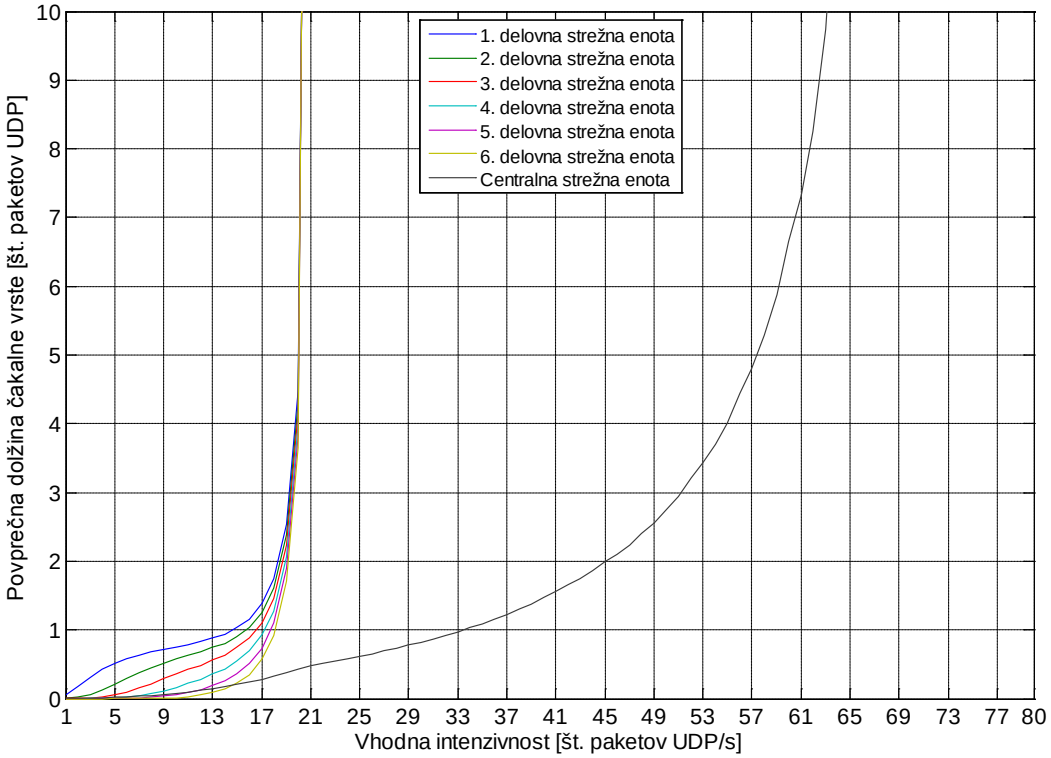
Eksperiment je podoben eksperimentu III z $m = 6$ delovnimi strežnimi enotami. Na splošno velja, da skušamo v prostovoljnih sistemih v čim večji meri izkoriščati najzmogljivejša delovna vozlišča. Tak pristop zagotavlja, da rešitev na oddani problem dobimo kar najhitreje. Zato razvrščanje po principu najkrajša čakalna vrsta najprej v strežni mreži deluje tako, da se v trenutku razvrščanja opravi pregled dolžin vseh čakalnih vrst delovnih strežnih enot. Pregledovanje se prične pri najzmogljivejši in konča pri najmanj zmogljivi delovni strežni enoti. Zahteva se nato pošlje delovni strežni enoti, ki ima najkrajšo čakalno vrsto v tistem trenutku. To pomeni, da se zahteva praviloma pošlje najmanj obremenjeni delovni strežni enoti. Tako skušamo najprej izkoristiti najzmogljivejše, nato srednje zmogljive in nazadnje najmanj zmogljive delovne strežne enote. Če obstaja več delovnih strežnih enot z enako najkrajšo dolžino čakalne vrste, se zahteva pošlje prvi izbrani.

Če je odzivni čas 60 sekund še sprejemljiv, potem lahko s pomočjo numeričnih rezultatov izvajanja simulacije sklepamo, da je strežna mreža uporabna na intervalu $\lambda = 1{,}0\ zahtev/s$ - $\lambda = 22{,}0\ zahtev/s$. Dobljeni rezultati so zelo blizu rezultatom eksperimenta III. Naključno razvrščanje povzroči približno enakomerno razporeditev zahtev na vhodnih delovnih strežnih enotah, zato večjih odstopanj pri poteku odzivnega časa strežne mreže tu ni bilo pričakovati (slika 4.30).



Slika 4.30: Potek povprečnega odzivnega časa strežne mreže eksperimenta IV

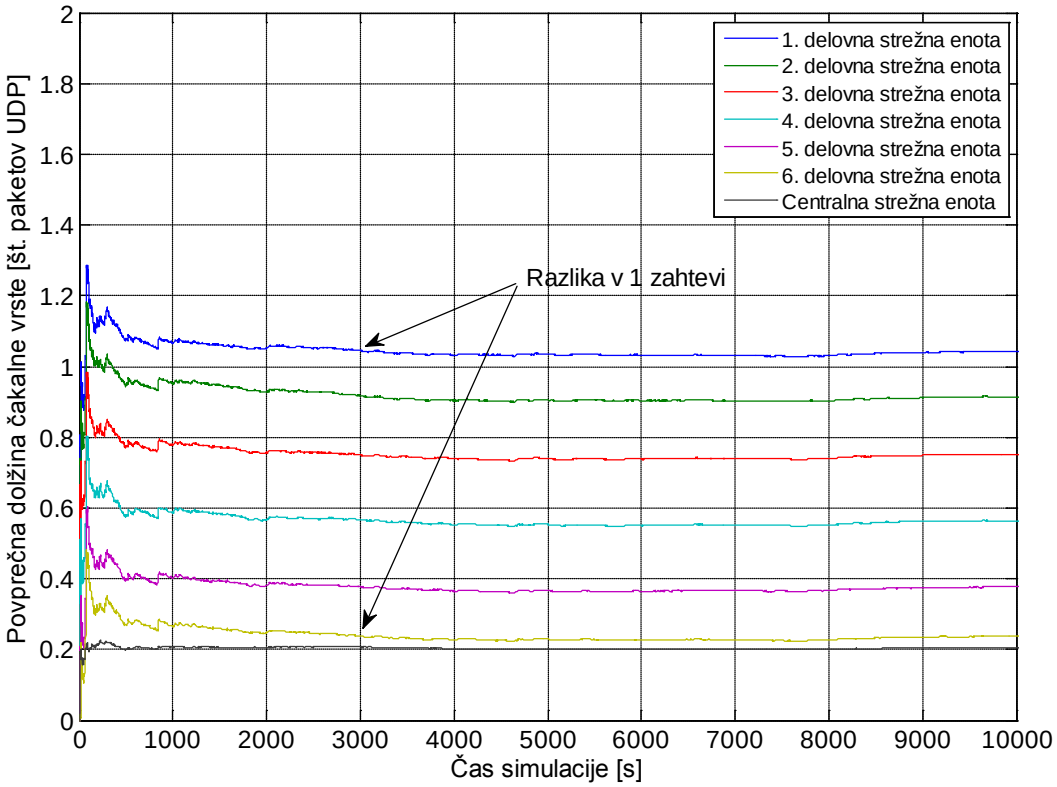
Povsem drugačen potek pa zasledimo pri poteku povprečnih dolžin čakalnih vrst delovnih strežnih enot (slika 4.31). Pri naključnem razvrščanju je bila favorizacija delovnih strežnih enot izrazita. Tu pa s spremembo razvrščanja dosežemo veliko enakomernejšo porazdelitev zahtev po delovnih strežnih enotah. Zaradi načina razvrščanja oziroma načina obremenjevanja delovnih strežnih enot (po vrsti glede na zmogljivost) nam slika 4.31 kaže izrazit snop potekov povprečnih dolžin čakalnih vrst. Kot prve z vhodno intenzivnostjo pričnejo naraščati čakalne vrste najzmogljivejših delovnih strežnih enot, nato narastejo čakalne vrste srednje zmogljivih in nazadnje najmanj zmogljivih delovnih strežnih enot. Centralna strežna enota preide v nasičenje kot zadnja.



Slika 4.31: Potek povprečnih dolžin čakalnih vrst eksperimenta IV

Razliko med dolžinami čakalnih vrst delovnih strežnih enot nam prikazuje tudi slika 4.32, na kateri razberemo potek dolžin čakalnih vrst za vse delovne strežne enote pri vhodni intenzivnosti $\lambda = 15{,}0\ zahtev/s$. Sliko 4.32 lahko primerjamo s sliko 4.13 (poglavje 4.6.1), pri postopku ugotavljanja primerne velikosti prehodnega časa. Pri tem eksperimentu zaradi načina razvrščanja ni zaslediti tako velikih nestabilnosti, kot jih kaže slika 4.13. Še vedno so v začetku prisotne, a je njihova izrazitost manjša. Konvergenca proti stabilnemu stanju je hitrejša in veliko enakomernejša. Potek pa je podoben za vse delovne strežne enote. Iz slike 4.32 je prav tako mogoče oceniti, da razlika med dolžino čakalne vrste najzmogljivejše delovne strežne enote in dolžino čakalne vrste najmanj zmogljive delovne strežne enote nikoli ne preseže vrednosti 1 zahteve.

Potek prepustnosti strežne mreže, centralne strežne enote in delovnih strežnih enot je zelo podoben poteku prepustnosti eksperimenta III, zato jih na tem mestu ne navajamo.



Slika 4.32: Potek dolžin čakalnih vrst delovnih strežnih enot v času simulacije eksperimenta IV

Rezultate za strežno mrežo eksperimenta IV prikazuje Tabela 16.

Izhodni parameter	Vrednost
$\overline{T}$	60 sekund
Interval uporabnosti	Strežna mreža je uporabna na intervalu $\lambda = 1{,}0$ paketov UDP/s do $\lambda = 22{,}0$ paketov UDP /s
$\overline{L_{cse}}$	Do 0,5 paketov UDP
$\overline{L_{čv_1}}$	Do 1629,4 paketov UDP
$\overline{L_{čv_2}}$	Do 1629,2 paketov UDP
$\overline{L_{čv_3}}$	Do 1629,1 paketov UDP
$\overline{L_{čv_4}}$	Do 1628,9 paketov UDP
$\overline{L_{čv_5}}$	Do 1628,7 paketov UDP
$\overline{L_{čv_6}}$	Do 1628,5 paketov UDP

Tabela 16: Rezultati eksperimenta IV

Sprememba načina razvrščanja v prostovoljnem sistemu omogoča bistveno enakomernejše obremenjevanje delovnih vozlišč. Sistem ne dovoljuje stanj, v katerih je eno delovno vozlišče bolj obremenjeno, medtem ko so druga povsem neobremenjena oziroma neizkoriščena. V nasičenje vedno preidejo vse delovne strežne enote hkrati. Ali so v nasičenju vse delovne strežne enote ali pa nobena.

4.6.2.5 Eksperiment V

Vhodne parametre eksperimenta prikazuje Tabela 17.

Vhodni parameter	Vrednost parametra
λ	1-1000
μ_{cse}	150
μ_{dseh}	30
μ_{dsesh}	10
μ_{dsep}	5
p	0,0 0,3 0,6 0,9
Razvrščanje na cse	0
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	2
n_{dsesh}	2
n_{dsep}	2
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	6
$L_{čv}$	∞
Čas simulacije	20 k sim. s

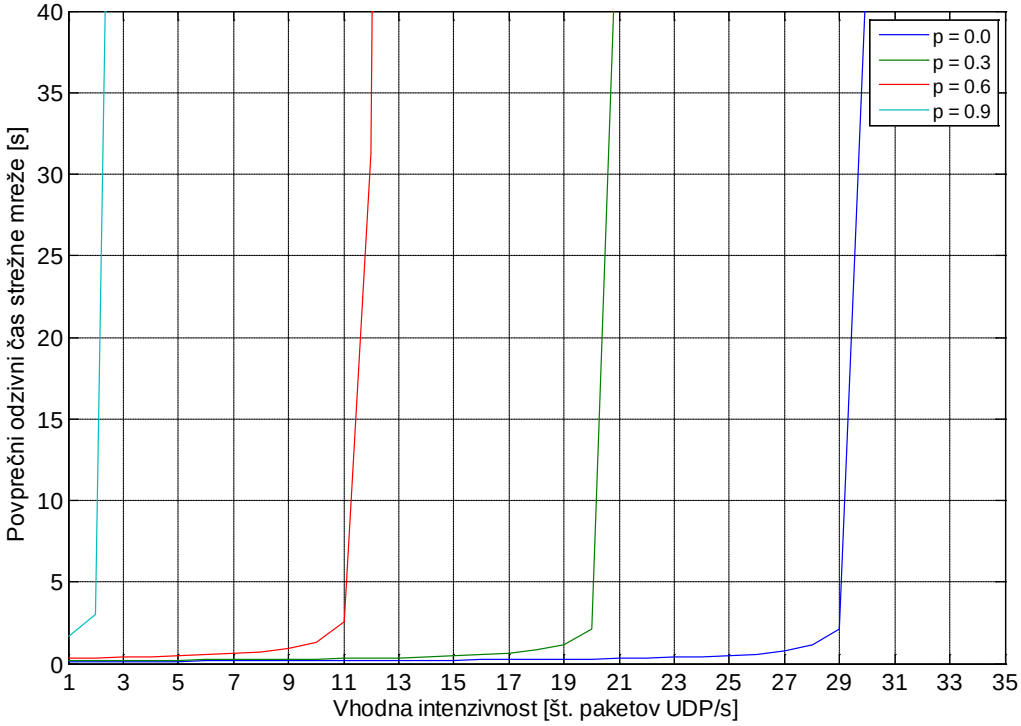
Tabela 17: Tabela vhodnih parametrov za simulacijski model eksperimenta V

Ugotoviti želimo:

1. vpliv vhodne intenzivnosti in različnih verjetnosti vračanja na potek povprečnega odzivnega časa strežne mreže; upoštevamo različno zmogljive delovne strežne enote;
2. vpliv vhodne intenzivnosti in verjetnosti vračanja $p = 0{,}3$ na potek povprečne dolžine čakalnih vrst delovnih strežnih enot in centralne strežne enote; upoštevamo različno zmogljive delovne strežne enote;
3. vpliv vhodne intenzivnosti na potek povprečne prepustnosti strežne mreže pri verjetnosti vračanja $p = 0{,}3$; upoštevamo različno zmogljive delovne strežne enote.

Potek odzivnega časa strežne mreže (slika 4.33) je bil pričakovan. Naraščanje vhodne intenzivnosti in večja verjetnost vračanja zahtev na vhod vplivata na ustrezno hitrejše povečevanje dolžine čakalnih vrst centralne strežne enote in delovnih strežnih enot. To pa pomeni hitrejše nasičenje strežne mreže.

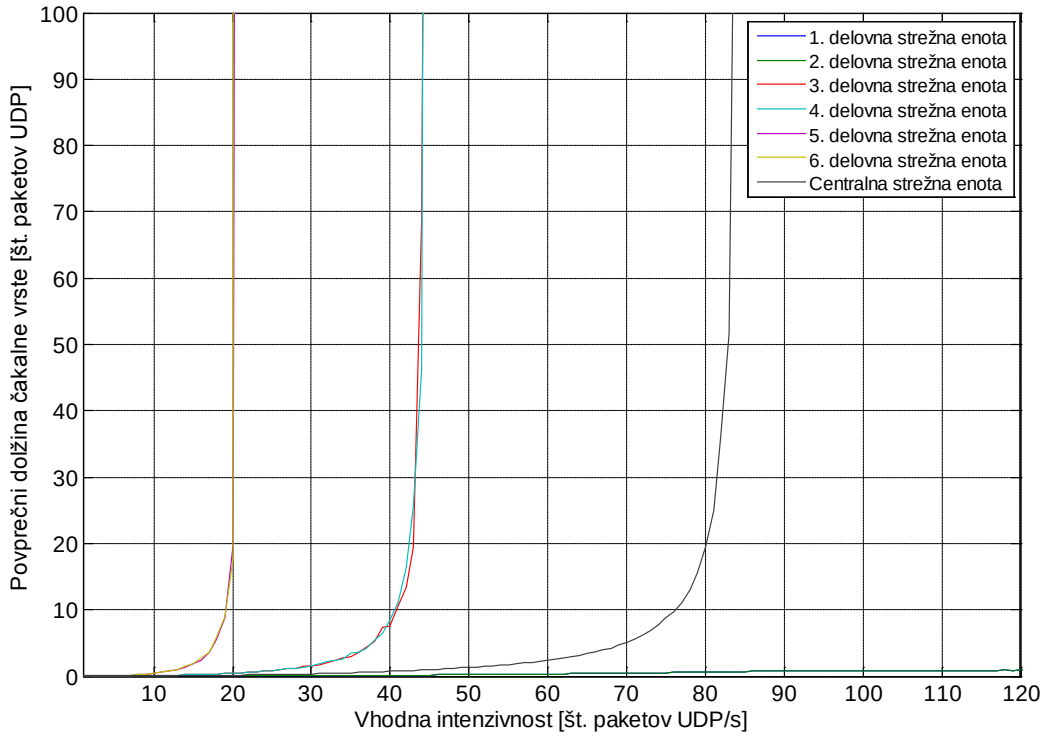
Do nasičenja strežne mreže prihaja v trenutkih, ko najmanj zmogljivi delovni strežni enoti strežne mreže dosežeta svojo največjo prepustnost, njuni čakalni vrsti pa se pričneta polniti. Zopet predpostavimo, da psevdonaključno razvrščanje povzroči približno enakomerno obremenitev delovnih strežnih enot. Strežna mreža preide v nasičenje, ko preideta v nasičenje najmanj zmogljivi delovni strežni enoti. Približno intenzivnost, pri kateri se to zgodi, lahko zopet določimo s pomočjo izraza $\lambda = m\mu_{dse_i}(1-p)$. Če je število delovnih strežnih enot enako $m = 6$, najmanjša zmogljivost delovne strežne enote v strežni mreži pa je $\mu_{dse} = 5{,}0\ zahtev/s$, potem strežna mreža z verjetnostjo vračanja $p = 0{,}0$ preide v nasičenje blizu vrednosti $\lambda = 30{,}0\ zahtev/s$. Podobno velja za vse ostale verjetnosti vračanja: če je verjetnost enaka $p = 0{,}3$, se nasičenje pojavi blizu vrednosti $\lambda = 21{,}0\ zahtev/s$, če je verjetnost enaka $p = 0{,}6$, se nasičenje pojavi blizu vrednosti $\lambda = 12{,}0\ zahtev/s$, če je verjetnost enaka $p = 0{,}9$, se nasičenje pojavi blizu vrednosti $\lambda = 3{,}0\ zahtev/s$.



Slika 4.33: Potek povprečnega odzivnega časa strežne mreže eksperimenta V

Slika 4.34 prikazuje potek povprečnih dolžin čakalnih vrst delovnih vozlišč pri verjetnosti vračanja $p = 0{,}3$. Peta in šesta delovna strežna enota preideta v nasičenje blizu vrednosti $\lambda = 21{,}0\ zahtev/s$. Njuna zmogljivost je v strežni mreži najmanjša. Kot naslednji preideta v nasičenje tretja in četrta delovna strežna enota, in sicer blizu vrednosti $\lambda = 42{,}0\ zahtev/s$. Prva in druga delovna strežna enota nikoli ne preideta v nasičenje. Prej namreč preide v nasičenje centralna strežna enota (slika 4.34). V trenutku nasičenja centralne strežne enote mora veljati izraz

$$\mu_{cse} = \lambda + 2\mu_{dsep} + 2\mu_{dsesh} + 2\frac{\lambda}{m} + \sum_{i=1}^{\infty} 2\frac{\lambda}{m}p^i\ .$$

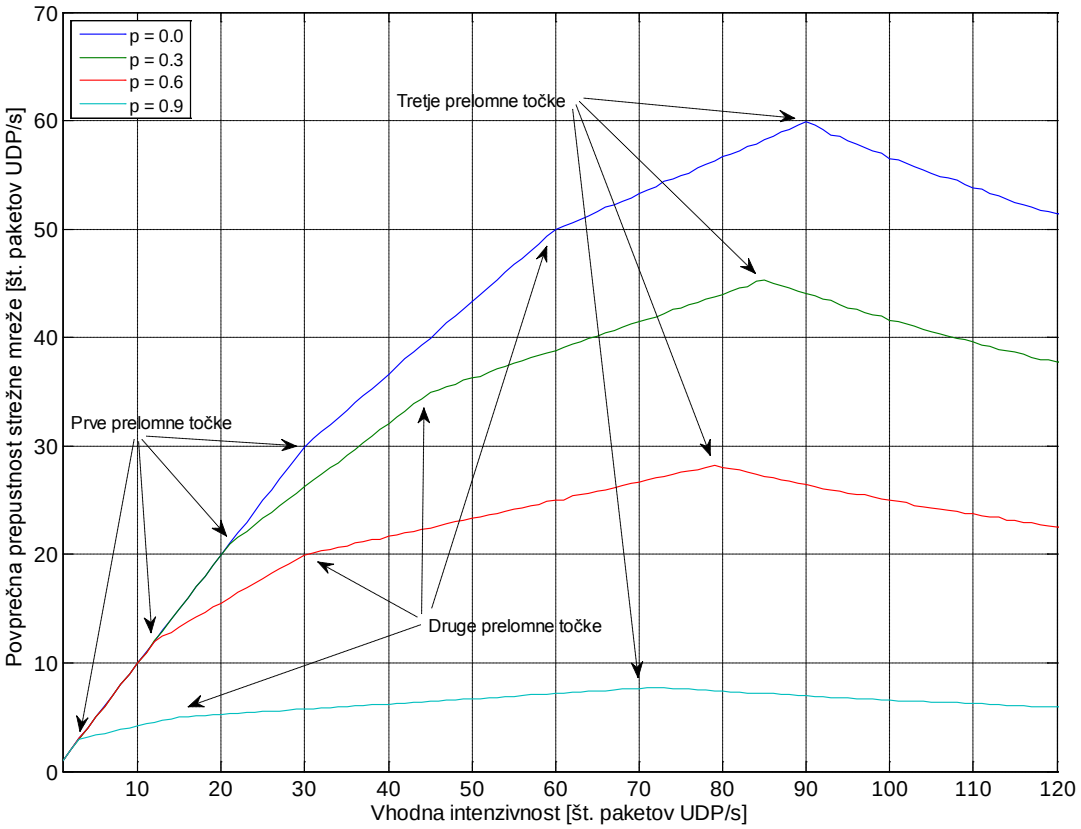


Slika 4.34: Potek povprečnih dolžin čakalnih vrst delovnih strežnih enot in centralne strežne enote eksperimenta V

Obremenjena je torej z vhodno intenzivnostjo, največjo možno izhodno intenzivnostjo najmanj zmogljivih in srednje zmogljivih delovnih strežnih enot ter izhodno intenzivnostjo obeh najzmogljivejših delovnih strežnih enot. Temu sta dodana tudi deleža povratne obremenitve zaradi verjetnosti vračanja $p = 0{,}3$. Vrednost vhodne intenzivnosti, pri kateri centralna strežna enota preide v nasičenje, določa izraz

$$\lambda = \frac{m\mu_{cse} - 2m\mu_{dsep} - 2m\mu_{dsesh}}{m + 2\left(\frac{1}{1-p}\right)} = \frac{6 \cdot 150 - 2 \cdot 6 \cdot 5 - 2 \cdot 6 \cdot 10}{6 + 2\left(\frac{1}{1-0{,}3}\right)} = 81{,}29\ zahtev/s\ .$$

Centralna strežna enota torej preide v nasičenje blizu vrednosti vhodne intenzivnosti $\lambda = 81{,}29\ zahtev/s$ (slika 4.34). Njena zmogljivost je premajhna, da bi lahko prenesla skupno breme vhodne intenzivnosti in povratne intenzivnosti vseh delovnih strežnih enot. Potek prepustnosti strežne mreže za različne verjetnosti vračanja prikazuje slika 4.35. Zopet je zaslediti 3 prelomne točke. Do prve pride zaradi nasičenja najmanj zmogljivih delovnih strežnih enot, do druge zaradi nasičenja srednje hitrih delovnih strežnih enot in do tretje zaradi nasičenja centralne strežne enote.



Slika 4.35: Potek povprečne prepustnosti strežne mreže eksperimenta V

Rezultate za strežno mrežo eksperimenta V prikazuje Tabela 18.

Izhodni parameter	Vrednost
$\overline{T}$	60 sekund
Interval uporabnosti pri $p = 0{,}0$	Model je uporaben na intervalu $\lambda = 1{,}0$ paketov UDP/s do $\lambda = 31{,}0$ paketov UDP/s
Interval uporabnosti pri $p = 0{,}3$	Model je uporaben na intervalu $\lambda = 1{,}0$ paketov UDP/s do $\lambda = 22{,}0$ paketov UDP/s
Interval uporabnosti pri $p = 0{,}6$	Model je uporaben na intervalu $\lambda = 1{,}0$ paketov UDP/s do $\lambda = 13{,}0$ paketov UDP/s
Interval uporabnosti pri $p = 0{,}9$	Model je uporaben na intervalu $\lambda = 1{,}0$ paketov UDP/s do $\lambda = 3{,}0$ paketov UDP/s
$\overline{L_{cse}}$ pri $p = 0{,}3$	Do 0,18 paketov UDP
$\overline{L_{čv_1}}$ pri $p = 0{,}3$	0,0 paketov UDP
$\overline{L_{čv_2}}$ pri $p = 0{,}3$	0,0 paketov UDP
$\overline{L_{čv_3}}$ pri $p = 0{,}3$	0,6 paketov UDP
$\overline{L_{čv_4}}$ pri $p = 0{,}3$	0,5 paketov UDP
$\overline{L_{čv_5}}$ pri $p = 0{,}3$	2034,2 paketov UDP
$\overline{L_{čv_6}}$ pri $p = 0{,}3$	2083,0 paketov UDP
$\overline{X}$ pri $p = 0{,}0$	Do 59,9 paketov UDP/s
$\overline{X}$ pri $p = 0{,}3$	Do 45,2 paketov UDP/s
$\overline{X}$ pri $p = 0{,}6$	Do 28,1 paketov UDP/s
$\overline{X}$ pri $p = 0{,}9$	Do 7,7 paketov UDP/s

Tabela 18: Rezultati eksperimenta V

Pričakovati je, da bo v prostovoljnem sistemu z različno zmogljivimi delovnimi vozlišči in naključnim razvrščanjem opravil nasičenje povzročila najprej množica najmanj zmogljivih delovnih vozlišč. Sledi jim nasičenje srednje zmogljivih in nato nasičenje najzmogljivejših delovnih vozlišč. Takšno obnašanje sistema je pričakovati le, če je zmogljivost centralne strežne enote dovolj visoka. Sicer lahko centralno vozlišče hitro postane preobremenjeno in ozko grlo sistema. Poleg tega pa nekatera delovna vozlišča ostajajo tudi nepopolno izkoriščena.

4.6.2.6 Eksperiment VI

Vhodne parametre eksperimenta prikazuje Tabela 19.

Vhodni parameter	Vrednost parametra
λ	1-1000
μ_{cse}	300
μ_{dseh}	30
μ_{dsesh}	10
μ_{dsep}	5
p	0,0 0,3 0,6 0,9
Razvrščanje na cse	1
s_{cse}	10 Mbps
τ	1 ms
n_{dseh}	2
n_{dsesh}	2
n_{dsep}	2
$m = n_{dseh} + n_{dsesh} + n_{dsep}$	6
$L_{čv}$	∞
Čas simulacije	20 k sim. s

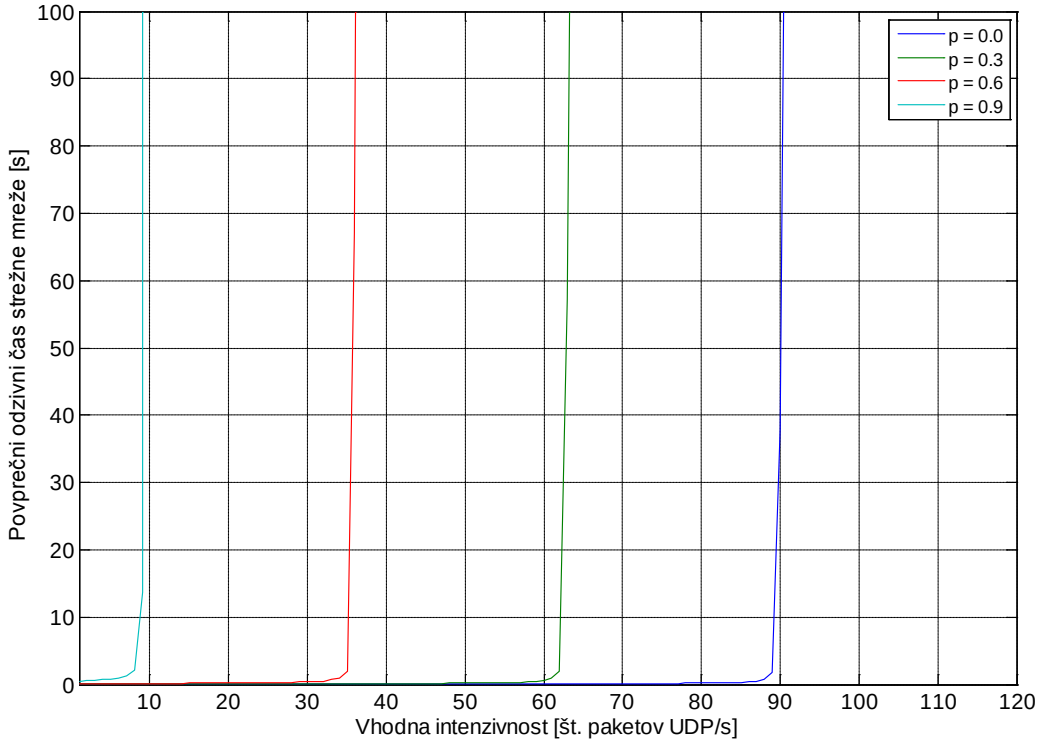
Tabela 19: Tabela vhodnih parametrov za simulacijski model eksperimenta VI

Pri tem eksperimentu je zagotovljena dovolj velika zmogljivost centralne strežne enote, da ta ne predstavlja več ozkega grla strežne mreže. Njena zmogljivost je povečana na vrednost $\mu_{cse} = 300{,}0\ zahtev/s$.

Ugotoviti želimo:

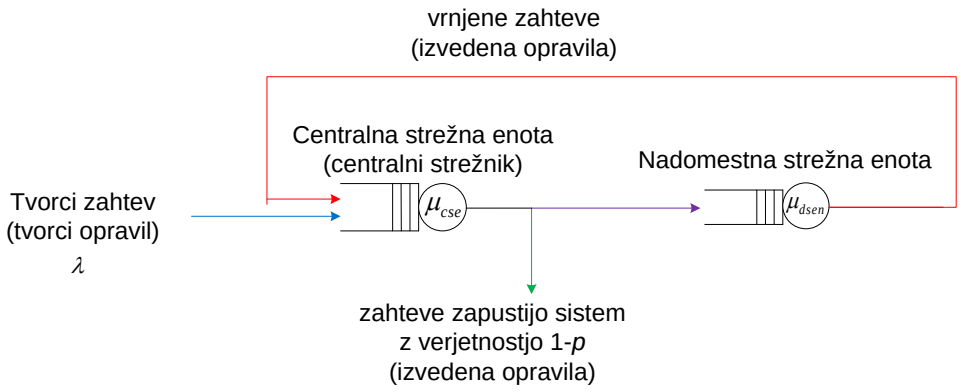
1. vpliv vhodne intenzivnosti in razvrščanje po principu najkrajša čakalna vrsta najprej na potek povprečnega odzivnega časa strežne mreže; upoštevamo različno zmogljive delovne strežne enote;
2. vpliv vhodne intenzivnosti in verjetnosti vračanja $p = 0{,}3$ na potek povprečne dolžine čakalnih vrst centralne strežne enote in delovnih strežnih enot; upoštevamo različno zmogljive delovne strežne enote;
3. vpliv vhodne intenzivnosti in verjetnosti vračanja $p = 0{,}3$ na potek povprečne prepustnosti strežne mreže in delovnih strežnih enot; upoštevamo različno zmogljive delovne strežne enote.

Denimo, da je verjetnost vračanja $p = 0{,}0$. Primerjajmo potek povprečnega odzivnega časa pri eksperimentu V in VI. Opazimo, da le s spremembo načina razvrščanja po principu najkrajša vrsta najprej dosežemo veliko kasnejše nasičenje strežne mreže. Pri eksperimentu V se ta pojavi blizu vhodne intenzivnosti $\lambda = 30{,}0\ zahtev/s$ (slika 4.33), medtem ko se tu nasičenje pojavi blizu vhodne intenzivnosti $\lambda = 90{,}0\ zahtev/s$ (slika 4.36). Torej kar 2-kratno povečanje uporabnega intervala.



Slika 4.36: Potek povprečnega odzivnega časa strežne mreže eksperimenta VI

Sprememba razvrščanja omogoča povsem uravnoteženo obremenitev delovnih strežnih enot. Dolžine čakalnih vrst se razlikujejo za največ 1 zahtevo (podobno kot eksperimentu IV, na sliki 4.32) in to pomeni, da delovne strežne enote preidejo v nasičenje hkrati. Takšen način delovanja omogoča, da delovne strežne enote lahko v principu delovanja strežne mreže obravnavamo s pomočjo ene nadomestne delovne strežne enote (slika 4.37). Njena dolžina čakalne vrste je enaka vsoti dolžin čakalnih vrst posameznih delovnih strežnih enot, njena zmogljivost μ_{dsen} pa enaka vsoti zmogljivosti posameznih strežnih enot.



Slika 4.37: Nadomestna delovna strežna enota v strežni mreži

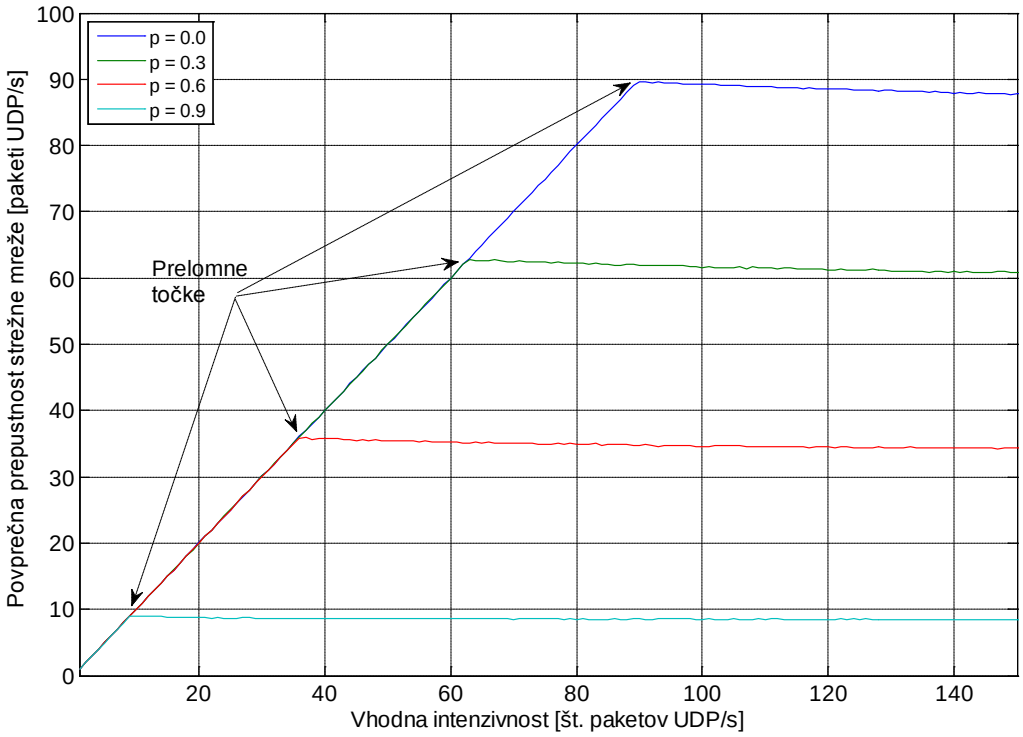
Strežna mreža preide v nasičenje zaradi nasičenja nadomestne delovne strežne enote. To se pojavi v trenutku, ko je njena zmogljivost enaka vhodni obremenitvi λ in povratni obremenitvi zanke:

$$\mu_{dsen} = \lambda + p\lambda + p^2\lambda + p^3\lambda + \cdots = \lambda + \sum_{i=1}^{\infty}(p^i\lambda) = \frac{\lambda}{1-p}.$$

Vhodno intenzivnost, pri kateri pride do nasičenja, določa izraz

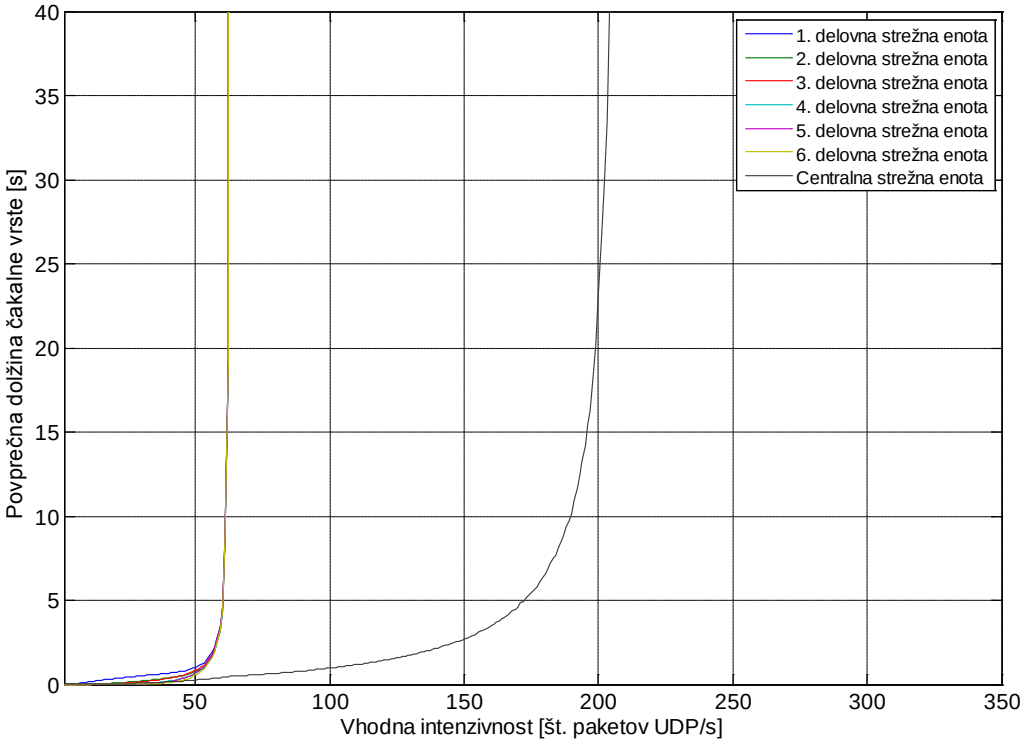
$$\lambda = \mu_{dsen}(1-p).$$

Za verjetnost vračanja zahtev $p = 0{,}0$, se nasičenje strežne mreže oziroma delovnih strežnih enot pojavi blizu vrednosti $\lambda = 90{,}0$, za verjetnost $p = 0{,}3$ blizu vrednosti $\lambda = 63{,}0$, za verjetnost $p = 0{,}6$ blizu vrednosti $\lambda = 36{,}0$ in za verjetnost $p = 0{,}9$ blizu vrednosti $\lambda = 9{,}0$. Pri teh vrednostih prihaja do prelomnih točk (slika 4.38). Strežna mreža doseže svojo največjo prepustnost le za kratek čas, nato prične zaradi nadaljnjega naraščanja vhodne intenzivnosti in s tem dodatnega obremenjevanja centralne strežne enote padati.



Slika 4.38: Potek povprečne prepustnosti strežne mreže eksperimenta VI

Slika 4.39 prikazuje potek povprečne dolžine čakalnih vrst delovnih strežnih enot in centralne strežne enote pri verjetnosti vračanja $p = 0{,}3$. Očitno je, da delovne strežne enote preidejo v nasičenje hkrati, in sicer blizu izračunane vrednosti vhodne intenzivnosti $\lambda = 63{,}0$.



Slika 4.39: Potek povprečnih dolžin čakalnih vrst delovnih vozlišč in centralnega strežnika eksperimenta VI

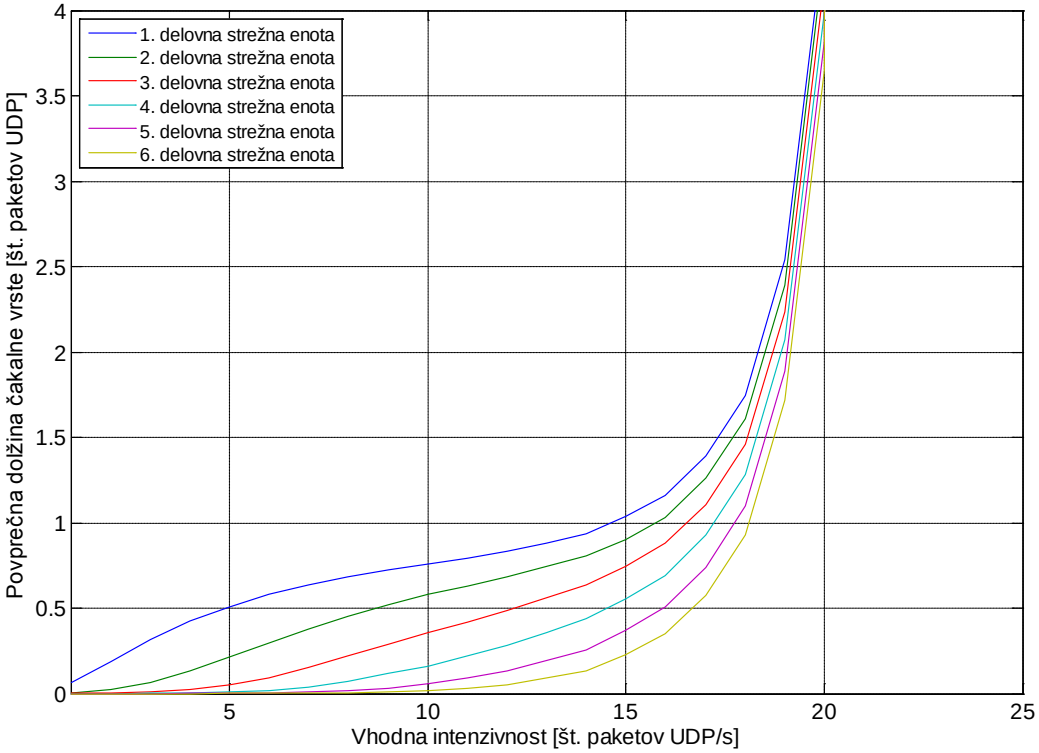
Centralna strežna enota preide v nasičenje, ko vhodna intenzivnost in zmogljivost nadomestne delovne strežne enote presežeta zmogljivost centralne strežne enote. Takrat mora veljati izraz

$$\mu_{cse} = \lambda + \mu_{dsen} .$$

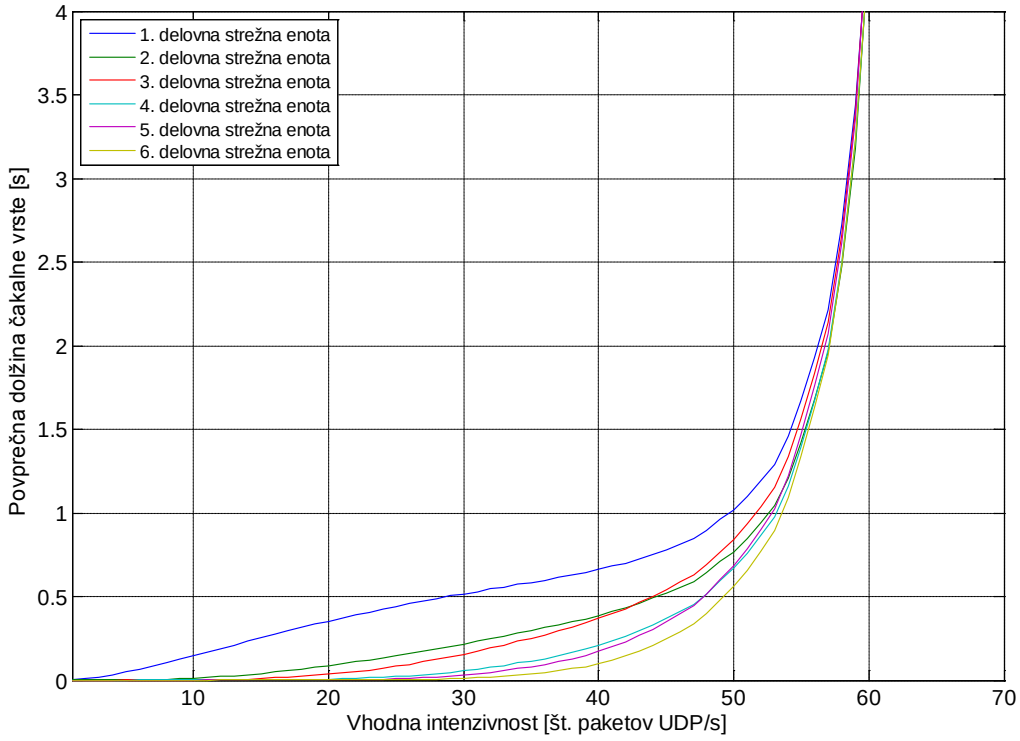
Točka nasičenja se tako pojavi blizu vrednosti vhodne intenzivnosti, kot določa izraz

$$\lambda = \mu_{cse} - \mu_{dsen} = 300{,}0 - 90{,}0 = 210{,}0 .$$

Zanimiva je tudi primerjava med podrobnejšim potekom povprečnih dolžin čakalnih vrst delovnih strežnih enot eksperimenta IV in podrobnejšim potekom povprečnih dolžin čakalnih vrst delovnih strežnih enot eksperimenta VI. Pri razvrščanju po principu najkrajša čakalna vrsta najprej pričnemo z obremenjevanjem najzmogljivejših delovnih strežnih enot in nadaljujemo vse do najmanj zmogljivih. Če so te enako zmogljive (eksperiment IV), potem je dolžina čakalne vrste prve izbrane delovne strežne enote pri postopku razvrščanja vedno večja od dolžin čakalnih vrst ostalih delovnih strežnih enot (modra barva, slika 4.40). Naslednja izbrana delovna strežna enota ima spet daljšo čakalno vrsto od vseh ostalih delovnih strežnih enot, ki ji sledijo (zelena barva, slika 4.40). Podobno velja za ostale dolžine čakalnih vrst delovnih strežnih enot. Pri eksperimentu VI je obremenitev prve delovne strežne enote sicer podobna, vendar prihaja do prepletanja pri ostalih delovnih strežnih enotah (slika 4.41). Obnašanje je zaradi različnih zmogljivosti delovnih strežnih enot veliko bolj dinamično in potrebno je več prilagajanja za doseganje enakomerne obremenitve vseh delovnih strežnih enot.

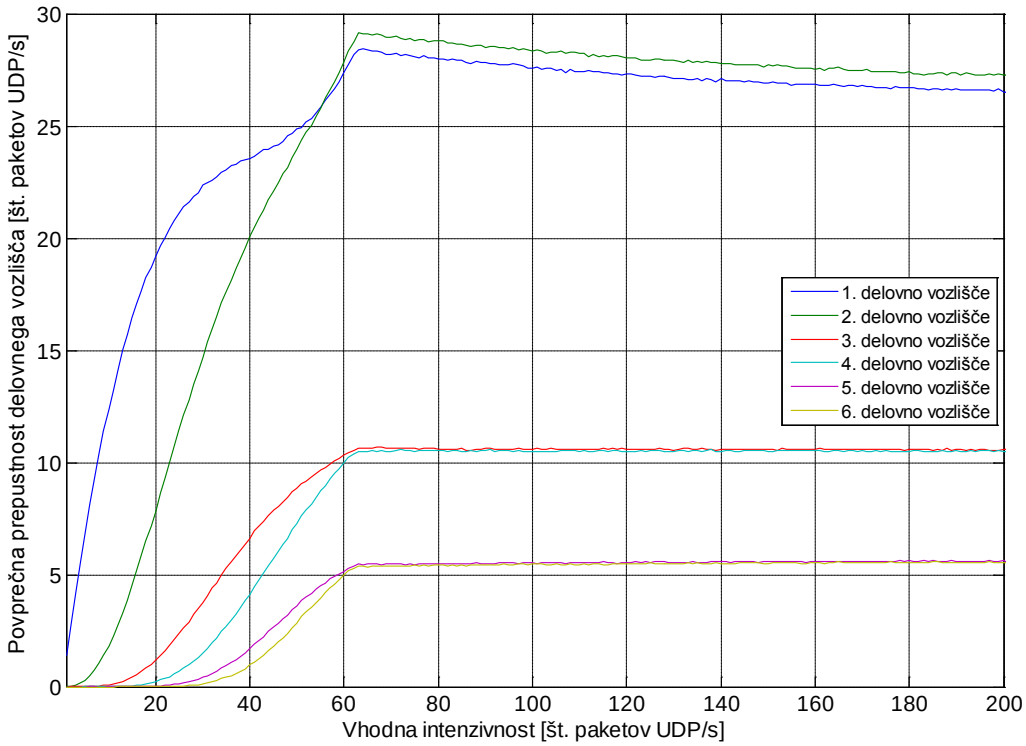


Slika 4.40: Podroben potek povprečne dolžine čakalnih vrst delovnih strežnih enot eksperimenta IV



Slika 4.41: Podroben potek povprečne dolžine čakalnih vrst delovnih strežnih enot eksperimenta VI

Presenetljive grafične rezultate smo zasledili pri poteku prepustnosti vseh šestih delovnih strežnih enot. Slika 4.42 kaže, da delovne strežne enote 3, 4, 5 in 6 pri verjetnosti vračanja $p = 0{,}3$, presežejo svojo največjo teoretično prepustnost. Prva in druga delovna strežna enota pa svoje največje prepustnosti nikoli ne dosežeta. Videti je, kot da srednje zmogljive in najmanj zmogljive delovne strežne enote "odžirajo" zmogljivost najzmogljivejšim delovnim strežnim enotam. Tak učinek je seveda zgolj navidezen.

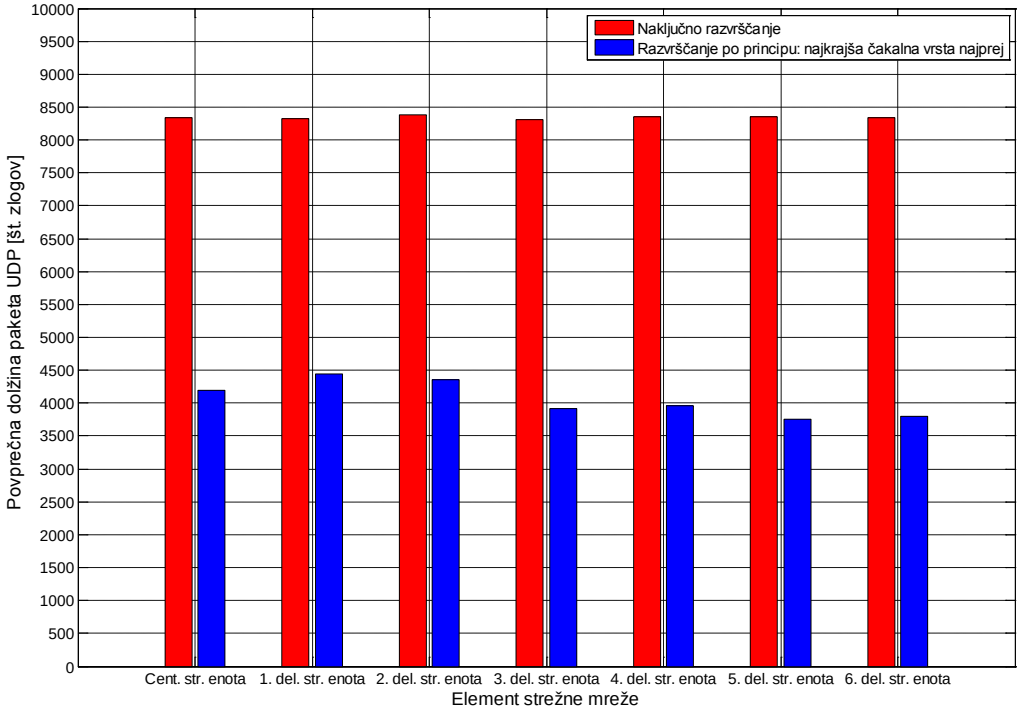


Slika 4.42: Potek povprečne prepustnosti delovnih strežnih enot eksperimenta VI

Zmogljivost strežnih enot je določena s hitrostjo povezave med vozliščema, ki strežno enoto definirata oziroma tvorita. Razvrščanje po principu najkrajša čakalna vrsta najprej povzroči, da se povezavam z večjo prepustnostjo dodeljujejo daljši paketi UDP. S postopkom simulacije eksperimentov V in VI pri vhodni intenzivnosti $\lambda = 100{,}0$ in verjetnosti vračanja $p = 0{,}0$, smo ugotovili povprečne dolžine paketov UDP, ki jih prikazujeta Tabela 20 in slika 4.43.

Strežna enota vozlišče	**Povprečna dolžina paketa UDP (eksperiment V)**	**Povprečna dolžina paketa UDP (eksperiment VI)**
Centralna strežna enota	8341,7 zlogov	4189,8 zlogov
1. delovna strežna enota	8323,0 zlogov	4440,8 zlogov
2. delovna strežna enota	8378,9 zlogov	4350,0 zlogov
3. delovna strežna enota	8307,7 zlogov	3913,7 zlogov
4. delovna strežna enota	8350,9 zlogov	3953,4 zlogov
5. delovna strežna enota	8358,2 zlogov	3752,9 zlogov
6. delovna strežna enota	8331,5 zlogov	3798,8 zlogov

Tabela 20: Povprečne dolžine paketov UDP na strežnih enotah za eksperiment V in VI

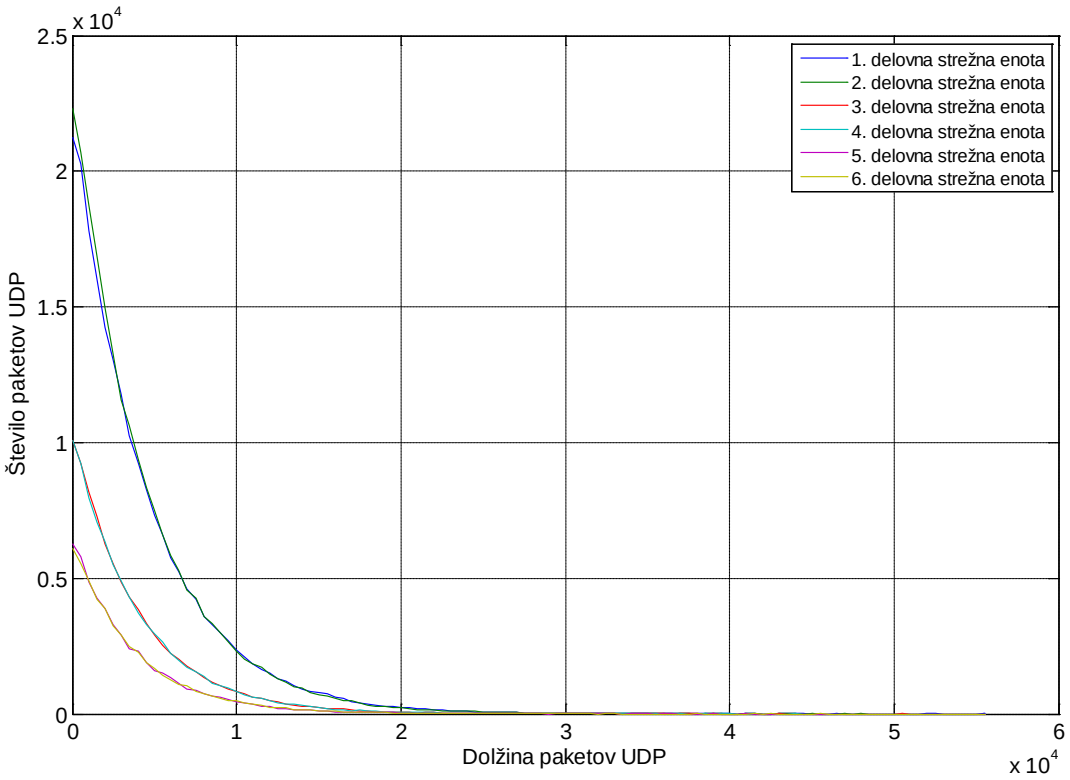


Slika 4.43: Povprečne dolžine paketov UDP na delovnih strežnih enotah pri obeh načinih razvrščanja (eksperiment V in VI)

Prvi dve povezavi ($V(2) \rightarrow V(3)$ in $V(2) \rightarrow V(4)$) tvorita najzmogljivejši delovni strežni enoti (1. in 2.) in preko teh se prenese največji delež daljših paketov UDP. Na prvi povezavi se prenašajo paketi s povprečno dolžino 4440,8 zlogov in na drugi paketi s povprečno dolžino 4350,0 zlogov. Povezave $V(2) \rightarrow V(5)$ in $V(2) \rightarrow V(6)$ tvorijo srednje zmogljive delovne strežne enote (3. in 4.). Preko teh dveh povezav se prenašajo paketi UDP, ki imajo povprečno dolžino 3913,7 zlogov oziroma 3953,4 zlogov. Podobno velja za povezavi $V(2) \rightarrow V(7)$ in $V(2) \rightarrow V(8)$, ki definirata delovni strežni enoti 5 in 6. Po teh se pošiljajo paketi UDP z najmanjšo povprečno dožino 3752,9 zlogov oziroma 3798,8 zlogov.

Pri eksperimentu V so vse strežne enote (vključno s centralno strežno enoto) zaradi izbranega naključnega razvrščanja prejemale v povprečju približno enako dolge pakete UDP. Povprečna dolžina sega od 8307,7 zlogov do 8378,9 zlogov. Z izbiro načina razvrščanja po principu najkrajša čakalna vrsta najprej pa se takšno povprečje poruši. Posledica tega je upad zmogljivosti najzmogljivejšim delovnim strežnim enotam. Večino časa se po povezavah prenaša sicer manj paketov UDP, kot bi se jih v resnici moralo, a so zato daljši. Ostalim delovnim strežnim enotam pa se zmogljivost navidezno poveča, saj prejmejo več paketov UDP, kot bi jih morali sicer, a so ti krajši.

Če prejete pakete UDP, ki jih sprejemajo posamezna vozlišča (vozlišča 3, 4, 5, 6, 7 in 8), razvrstimo v skupine glede na dolžino, lahko potrdimo, da se porazdelitev časa procesiranja delovnih strežnih enot ohranja. Slika 4.44 prikazuje eksponentno funkcijo gostote verjetnosti časa procesiranja za vse delovne strežne enote. To pomeni, da se porazdelitev na posameznih delovnih strežnih enotah ne spreminja. S postopkom razvrščanja tudi ne vplivamo na porazdelitev časa prihodov paketov UDP na posamezne delovne strežne enote, zato tudi to ostaja nespremenjeno.



Slika 4.44: Gostota verjetnosti dolžin paketov UDP na vhodu delovnih strežnih enot eksperimenta VI

Rezultate za strežno mrežo eksperimenta VI prikazuje Tabela 21.

Izhodni parameter	Vrednost
$\overline{T}$	60 sekund
Interval uporabnosti pri $p = 0{,}0$	Model je uporaben na intervalu $\lambda = 1{,}0$ do $\lambda = 91{,}0$
Interval uporabnosti pri $p = 0{,}3$	Model je uporaben na intervalu $\lambda = 1{,}0$ do $\lambda = 63{,}0$
Interval uporabnosti pri $p = 0{,}6$	Model je uporaben na intervalu $\lambda = 1{,}0$ do $\lambda = 36{,}0$
Interval uporabnosti pri $p = 0{,}9$	Model je uporaben na intervalu $\lambda = 1{,}0$ do $\lambda = 9{,}0$
$\overline{L_{čv_{cse}}}$	0,47 paketov UDP
$\overline{L_{čv_{dse1}}}$	556.14 paketov UDP
$\overline{L_{čv_{dse2}}}$	555.87 paketov UDP
$\overline{L_{čv_{dse3}}}$	556.16 paketov UDP
$\overline{L_{čv_{dse4}}}$	556.01 paketov UDP
$\overline{L_{čv_{dse5}}}$	556.18 paketov UDP
$\overline{L_{čv_{dse6}}}$	556.06 paketov UDP
$\bar{X}$ pri $p = 0{,}0$	Do 89,5 paketov UDP/s
$\bar{X}$ pri $p = 0{,}3$	Do 62,7 paketov UDP/s
$\bar{X}$ pri $p = 0{,}6$	Do 35,8 paketov UDP/s
$\bar{X}$ pri $p = 0{,}9$	Do 8,9 paketov UDP/s

Tabela 21: Rezultati eksperimenta V

Gotovo lahko trdimo, da način razvrščanja po principu najkrajša čakalna vrsta najprej zagotavlja, da se zahtevnejša opravila izvajajo na zmogljivejših delovnih vozliščih, manj zahtevna pa na manj zmogljivih. Sistema BOINC in Condor lahko, v okolju z različno zmogljivimi delovnimi vozlišči, tak način delovanja dosežeta z ustreznimi opisi zahtev delovnih vozlišč in opravil (na primer opisi ClassAd). Razvrščevalnik te opise upošteva in poskrbi, da se zahtevnejša opravila vedno pošiljajo zmogljivejšim delovnim vozliščem, manj zahtevna opravila pa manj zmogljivim. V sistemu Condor lahko uravnoteženo obremenitev dosežemo tudi tako, da centralni upravitelj nenehno spremlja obremenitev delovnega vozlišča (parameter LoadAvg v opisu ClassAd). Prostovoljni sistem bi bil tako optimalno obremenjen. Množico delovnih vozlišč lahko v tem primeru za lažjo obravnavo nadomestimo z nadomestnim delovnim vozliščem. Njegove procesne in pomnilniške zmogljivosti lahko obravnavamo kot vsoto posameznih procesnih in pomnilniških zmogljivosti delovnih vozlišč.

5 Zaključek

V magistrski nalogi smo jasno postavili ločnico med vzporednimi in porazdeljenimi sistemi. Preučili smo dve najpogostejši obliki prostovoljnih sistemov: BOINC in Condor. Z ustreznimi poenostavitvami teh dveh okolij smo zasnovali model prostovoljnega sistema, implementacijo pa izvedli s pomočjo odprtokodnega orodja ns-2. Glavni cilj je bil izvesti simulacijo v sklopu različnih eksperimentov, ki bi omogočala vpogled v lastnosti in napovedovanje obnašanja prostovoljnega sistema.

Vsak eksperiment je določala množica izbranih parametrov, ki je jasno opredelila simulacijski model oziroma strežno mrežo. Opazovali smo vse pomembnejše karakteristične parametre strežne mreže: odzivni čas, dolžine čakalnih vrst, prepustnost in interval uporabnosti. Pri vseh eksperimentih se je izkazalo, da z naraščanjem vhodne intenzivnosti opravil narašča tudi obremenitev prostovoljnega sistema. Pri določeni vhodni intenzivnosti bi prostovoljni sistem postal preobremenjen – prehod v nasičenje. V takšnem stanju bodo uporabniki čakali na rezultate svojih oddanih opravil veliko dlje kot običajno. Če centralno vozlišče prostovoljnega sistema ni dovolj zmogljivo, lahko celotni sistem hitro preide v nasičenje, četudi so delovna vozlišča dovolj zmogljiva. Velika verjetnost vračanja opravil in dodajanje večjega števila delovnih vozlišč ali celo zmogljivejših delovnih vozlišč povzročata še dodatno obremenitev centralnega vozlišča. Če je zmogljivost centralnega vozlišča dovolj velika v primerjavi s skupno zmogljivostjo delovnih vozlišč, lahko ozko grlo tvorijo delovna vozlišča. Prevelika obremenitev centralnega vozlišča oziroma delovnih vozlišč povzroči, da se lahko njihove čakalne vrste opravil močno povečajo. Hitro povečevanje čakalnih vrst je še posebno izrazito, kadar so zmogljivosti delovnih vozlišč različne. Pri naključnem razvrščanju in predpostavki, da je centralno vozlišče dovolj zmogljivo, preidejo v nasičenje najprej najmanj zmogljiva, nato srednje zmogljiva in nazadnje najzmogljivejša delovna vozlišča. Pri tem tipu razvrščanja je obremenitev delovnih vozlišč neenakomerna in povzroča hitrejše nasičenje sistema. Izkaže se, da lahko z alternativnim, dinamičnim razvrščanjem (razvrščanje v najkrajšo čakalno vrsto naprej) bistveno izboljšamo izkoristek delovnih vozlišč in uravnotežimo obremenitev prostovoljnega sistema. S takšnim tipom razvrščanja lahko v sistemih, kjer so si zmogljivosti delovnih vozlišč približno enakovredne, pričakujemo enakomerno porazdelitev zahtevnosti opravil po vseh delovnih vozliščih. Povsem nepričakovano obnašanje simulacijskega modela pa se je pokazalo pri zadnjem eksperimentu. Izkazalo se je namreč, da je prepustnost nekaterih delovnih strežnih enot celo višja od njihove največje teoretično možne prepustnosti. Učinek je seveda le navidezen. Omogočil pa je odkriti dejstvo, da v sistemih z delovnimi vozlišči različnih zmogljivosti lahko pričakujemo, da bodo zmogljivejša delovna vozlišča obremenjena z zahtevnejšimi, manj zmogljiva pa z manj zahtevnimi opravili. To pomeni, da delovna vozlišča v primeru razvrščanja po principu najkrajša vrsta najprej lahko že samodejno s svojimi lastnostmi določajo, kako zahtevna opravila bodo izvajala.

Model prostovoljnega sistema sicer ne odraža popolnega stanja realnega prostovoljnega sistema, njegova simulacija pa vendarle podaja dovolj dobre približke, s pomočjo katerih lahko predvidimo obnašanje prostovoljnega sistema pri spremembah njegovih parametrov. Simulacijsko orodje ns-2 se je izkazalo za zelo prožno in vsestransko. Poleg osnovnega orodja je na voljo tudi posebno animacijsko orodje NAM, ki je z vizualno podobo simulacije omogočala preverbo izvajanja simulacije. Po drugi strani pa je bil postopek simulacije, še posebno pri vhodnih intenzivnostih, ki povzročijo nasičenje, precej dolgotrajen in zahteven po

virih. Tipična velikost datoteke za sledenje dogodkov je bila okoli vrednosti 3 GB. Datoteka za sledenje, ki jo potrebuje orodje NAM, pa okoli 1,5 GB. Bistveno hitrejše izvajanje simulacije smo lahko dosegli z izklopom sledenja izvajanja simulacije, ki jo uporablja orodje NAM. Pri majhnih količinah delovnega pomnilnika je prihajalo tudi do pogostejše aktivnosti trdega diska in posledično upočasnjenega izvajanja simulacije. Eksperimenti se zato niso izvajali s pomočjo običajnega namiznega računalnika, ampak na zmogljivejših strežniških rezinah. Alternativno orodje, ki bi bilo primerno za izvajanje te vrste simulacije, je Matlab in OMNeT++. Matlab ima sicer uporabniku zelo prijazno snovanje diskretnih modelov, vendar je izvajanje simulacije nad modeli dokaj počasno. OMNeT++ pa predstavlja ugoden kompromis med uporabniškim vmesnikom in hitrostjo delovanja simulacije.

Simulacijski model bi lahko izboljšali na več načinov:

1 z uvedbo različnih razredov opravil bi lahko povsem ločeno analizirali izvajanje posameznih vrst opravil. Omogočalo bi tudi niteno interpretacijo opravil. Rešitev je v dodajanju posebnega polja v paketih UDP, ki določa vrsto oziroma razred, ki mu paket UDP pripada;
2 lastnosti delovnih strežnih enot lahko dopolnimo z verjetnostmi vračanja zahtev, ki so značilne za posamezne delovne strežne enote. Rešitev je v ločeni hrambi verjetnosti vračanja paketov UDP za vsako delovno strežno enoto posebej;
3 upoštevanje večjedrnih delovnih strežnih enot. Rešitev je v grupiranju delovnih strežnih enot s pomočjo dodatne podatkovne strukture v navidezno delovno strežno enoto. Paket UDP se znotraj grupe pošlje prvi prosti delovni strežni enoti;
4 preverili bi lahko tudi drugačne tehnike razvrščanja, predvsem takšne, ki jih analitično ni mogoče dobro ovrednotiti.

Rezultati simulacijskega modela so lahko uporabni predvsem pri proučevanju posledic sprememb v prostovoljnem sistemu. Na primer: kakšen učinek povzroči odstranitev določenega števila delovnih vozlišč s podano zmogljivostjo oziroma kako se v tem primeru spremeni zmogljivost oziroma prepustnost sistema, koliko dlje bomo čakali na odziv sistema in podobno. Ni pa nujno, da se omejujemo le na takšne vrste sistemov. Pomembno je le, da je v obravnavanem sistemu prisotna centralna enota in ena ali več delovnih enot. Delovne enote sprejemajo naloge iz centralne enote, jih izvedejo, rezultate pa pošljejo nazaj na centralno enoto.

LITERATURA IN VIRI

[1] Shameem Akhter, Jason Roberts, *Multi-core programming*, Intel Press, 2006.

[2] David P. Anderson, "Local Scheduling for Volunteer Computing," v zborniku Parallel and Distributed Processing Symposium, IPDPS 2007. IEEE International, str. 1−8, 2007.

[3] David P. Anderson, "BOINC: a system for public-resource computing and storage," v zborniku Proceedings 5th IEEE/ACM International Workshop on grid computing, zv. 4−10, nov. 2004.

[4] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, "SETI@home An Experiment in Public-Resource Computing," Communications of the ACM, št. 11, zv. 45, str. 56−61, 2002.

[5] David P. Anderson, Gilles Fedak, "The Computational and Storage Potential of Volunteer Computing," v zborniku Sixth IEEE International Symposium on Cluster Computing and the Grid, št. 1, str. 73−80, maj 2006.

[6] David P. Anderson, Eric Korpela, Rom Walton, "High-Performance Task Distribution for Volunteer Computing," v zborniku First International Conference on e-Science and Grid Computing, str. 196−203, 2005.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, "A View of Cloud computing," Communications of the ACM, št. 4, zv. 53, str. 50–58, apr. 2010.

[8] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, Katherine Yelickad, "A view of the Parallel Computing Landscape," Communications of the ACM, št. 10, zv. 52, str. 56−67, okt. 2009.

[9] Kevin J. Baker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Mike Lang, Scott Pakin, Jose C. Sancho, "Entering the Petaflop Era: The architecture and performance of Roadrunner," v zborniku Conference on High Performance Computing, Networking, Storage and Analysis, str. 1−11, nov. 2008.

[10] Jerry Banks, *Handbook of simulation; Principles, Methodology, Advances, applications and Practice*, Wiley, 1998, pogl. 1, 2, 10.

[11] Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Mike Lang, Scott Pakin, Jose Carlos Sancho, "A Performance evaluation of the nehalem quad-core porcessor for scientific computing," Parallel Processing Letters, št. 4, zv. 18, str. 453–469, 2008.

[12] Babak Behsaz, Pooya Jaferian, Mohammad Reza Meybodi, "Comparison of Global Computing with Grid Computing," v zborniku Seventh International Conference on

Parallel and Distributed Computing, Applications and Technologies, str. 531−534, dec. 2006.

[13] C. Gordon Bell, "Multis: A New Class of Multiprocessor computers," v zborniku Science, zv. 228, str. 462−467, apr. 1985.

[14] Gordon Bell, "Ultracomputer: A teraflop before its time," Communications of ACM, št. 8, zv. 35, str. 27−47, avg. 1992.

[15] Gordon Bell, Jim Gray, "What's Next in High-Performance Computing," Communications of the ACM, št. 2, zv. 45, str. 27−29, feb. 2002.

[16] Rajkumar Buyya, James Broberg, Andrzej Goscinski, *Cloud computing, Principles and Paradigms*, Wiley, 2011, pogl. 1.

[17] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Future Generation Computer Systems, št. 6, zv. 25, str. 599−616, jun. 2009.

[18] Rene J. Chevance, *Server Architectures; Multiprocessors, Clusters, Parallel systems, WEB servers, Storage Solutions*, Elsevier Digital Press, 2005, pogl. 1, 4, 5, 7, 12.

[19] Vincenzo D. Cunsolo, Salvatore Distefano, Antonio Puliafito, Marco Scarpa, "Volunteer computing and desktop Cloud: The Cloud@home Paradigm," v zborniku Eighth IEEE International Symposium on Network Computing and Applications, str. 134−139, jul. 2009.

[20] Subrata Dasgupta, "A hierarchical taxonomic system for computer architectures," Computer, št. 3, zv. 23, str. 64−74, mar. 1990.

[21] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, Dean M. Tullsen, "Simultaneous multithreading: A platform for next-generation processors," IEEE Micro, št. 5, zv. 17, str. 12−19, sept./okt. 1997.

[22] Hesham El-Rewini, Mostafa Abd-El-Barr, *Advanced computer architecture and parallel processing*, Wiley-Interscience, 2005, pogl. 1−5, 9−10.

[23] Trilce Estrada, Michela Taufer, Kevin Reed, David P. Anderson, "EmBOINC: An emulator for performance analysis of BOINC projects," v zborniku IEEE International Symposium on Parallel&Distributed Processing, str. 1−8, 23−29, maj 2009.

[24] Alexandra Fedorova, Juan Carlos Saez,Daniel Shelepov, Manuel Prieto, "Maximizing Power Efficiency with Asymmetric Multicore Systems," Communications of the ACM, št. 12, zv. 52, str. 48−57, nov. 2009.

[25] Renato J. Figueiredo, Peter A. Dinda, Jose A. B. Fortes, "A Case For Grid Computing On Virtual Machines," v zborniku Proceedings of 23rd IEEE International Conference on Distributed Computing Systems, str. 550−559, maj 2003.

[26] Ian Foster, Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998, pogl. 1, 2.

[27] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam, *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*, 1994, pogl. 1.

[28] K. Gopal Gopalan, *Introduction To Digital Microelectronic Circuits*, Irwin, 1996, pogl. 3.

[29] William Gropp, Ewig Lusk, Anthony Skjellum, *Using MPI, Portable parallel programming with message-passing interface*, MIT Press, 1999, pogl. 1.

[30] William Gropp, Ewing Lusk, Thomas Sterling, *Beowulf Cluster Computing with Linux*, London: MIT Press, 2003, pogl. 1, 2, 9, 14.

[31] Brian Hayes, "Cloud Computing," Communications of the ACM, št. 7, zv. 51, str. 9–11, jul. 2008.

[32] Roger W. Hockney, "Classification and evaluation of parallel computer systems," v zborniku 4th International DFVLR Seminar on Foundations of Engineering Sciences on Parallel Computing in Science and Engineering, str. 13−25, 1988.

[33] Michael Huebner, Juergen Becker, *Multiprocessor system-on-chip*, Springer, 2011, pogl. 1, 9.

[34] Teerawat Issariyakul, Ekram Hossain, *Introduction to Network Simulator NS2*, Springer, 2009, pogl. 1−9, 13, dodatek A.

[35] Raj Jain, *The art of computer systems performance analysis; Techniques for experimental design, measurement, simulation and modeling*, Wiley, 1991, pogl. 1−3, 24−25, 30−33.

[36] C. R. Johns, D.A. Brokenshire, "Introduction to the Cell Broadband Engine Architecture," v zborniku IBM Journal of Research and Development, št. 5, zv. 51, str. 503−519, sept. 2007.

[37] Stephen W. Keckler, Kunle Olukotun, H. Peter Hofstee, *Multicore processors and systems*, Springer, 2009, pogl. 1, 6, 9.

[38] Leonard Kleinrock, *Queueing systems*, Wiley-Interscience, 1975, pogl. 1, 2, 3.

[39] Dušan Kodek, *Arhitektura in organizacija računalniških sistemov*, Bi-Tim, 2008, pogl. 2, 3, 7, 8.

[40] James Larus, "Spending Moore's Dividend," Communications of the ACM, št. 5, zv. 52, str. 62−69, maj 2009.

[41] Claudia Leopold, *Parallel and Distributed Computing*, Wiley-Interscience, 2000, pogl. 1−6.

[42] Michael J. Litzkow, Miron Livny, Matt W. Mutka, "Condor - A Hunter of Idle Workstations," v zborniku Eighth International Conference on Distributed Computing Systems, str. 104−111, jun. 1988.

[43] O. Lodygensky, G. Fedak, V. Neri, M. Livny, D. Thain, "XtremWeb and Condor: Sharing Resources Between Internet Connected Condor Pool," v zborniku In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, str. 382−389, maj 2003.

[44] Ami Marowka, "Parallel computing on any desktop," Communications of the ACM, št. 9, zv. 50, str. 75−78, sept. 2007.

[45] Cary Millsap, "Thinking clearly about Performance, Part 1," Communications of the ACM, št. 9, zv. 53, str. 55−60, sept. 2010.

[46] Cary Millsap, "Thinking clearly about Performance, Part 2," Communications of the ACM, št. 10, zv. 53, str. 39−45, okt. 2010.

[47] Gordon E. Moore, "Craming more components onto integrated circuits," Electronics, št. 8, zv. 38, str. 114−117, apr. 1965.

[48] Dan Nessett, "Massively Distributed Systems: Design Issues and Challenges," v zborniku WOES'99 Proceedings of the Workshop on Embedded Systems on Workshop on Embedded Systems, str. 8−8, 1999.

[49] Peter S. Pacheco, *Parallel programming with MPI*, Morgan Kaufman, 1997, pogl. 1, 2.

[50] M. Parashar, C. Lee, "Grid Computing - Introduction and Overview," *v zborniku* Proceedings of the IEEE, Special Issue on Grid Computing, IEEE Press, št. 3, zv. 93, mar. 2005.

[51] Gregory F. Pfister, *In search of clusters*, Prentice Hall, 1997, pogl. 1, 3, 4.

[52] Pawel Plaszczak, Richard Wellner, Jr., *Grid computing, the savvy manager's guide*, Morgan Kaufmann, 2006, pogl. 1, 2, 5.

[53] Dongyu Qiu, R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," v zborniku Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communication, št. 4, zv. 34, str. 367−378, okt. 2004.

[54] John W. Rittinghouse, James F. Ransome, *Cloud computing, Implementation, Management and security*, CRC Press, 2010, pogl. 1, 2.

[55] Rodrigo Rodrigues, Peter Druschel, "Peer-to-Peer Systems," Communications of the ACM, št. 10, zv. 53, str. 72−82, okt. 2010.

[56] Seyed H. Roosta, *Paralell processing and parallel algorithms, theory and computation*, Springer, 2000, pogl. 1, 2, 3.

[57] F. D. Sacerdoti, S. Chandra, and K. Bhatia, "Grid systems deployment & management using Rocks," v zborniku In Proceedings of the 2004 IEEE International Conference on Cluster Computing, str. 337−345, 2004.

[58] Sriya Santhanam, Pradheep Elango, Andrea Arpaci Dusseau, Miron Livny, "Deploying Virtual Machines as Sandboxes for the Grid," v zborniku WORLDS'05 Proceedings of the 2nd conference on Real, Large Distributed Systems, str. 7−12, 2005.

[59] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications," v zborniku First International Conference on Peer-to-Peer Computing, str. 101−102, avg. 2002.

[60] Yuan Shi, "Reevaluating Amdahl's Law and Gustafson's Law," okt. 1996. Dostopno na: http://www.cis.temple.edu/~shi/docs/amdahl/amdahl.html (ogled spletne strani: junij 2010)

[61] Howard Jay Siegel, "The Universality of Various Types of SIMD Machine lnterconnection Networks," v zborniku Proceedings of 4th Annual Symposium on Computer Architecture, št. 7, zv. 5, str. 70−79, mar. 1977.

[62] M. Singhal, N. Shivaratri, *Advanced concepts in operating systems*, McGraw-Hill, 1994, pogl. 1, 4, 5.

[63] Joseph Sloan, *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI*, O'Reilly Media, 2004, pogl. 1, 5−7, 13.

[64] Joachim Stolze, Dieter Suter, *Quantum computing: a short course from theory to experiment*, Wiley, 2004, pogl. 1.

[65] Michela Taufer, Andre Kerstens, Trilce Estrada, David Flores, Patricia J. Teller, "SimBA: a Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects," v zborniku 21st International Workshop on Principles of Advanced and Distributed Simulation, str. 189−197, jun. 2007.

[66] Douglas Thain, Todd Tannenbaum, Miron Livny, *Grid computing – Making the global infrastructure a reality*, John Wiley & Sons, 2003, pogl. 11.

[67] Douglas Thain, Todd Tannenbaum, Miron Livny, "Distributed Computing in Practice: The Condor Experience," v zborniku Concurrency and Computation: Practice and Experience, št. 2-4, zv. 17, str. 323−356, feb. 2005.

[68] Yuan Xie, Jason Cong, Sachin Sapatnekar, *Three-dimensional integrated circuit design*, Springer, 2010, pogl. 1.

[69] http://www.mcs.anl.gov/research/projects/mpich2/about/index.php?s=about (ogled spletne strani: februar 2011)

[70] http://www.open-mpi.org/ (ogled spletne strani: februar 2011)

[71] http://www.mosix.org/txt_about.html
(ogled spletne strani: februar 2011)

[72] http://idea.uab.es/mcreel/ParallelKnoppix/
(ogled spletne strani: februar 2011)

[73] http://public.eu-egee.org/intro/index.html
(ogled spletne strani: december 2010)

[74] http://boinc.berkeley.edu/wiki/How_BOINC_works
(ogled spletne strani: december 2010)

[75] http://www.cs.wisc.edu/condor/manual/v7.5/
(ogled spletne strani: december 2010)

[76] "An Experimental Study of the Skype Peer-to-Peer VoIP System". Dostopno na:
http://research.microsoft.com/en-us/um/people/saikat/pub/iptps06-skype/
(ogled spletne strani: februar 2011)

[77] http://setiathome.berkeley.edu/top_users.php
(ogled spletne strani: december 2010)

[78] http://www.nordugrid.org/arc/
(ogled spletne strani: februar 2011)

[79] http://www.unicore.eu/
(ogled spletne strani: februar 2011)

[80] http://www.globus.org/toolkit/about.html
(ogled spletne strani: februar 2011)

[81] http://www-formal.stanford.edu/jmc/
(ogled spletne strani: junij 2010)

[82] "The science of SETI@home". Dostopno na:
http://setiathome.berkeley.edu/sah_about.php
(ogled spletne strani: december 2010)

[83] "Nvidia Tesla, GPU Computing Technical Brief". Dostopno na:
http://www.nvidia.com/docs/IO/43395/tesla_technical_brief.pdf
(ogled spletne strani: junij 2010)

[84] http://freenetproject.org/whatis.html
(ogled spletne strani: februar 2011)