Aplikacija za praćenje programskog opterećenja klastera računala u stvarnom vremenu

Barbarić, Tomislav

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:789232

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

APLIKACIJA ZA PRAĆENJE PROGRAMSKOG OPTEREĆENJA KLASTERA RAČUNALA U STVARNOM VREMENU

Diplomski rad

Tomislav Barbarić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Tomislav Barbarić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1265R, 07.10.2022.
JMBAG:	0165081277
Mentor:	izv. prof. dr. sc. Zdravko Krpić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Goran Martinović
Član Povjerenstva 1:	izv. prof. dr. sc. Zdravko Krpić
Član Povjerenstva 2:	izv. prof. dr. sc. Ivica Lukić
Naslov diplomskog rada:	Aplikacija za praćenje programskog opterećenja klastera računala u stvarnom vremenu
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Cilj rada je istražiti postojeća rješenja za vizualizaciju opterećenja klastera računala u stvarnom vremenu, te izazove i specifičnosti u izvedbi istih. U praktičnom dijelu rada implementirati aplikaciju koja u stvarnom vremenu prikazuje opterećenje najvažnijih resursa klastera računala, kao što su zauzeće CPU-a i radne memorije te mrežne aktivnosti. Navedeno programsko rješenje moguće je implementirati kao desktop, web ili mobilnu aplikaciju korištenjem aktualnih tehnologija.

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	02.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O IZVORNOSTI RADA

Osijek, 02.10.2024.

Ime i prezime Pristupnika:	Tomislav Barbarić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1265R, 07.10.2022.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za praćenje programskog opterećenja klastera računala u stvarnom vremenu**

izrađen pod vodstvom mentora izv. prof. dr. sc. Zdravko Krpić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD 1

2. PREGLED POSTOJEĆIH PROGRAMSKIH RJEŠENJA 2

2.1. Ganglia 2

2.2. Prometheus 4

2.3. Zabbix 6

2.4. Netdata 8

2.5. Nagios 10

3. PRIKUPLJANJE MJERENIH VELIČINA S RAČUNALNIH ČVOROVA 12

3.1. Mjerene veličine procesora 12

3.2. Mjerene veličine memorije 14

3.3. Mjerene veličine vezane uz sustave za pohranu podataka 16

3.4. Mjerene veličine ulazno-izlaznih operacija diska 18

3.5. Mjerene veličine mrežnih uređaja 21

3.6. Ostale mjerene veličine 23

3.7. Razlika mjerenih veličina pojedinog čvora i kumulativnih mjerenih veličina računalnog klastera 25

4. PROGRAMSKO RJEŠENJE 28

4.1. Korištene tehnologije i biblioteke 29

4.2. Princip rada prikupljanja mjerenih veličina 31

4.3. Značajke aplikacije 33

5. VREDNOVANJE PROGRAMSKOG RJEŠENJA 39

5.1. HIGH PERFORMANCE LINPACK BENCHMARK (HPL) 39

5.2. Analiza rezultata mjerila 42

6. ZAKLJUČAK 45

LITERATURA 47

SAŽETAK 49

ABSTRACT **50**

ŽIVOTOPIS **51**

PRILOZI **52**

1. UVOD

U dinamičnom okruženju računalnih klastera, pravovremeno praćenje programskog opterećenja je presudno za održavanje stabilnosti i performansi sustava. Klaster računala predstavlja skup međusobno povezanih računala koja rade zajedno kao jedinstven sustav, pružajući veću računalnu snagu i pouzdanost u usporedbi s pojedinačnim računalima. Kako se računalni klasteri sve više koriste u raznim područjima, od znanstvenih istraživanja do poslovnih aplikacija, raste potreba za naprednim alatima koji mogu pružiti točne i pravovremene informacije o stanju sustava. Pravovremeno praćenje opterećenja omogućuje administratorima da identificiraju i riješe potencijalne probleme prije nego što eskaliraju u ozbiljnije kvarove. Točne i pravovremene informacije o stanju sustava ključne su za optimizaciju resursa, balansiranje opterećenja i održavanje visoke dostupnosti sustava. Bez odgovarajućih alata za praćenje, teško je prepoznati uzroke degradacije performansi, što može dovesti do neefikasnosti i smanjenja pouzdanosti sustava. Stoga je potrebno učinkovito rješenje koje će administratorima sustava omogućiti praćenje i analizu programskog opterećenja u stvarnom vremenu, značajno smanjujući vrijeme potrebno za otkrivanje i rješavanje problema te optimizirajući resurse i poboljšavajući ukupnu učinkovitost računalnih klastera. Programsko rješenje mora biti jednostavno za postavljanje i instalaciju, kako bi se olakšala implementacija u različite okoline računalnih klastera i osigurala njegova široka primjenjivost. Cilj rada je definirati i razviti aplikaciju za praćenje programskog opterećenja klastera računala u stvarnom vremenu, koja će korisnicima pružiti intuitivno sučelje za vizualizaciju podataka o opterećenju.

Drugo poglavlje detaljno prikazuje pregled postojećih programskih rješenja te načine rada i razlike istih. Treće poglavlje pokazuje koje kategorije mjerenih veličina za nadziranje postoje te koje je bitno prikupljati, a koje nije. Također je u trećem poglavlju objašnjena razlika prikupljanja mjerenih veličina za pojedini čvor te kumulativno za cijeli računalni klaster. Zatim je u četvrtom poglavlju objašnjen princip rada programskog rješenja te su pokazane osnovne značajke aplikacije. U posljednjem poglavlju ispitan je utjecaj razvijene aplikacije na čvorove računalnog klastera te je objašnjeno samo vrednovanje ispravnosti aplikacije.

2. PREGLED POSTOJEĆIH PROGRAMSKIH RJEŠENJA

Za praćenje programskog opterećenja računalnih klastera razvijene su različite aplikacije za nadzor i upravljanje. Takve aplikacije omogućuju administratorima uvid u stanje svakog čvora u računalnom klaster, pružajući podatke u stvarnom vremenu te analizu performansi kako bi se optimizirala iskorištenost resursa. Među najpoznatijim aplikacijama za praćenje klastera računala su Ganglia, Prometheus, Zabbix, Netdata te Nagios. Svaka od aplikacija pruža jedinstvene mogućnosti za nadzor specifičnih aspekta klastera računala, poput opterećenja procesora, zauzeća memorije, mrežnog prometa, I/O operacija i slično. Načini prikupljanja mjerenih veličina i sama vizualizacija podataka razlikuju se od aplikacije do aplikacije.

2.1. Ganglia

Prema [1], Ganglia je skalabilni raspodijeljeni sustav za nadziranje računalnih sustava računarstva visokih performansi (eng. *High Performance Computing*, HPC) poput računalnih klastera i *grid* mreža. Temelji se na hijerarhijskom dizajnu federacije računalnih klastera, što znači da postoje različite razine ili slojevi čvorova koji se nadziru, gdje svaka razina može skupljati podatke od više čvorova niže razine. Takva arhitektura omogućuje skalabilno i učinkovito praćenje velikih mreža međusobno povezanih računalnih klastera. Koristi proširivi jezik za označavanje (eng. *Extensible Markup Language*, XML) za strukturiranje podatka, vanjsko postavljanje podataka (eng. *eXternal Data Representation*, XDR) za učinkoviti prijenos podataka te kružnu bazu podataka (eng. *Round-Robin Database*, RRD) za pohranu i vizualizaciju. XDR je standard koji definira način kako se podaci mogu predstaviti i prenositi neovisno o arhitekturi računala. RRD je vrsta baze podataka optimizirana za pohranu vremenski serijaliziranih podataka, kao što su performanse mreže, podaci o temperaturi ili vremenu, statistike poslužitelja i slično.

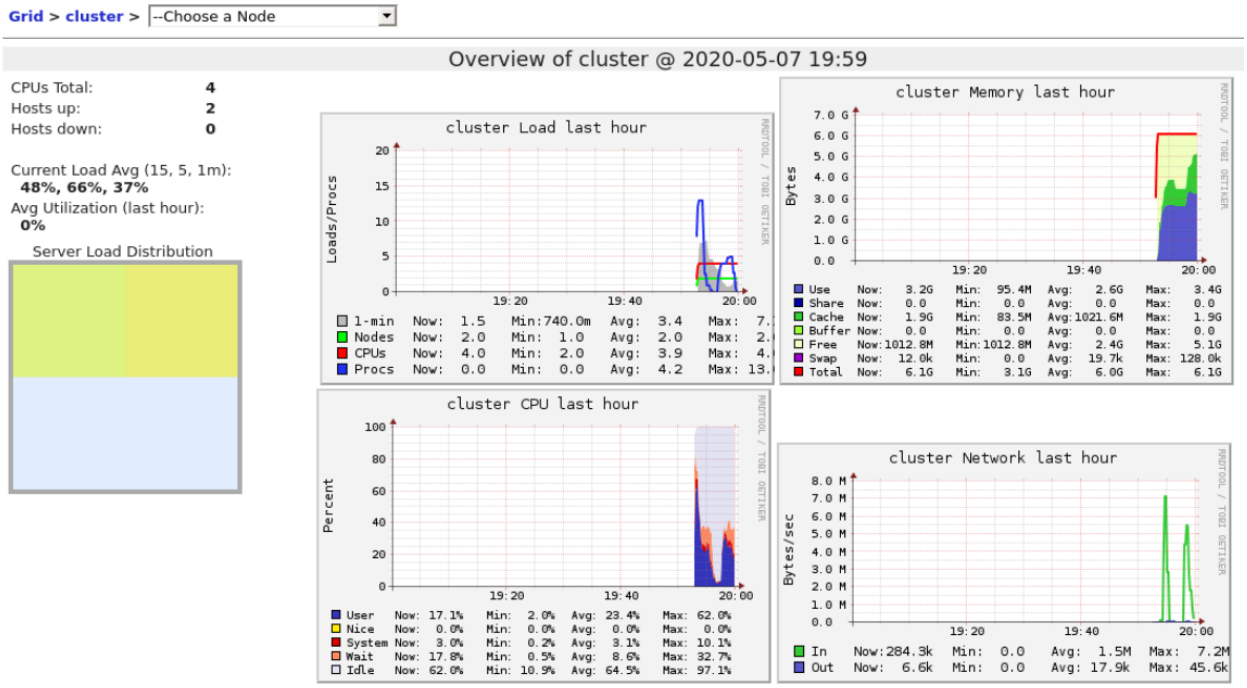
Posebno razvijene strukture podatka i algoritmi omogućuju vrlo nisko opterećenje za pojedini čvor te visoku konkurentnost. Ganglia sustav je robustan te podržava širok raspon operacijskih sustava i procesorskih arhitektura, a trenutno se koristi na tisućama računalnih klastera diljem svijeta. Za prikupljanje podataka oslanja se na daemone, što je vrsta pozadinskog procesa ili servisa koji radi u pozadini sustava bez direktnog sudjelovanja korisnika.

Ganglia se sastoji od tri glavna dijela, kao što je navedeno pod [1]:

1. Ganglia daemon za nadziranje (eng. *Ganglia monitoring daemon*, gmond) – daemon koji se instalira na svakom čvoru koji se želi nadzirati. Koristi listen/announce protokol putem XDR-a te prikupljene podatke šalje u XML formatu TCP protokolom. Prikuplja razne podatke vezane za procesor, memoriju, disk i mrežu.
2. Ganglia meta servis (eng. *Ganglia Meta Daemon*, gmetad) – usluga koja prikuplja podatke od ostalih gmetad i gmond izvora te njihovo stanje sprema na disk koristeći RRD. Za prikupljanje podatka grupe računalnih klastera koristi mehanizam upita te podržava hijerarhijsko delegiranje za uspostavu nadzora.
3. Ganglia web sučelje (eng. *Ganglia Web Interface*, gwen2) – korisničko sučelje koje prikazuje podatke koje je spremio gmetad. Koristi PHP kako bi se izradilo grafičko korisničko sučelje.

Ganglia aplikacija podržava dva načina rada, multicast i unicast. Prema [1], zadani način rada je multicast, pri čemu gmond instance s različitih čvorova ili računalnih klastera mogu međusobno komunicirati te će, u slučaju da nedostaju metapodaci, pokušati pronaći te informacije jedan od drugoga. Ovaj način rada jednostavan je za postaviti te pruža redundanciju. Određena okruženja, poput Amazonovog AWS EC2, ne podržavaju multicast način rada. Zato postoji unicast način rada koji je složeniji za postaviti te u njemu gmond može direktno komunicirati samo s instancom gmetad.

Aplikacija može prikupljati širok raspon sustavnih i programskih mjerenih veličina računalnih klastera i mreža za različite operacijske sustave. Pod [2] su navedene sve mjerene veličine koje aplikacija može skupljati. Neke od važnijih mjerenih veličina koje može prikupljati su detalji opterećenja procesora, detalji zauzeća memorije i diska, trenutne ulazno-izlazne operacije diska, mrežni promet i slično. Slika 2.1. prikazuje korisničko sučelje Ganglije na kojemu su vidljivi grafovi za opterećenje procesora, zauzeće memorije i mrežni promet jednog čvora. Također na slici se vidi da je moguće odabrati vizualizaciju mjerenih veličina za različite čvorove unutar računalnih klastera.



Slika 2.1. Korisničko sučelje Ganglije

Dok Ganglia zahtijeva postavljanje agenata (*gmond*) na svaki čvor u mreži, programsko rješenje izrađeno u okviru ovog rada nema potrebu za dodatnim agentima. To omogućava prikupljanje informacija s čvorova bez dodatnih instalacija, što značajno smanjuje složenost administracije. Na taj način, izrađeni sustav smanjuje potrebu za konfiguracijom i održavanjem na svakom čvoru, što je ključna prednost u odnosu na Gangliu. Korisničko sučelje i grafovi koje Ganglia izrađuje služili su kao referenca za dizajn grafova u izrađenom programskom rješenju. Iako se metodologije prikupljanja podataka razlikuju, pristup u izrađenom rješenju odražava slične principe u vizualizaciji podataka, pružajući jasno i pregledno predstavljanje performansi sustava.

2.2. Prometheus

Prometheus je rješenje za nadziranje sustava temeljeno na bazi podataka vremenskih serija (eng. *Time Series Database*, TSDB), odnosno spremanju čistih numeričkih podataka u vremenskom nizu, kao što je navedeno pod [3]. Prikuplja, organizira i sprema mjerene veličine zajedno s vremenskim oznakama i identifikatorima. Mjerene veličine se prikupljaju s pomoću HTTP endpointa na željenim čvorovima. Pored prikupljanja mjerenih veličina raznih sustava Prometheus ima mogućnost i slanja obavijesti korisnicima. S obzirom na predefinirana pravila, poput prekoračenja iskorištenosti memorije ili postotka opterećenja procesora, korisniku se na različite načine može poslati obavijest. Mjerene veličine koje Prometheus skuplja su identificirane

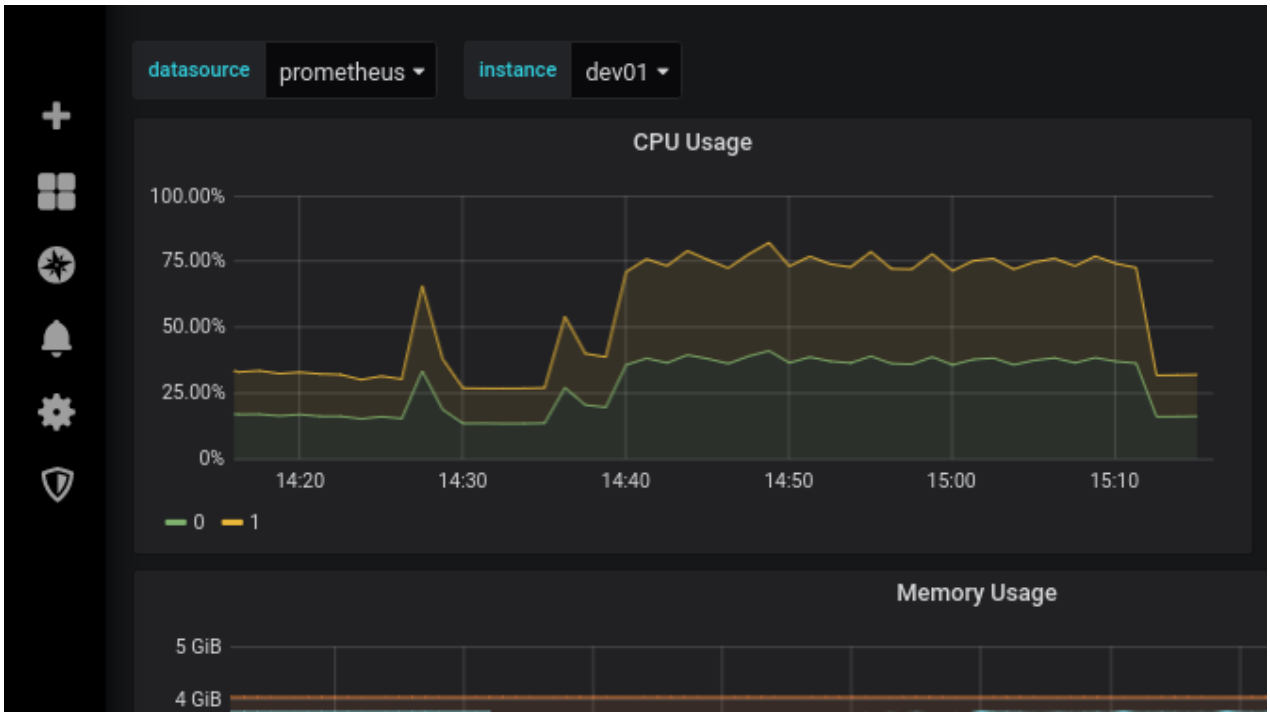
jedinstvenom kombinacijom ključa i vrijednosti što omogućava fleksibilnost pri upitima. Za upite i procesiranje skupljenih mjerenih veličina Prometheus koristi PromQL (eng. *Prometheus Query Language*, PromQL), vlastiti jezik za upite koji korisnicima dopušta biranje i agregaciju podataka iz TSDB u stvarnom vremenu.

Prema [3], komponente od kojih se Prometheus sastoji su:

1. Prometheus poslužitelj – glavna svrha poslužitelja je prikupljanje mjerenih veličina sa željnih čvorova u stalnim intervalima te spremanje istih.
2. Exporter – komponenta koja pomaže prikupiti postojeće mjerne veličine sa sustava, pretvara ih u format kompatibilan s Prometheusom te ih otkriva poslužitelju preko HTTP endpointa.
3. AlertManager – obrađuje obavijesti koje Prometheus poslužitelj pošalje i prema njima šalje obavijesti korisniku putem maila, Slacka i sličnih medija.
4. Push Gateway – omogućuje kratkotrajnim poslovima da šalju mjerne veličine Prometheusu.

Prometheus podržava četiri vrste mjerenih veličina, kao što je navedeno pod [3], a to su *counter*, *gauge*, *histogram* i *summary*. Za potrebe nadziranja računalnih klastera korisne su *gauge* mjerene veličine, odnosno numeričke vrijednosti koje mogu rasti ili se smanjivati. U tu vrstu spadaju detalji o opterećenju procesora, zauzeće memorije i diska, mrežni promet i slično. Mjerene veličine vrste *counter* predstavljaju samo neprekidno rastuće vrijednosti, kao što su broj zahtjeva ili grešaka. Mogu se samo povećavati, a u slučaju ponovnog pokretanja aplikacije mogu se resetirati na nulu. Mjerene veličine vrste *histogram* mjere distribuciju vrijednosti, omogućujući analizu raspodjele i agregaciju mjerenja kao što su trajanje zahtjeva ili veličina paketa. Mjerene veličine vrste *summary* pružaju kvantilne i agregirane statistike (npr. prosjeke), što omogućuje detaljniju analizu raspodjele podataka, poput trajanja zahtjeva ili vremena odgovora.

Za vizualizaciju prikupljenih podataka Prometheus ima ugrađeni funkciju zvanu *expression browser* koja je korisna za specifične upite i debugging te rezultate prikazuje unutar tablice ili vremenskog grafa. Za detaljnije grafove i bolje korisničko iskustvo Prometheus se najčešće koristi u kombinaciji s Grafanom. Grafana je platforma za analitiku i vizualizaciju podataka iz različitih izvora podataka. Slika 2.2. prikazuje dio korisničkog sučelja Grafane. Na slici je vidljiv graf opterećenosti procesora. Također vidljiva je mogućnost odabira čvora za kojega se vizualiziraju podaci.



Slika 2.2. Dio korisničko sučelja Grafane

Prometheus zahtjeva instalaciju i konfiguraciju svojih agenata na svakom čvoru, čime se razlikuje od izrađenog rješenja. Eliminiranjem potrebe za dodatnim agentima na čvorovima, programsko rješenje izrađeno u okviru ovog rada smanjuje administrativne zadatke na pojedinom čvoru.

2.3. Zabbix

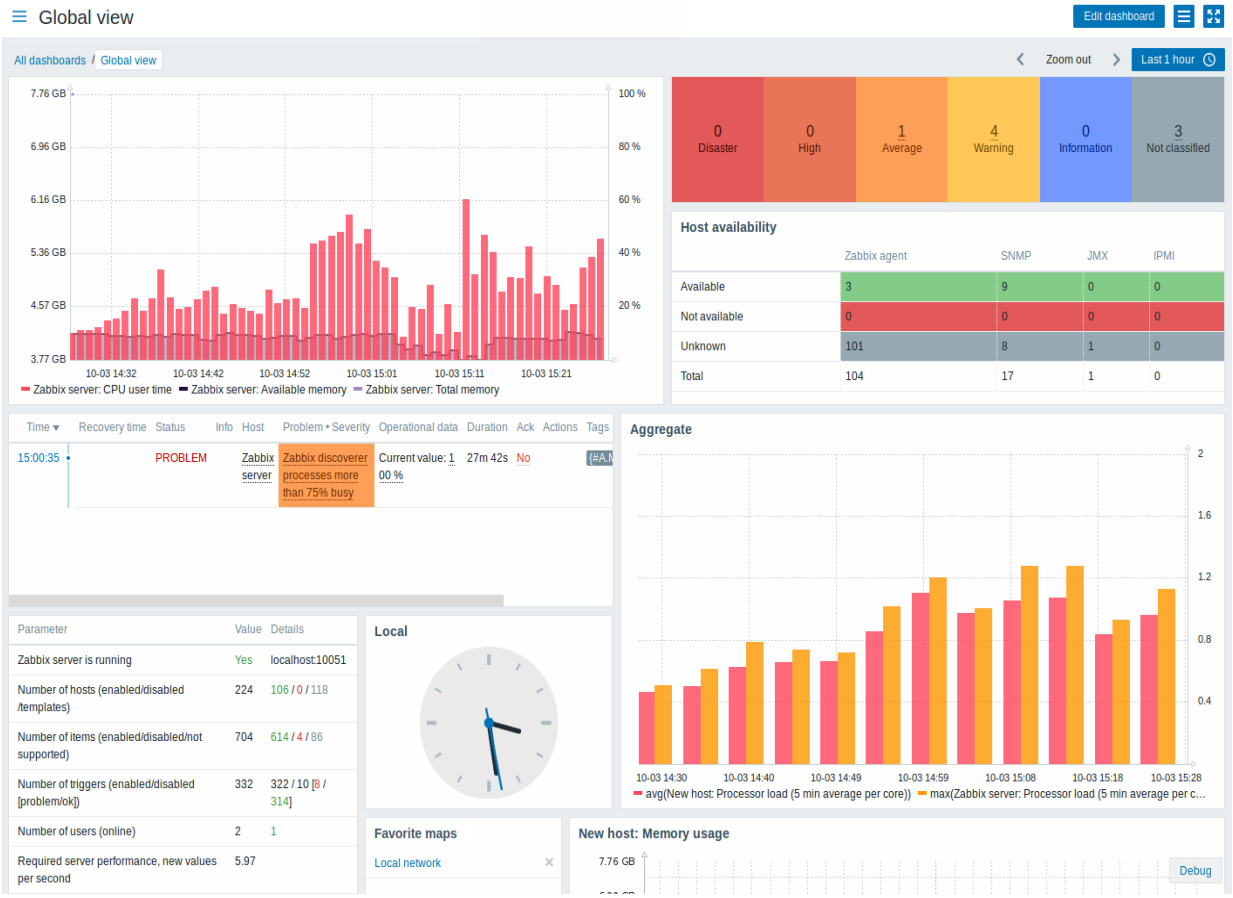
Zabbix je open-source alat za nadzor i upravljanje računalne infrastrukture u stvarnom vremenu prema [4]. Koristi se za praćenje i prikupljanje podataka o performansama mrežnih uređaja, poslužitelja, virtualnih strojeva i aplikacija. Podatke prikuplja koristeći razne metode kao što su SNMP (eng. *Simple Network Management Protocol*, SNMP), IPMI (eng. *Intelligent Platform Management Interface*, IPMI), JMX (eng. *Java Management Extensions*, JMX), kao i putem SSH (eng. *Secure Shell*, SSH) i Telnet protokola. Podaci se prikupljaju kroz aktivne i pasivne provjere, kao što je navedeno pod [4]. Pasivne provjere podrazumijevaju da Zabbix poslužitelj traži podatke od agenata, dok aktivne provjere znače da agenti samostalno prikupljaju i šalju podatke poslužitelju. Ovi podaci uključuju statistike o korištenju procesora, memorije, prostora na disku, mrežnom prometu i dostupnosti usluga. Zabbix također omogućava konfiguriranje pravila za otkrivanje problema i generiranje alarma te grafički prikaz performansi sustava [4].

Za ispravan rad cijelog sustava potrebno je nekoliko komponenti, kao što je navedeno pod [4]:

1. Zabbix poslužitelj – obavlja prikupljanje podataka te pohranjuje prikupljene podatke u bazu podatka. Također upravlja konfiguracijom agenata, provodi pasivne i aktivne provjere, te generira alarme u slučaju otkrivanja problema.
2. Zabbix agent – komponenta koja se instalira na sve uređaje kojima se žele pratiti mjerene veličine. Prikupljene podatke šalju poslužitelju na obradu i pohranu. Agenti podržavaju aktivne i pasivne provjere te su dostupni za razne operacijske sustave.
3. Baza podataka – služi za pohranjivanje svih podataka koje Zabbix prikuplja i generira.
4. Web sučelje – grafički dio Zabbix aplikacije koji omogućava korisnicima interakciju s podacima prikupljenim od strane Zabbix poslužitelja putem preglednika.

Pored potrebnih komponenti postoji i opcionalna komponenta Zabbix proxy. Služi za posredovanje između Zabbix poslužitelja i nadziranih uređaja. Glavna uloga mu je preuzimanje tereta prikupljanja podatak i distribucija nadzornog opterećenja, pogotovo u okruženjima s jednim Zabbix poslužiteljem.

Zabbix prikuplja širok spektar mjerenih veličina koje pokrivaju različite aspekte infrastrukture. Prema [4], svaka mjerena veličina ima svoju ključnu vrijednost kojom se raspoznaje što je prikupljeno. Mjerene veličine su organizirane u kategorije, uključujući *Kernel*, *Network*, *Processes*, *System*, *Web Monitoring* i *Zabbix*. Kategorija *Kernel* obuhvaća podatke o jezgru operativnog sustava, *Network* prati mrežnu aktivnost, *Processes* se odnosi na informacije o pokrenutim procesima, *System* pruža podatke o općem stanju sustava, *Web Monitoring* nadzire performanse web usluga, dok *Zabbix* pokriva mjerene veličine vezane uz samu platformu Zabbix. Prikupljeni podaci se vizualiziraju se na različite načine. Slika 2.3. prikazuje nadzornu ploču Zabbix poslužitelja na kojoj je vidljiva iskorištenost memorije, obavijesti o problemima, prosjek skupnog opterećenja procesora uređaja te ostale korisne informacije poput stanja poslužitelja i broja nadgledanih uređaja, lokalnog vremena te alarma.



Slika 2.3. Nadzorna ploča Zabbix poslužitelja

Zabbix prati različite aspekte sustava koristeći agente koji se instaliraju na svakom čvoru. Suprotno tome, programsko rješenje izrađeno u okviru ovog rada nema potrebu za dodatnim agentima kako bi prikupljao informacije o sustavu, čime se pojednostavljuje upravljanje i smanjuju administrativni zahtjevi u usporedbi sa Zabbixom.

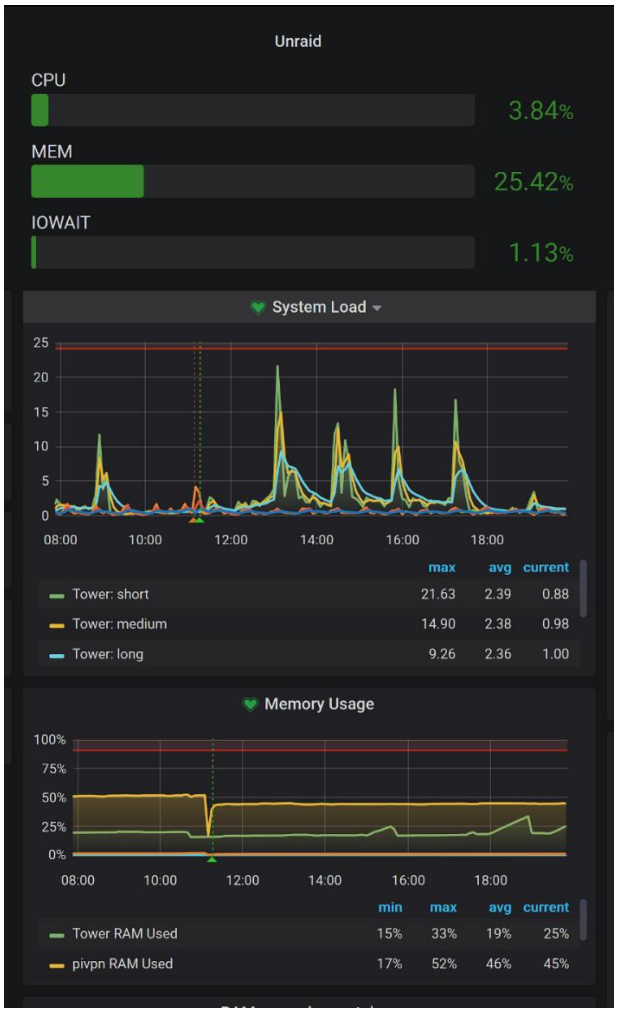
2.4. Netdata

Netdata je open-source alat za praćenje i vizualizaciju mjerenih veličina računalnih sustava u stvarnom vremenu. Prema [5], prikupljanje mjerenih veličina se odvija svake sekunde, a rezultati se prikazuju u nadzornoj ploči. Dizajniran je da prikuplja podatke s fizičkih i virtualnih poslužitelja, cloud instanci, Kubernetes klastera i sličnih infrastrukturnih rješenja. Kao i prijašnji alati ima mogućnost obavijesti i alarma u slučaju problema s nadziranim uređajem. Podržava razne metode slanja obavijesti poput maila, Slacka ili SMS-a.

Prednost Netdata alata nad prije spomenutim alatima je jednostavna instalacija. Podržava Linux, macOS i FreeBSD operacijske sustave. Kao što je navedeno pod [6] instalacija se odvija

pokretanjem skripte koja instalira sve potrebno za pokretanje alata. Nakon instaliranja agenta na željenom uređaju otvara se port 19999 preko kojega se može pristupiti nadzornoj ploči. Svaki uređaj na koji se instalira Netdata ima svoju nadzornu ploču što za veliki broj uređaja nije pregledno. Radi bolje organizacije i grupiranja uređaja postoji *Netdata Cloud*. Prema [6], *Netdata Cloud* dopušta uvid u prikupljene podatke od bilo kuda, skalabilnost, pristup temeljen na ulogama te slanje obavijesti i alarma.

Netdata može prikupljati širok spektar mjerenih veličina. Glavne grupe koje prikuplja su resursi sustava, mreža, pohrana, procesi, hardver i senzori, kao što je navedeno pod [6]. Ako je instaliran na Linux operativnom sustavu također može pratiti i dostupne mjerene veličine jezgre. Slika 2.4. prikazuje dio nadzorne ploče Netdata alata na kojoj su vidljivi grafovi zauzeća memorije i opterećenje sustava. Također su vidljivi postotci opterećenja procesora, zauzeća memorije i ulazno-izlaznih operacija.



Slika 2.4. Dio nadzorne ploče Netdata alata

Netdata pruža detaljan uvid u performanse sustava koristeći agente koji se instaliraju na svaki čvor. programsko rješenje izrađeno u okviru ovog rada koristi SSH za prikupljanje informacija, eliminirajući potrebu za dodatnim agentima. Ovaj pristup smanjuje potrebu za konfiguracijom na svakom čvoru i pojednostavljuje održavanje sustava. Korištenjem SSH za prikupljanje mjerenih veličina, programsko rješenje izrađeno u okviru ovog rada omogućuje bržu i lakšu implementaciju u usporedbi s Netdata, što može značajno smanjiti složenost upravljanja infrastrukturom.

2.5. Nagios

Nagios je open-souce alat za nadzor sustava i mreža. Napravljen je za Linux operacijske sustave. Prema [7], omogućava periodično praćenje dostupnosti mrežnih usluga poput SMTP, POP3, SSH, HTTP i drugih mrežnih protokola. Pored tog prati i iskorištenost resursa poput zauzeća memorije i diska, opterećenja procesora, broja pokrenutih procesa i slično. Također može slati obavijesti u slučaju pada sustava ili drugih definiranih pravila okidanja.

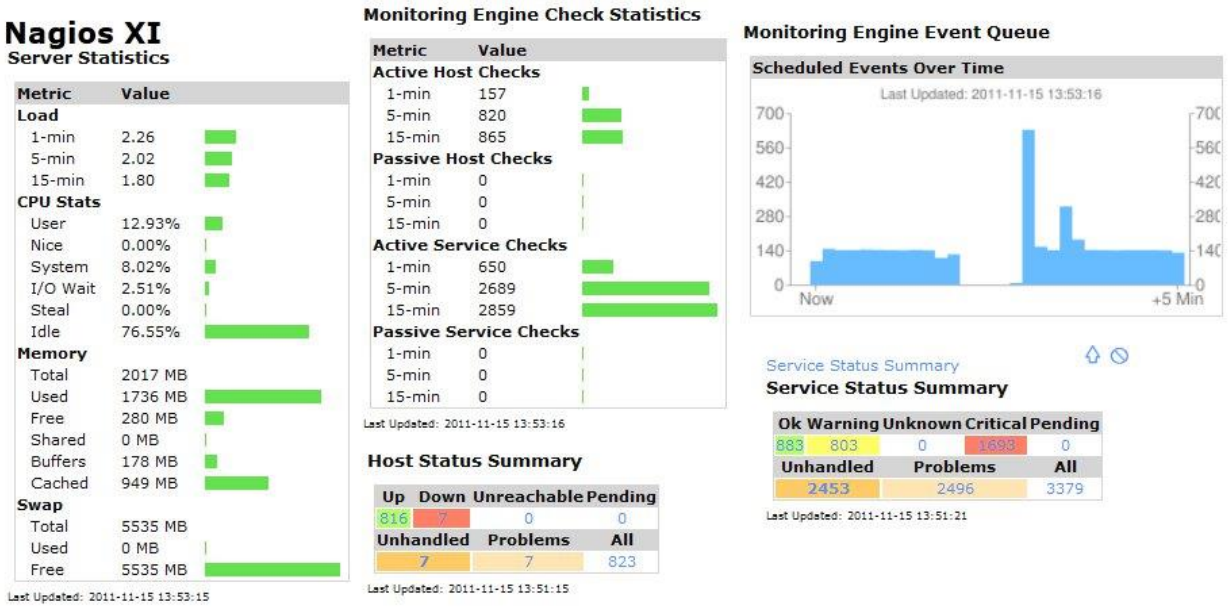
Za postavljanje Nagios sustava potrebno je instalirati različite komponente na glavni poslužitelj i sam nadgledani uređaj. Kao što je navedeno pod [7] na glavnom poslužitelju je potrebno imati:

1. Nagios Core – osnovna komponenta koja omogućava praćenje i obavještavanje.
2. Nagios Plugins – dodatci koji obavljaju provjere stanja servisa i prikupljanje mjerenih veličina.
3. Web poslužitelj – Apache ili neki drugi web poslužitelj potreban za pokretanje web sučelja Nagiosa.
4. NRPE (eng. *Nagios Remote Plugin Executor*, NPRE) – omogućuje daljinsko izvršavanje provjera na nadgledanim uređajima.

Na svakom nadgledanom uređaju instaliraju se:

1. Nagios NPRE – omogućuje izvršavanje provjera.
2. Nagios Plugins – dodatci koji se koriste za provjere.

Za prikupljanje mjerenih veličina i provjeru statusa mrežnih servise koriste se dodatci koji se izvršavaju s pomoću NRPE-a. Prema [7], dostupni dodatci za mjerene veličine sustava prikupljaju opterećenje procesora, zauzeće memorije i diska i slično. *Nagios Core* ima ugrađeno korisničko sučelje za vizualizaciju podataka kojemu se pristupa putem web poslužitelja. Slika 2.5. prikazuje korisničko sučelje Nagiosa za nadgledani uređaj.



Slika 2.5. Korisničko sučelje Nagiosa

Nagios koristi agente za nadzor i provjeru statusa, koji se moraju instalirati i konfigurirati na svakom čvoru. U usporedbi s ovim pristupom, programsko rješenje izrađeno u okviru ovog rada koristi SSH za prikupljanje podataka bez potrebe za instalacijom agenata. Ova metoda omogućava centralizirano prikupljanje informacija i smanjuje složenost instalacije i održavanja. Korištenje SSH za pristup podacima nudi jednostavnije upravljanje sustavom i može značajno smanjiti operativne troškove u odnosu na Nagios.

3. PRIKUPLJANJE MJERENIH VELIČINA S RAČUNALNIH ČVOROVA

Prikupljanje mjerenih veličina je ključni dio upravljanja i optimizacije računalnih sustava, posebno u složenim okruženjima kao što su računalni klasteri. Mjerene veličine pružaju vrijedne informacije koje pomažu administratorima bolje razumijevanje ponašanja sustava, identifikaciju problema, optimizaciju performansi te planiranje budućih potreba. Mjerene veličine se mogu podijeliti u različite kategorije ovisno za koju komponentu se prikupljaju. Osnovne kategorije uključuju mjerene veličine procesora, mjerene veličine memorije, mjerene veličine vezane uz sustave za pohranu podataka, mjerene veličine ulazno-izlaznih operacija diska te mjerene veličine mrežnih uređaja.

3.1. Mjerene veličine procesora

Prikupljanje procesorskih mjerenih veličina ključno je za detaljno razumijevanje performansi računalnih klastera i identifikaciju potencijalnih problema koji mogu utjecati na učinkovitost sustava. Ove veličine pružaju precizan uvid u stupanj iskorištenosti procesorskih resursa, omogućujući analizu ukupnog opterećenja sustava, kao i evaluaciju efikasnosti pojedinih operacija. Takvi podaci su nužni za optimizaciju i održavanje stabilnosti računalnih klastera, jer pomažu u prepoznavanju neadekvatne distribucije resursa i osiguravaju da sustav funkcionira unutar optimalnih granica. Kroz praćenje ovih veličina, moguće je detektirati neravnomjernu raspodjelu opterećenja među čvorovima, što često ukazuje na potrebu za skaliranjem broja čvorova, prilagodbom zadataka ili optimizacijom programskog koda.

Osnovne procesorske mjerene veličine su:

1. **Opterećenost procesora** (eng. *CPU Utilization*) – ova mjerena veličina izražava postotak vremena u kojem je procesor aktivno korišten za izvršavanje zadataka. Postotak opterećenja procesora reflektira koliko su procesorski resursi zauzeti unutar određenog vremenskog okvira. Na primjer, ako procesor radi na 80 % iskorištenosti, to znači da je procesor bio aktivan 80 % vremena, dok je preostalih 20 % vremena bio neaktivan ili u stanju mirovanja. Sustavi koji konstantno pokazuju visok postotak opterećenja mogu biti preopterećeni, što dovodi do smanjenja performansi i povećanja vremena odziva aplikacija, te potencijalno uzrokuje degradaciju performansi cijelog računalnog klastera.
2. **Korisničko vrijeme procesora** (eng. *User Time*) – ova mjerena veličina bilježi vrijeme koje procesor troši na izvršavanje korisničkih procesa, tj. aplikacija i programa

pokrenutih od strane korisnika. Izražava se kao postotak ukupnog procesorskog vremena. Na primjer, korisničko vrijeme od 50 % ukazuje na to da se polovica ukupnog vremena procesora koristi za izvršavanje korisničkih zadataka, poput aplikacija, znanstvenih simulacija ili poslovnih procesa. Visoko korisničko vrijeme može ukazivati na intenzivne zadatke koje obavlja aplikacija, te pomoći u analizi koja od aplikacija zahtijeva najviše resursa.

3. **Sustavno vrijeme procesora** (eng. *System Time*) - ova mjerena veličina prikazuje vrijeme koje procesor koristi za izvršavanje sustavnih operacija koje upravljaju resursima računala, poput rada jezgre operacijskog sustava, upravljanja memorijom ili hardverskim resursima. Također se izražava kao postotak ukupnog vremena procesora. Visoko sustavno vrijeme može signalizirati prekomjerno trošenje procesorskih resursa na aktivnosti poput upravljanja procesima, rukovanja mrežnim zahtjevima ili izvršavanja operacija održavanja sustava, umjesto na korisničke procese. Ovo može ukazivati na neefikasnost upravljanja hardverom, što može smanjiti ukupnu učinkovitost sustava.
4. **Prosječno opterećenje sustava** (eng. *Load Average*) – predstavlja broj procesa koji su spremni za izvršavanje u određenom vremenskom periodu. Najčešće se prikazuje kao tri vrijednosti koje predstavljaju prosječno opterećenje u proteklih 1, 5 i 15 minuta. Na primjer, prosječno opterećenje od 4 na četverojezgrenom sustavu signalizira ujednačeno korištenje procesora, dok prosječno opterećenje iznad broja jezgri (npr. 6 na sustavu s četiri jezgre) može ukazivati na preopterećenje i potrebu za optimizacijom resursa ili dodavanjem novih čvorova.

U izrađenom programskom rješenju prikupljaju se podaci o opterećenosti procesora, korisničko i sustavno vrijeme procesora te prosječno opterećenje sustava. Praćenje procesorskih mjerenih veličina je od ključne važnosti za proaktivno upravljanje performansama sustava. Ove mjerene veličine omogućuju pravovremeno identificiranje problema, poput prekomjernog opterećenja ili neadekvatne raspodjele zadataka, što može rezultirati padom performansi aplikacija i sustava u cjelini. Na primjer, ako je opterećenje procesora neujednačeno i neki čvorovi rade blizu maksimalnog kapaciteta (90 % ili više), dok su drugi relativno neiskorišteni, to može značiti da su zadaci loše raspoređeni ili da postoji potreba za optimizacijom algoritama za raspodjelu poslova. Redistribucija radnih zadataka među čvorovima ili dodavanje novih čvorova može pomoći u prevenciji uskih grla i osigurati ujednačeno opterećenje cijelog računalnog klastera.

Većina modernih operacijskih sustava pruža ugrađene alate za praćenje procesorskih mjerenih veličina. Na primjer, u Unix i Linux sustavima, alati kao što su *top*, *htop* i *vmstat* nude uvid u opterećenost procesora i korištenje resursa. Windows operacijski sustav nudi slične mogućnosti kroz Upravitelj zadataka (eng. *Task Manager*). Ovi alati omogućuju korisnicima i administratorima praćenje performansi sustava.

Pravilno prikupljanje i analiza ovih mjerenih veličina ključni su za optimizaciju performansi i dugoročnu stabilnost računalnih klastera. Ispravno tumačenje podataka može pomoći u identificiranju potencijalnih problema, omogućiti pravovremene intervencije, poput skaliranja resursa sustava (npr. dodavanje novih čvorova) ili optimizacije postojećih resursa i aplikacija i osigurati optimalan rad složenih sustava.

3.2. Mjerene veličine memorije

Memorija je jedan od najvažnijih resursa u računalnim sustavima, posebno u okruženjima s velikim opterećenjima kao što su računalni klasteri. Brzi pristup podacima i učinkovita distribucija memorijskih resursa omogućuju optimizirano izvođenje aplikacija, dok neadekvatno upravljanje memorijom može uzrokovati značajne padove performansi ili čak potpuni zastoj sustava. Praćenje ključnih mjerenih veličina memorije omogućuje detaljan uvid u iskorištenost memorijskih kapaciteta te pomaže u identifikaciji potencijalnih problema poput nedostatka memorije, neučinkovite alokacije resursa ili curenja memorije. U kontekstu upravljanja računalnim klasterima, bitne su mjerene veličine koje omogućuju kontinuirano praćenje ukupne, iskorištene i slobodne memorije.

Osnovne memorijske mjerene veličine su:

1. **Ukupna dostupna memorija** (eng. *Total Memory*) – ova mjerena veličina prikazuje ukupnu količinu fizičke RAM memorije ugrađene na pojedinom čvoru unutar računalnog klastera. Praćenje ukupne memorije važno je za razumijevanje maksimalnog kapaciteta svakog čvora, što je presudno za planiranje i prilagodbu opterećenja aplikacija koje koriste velike količine memorijskih resursa. Povećanje opterećenja sustava ili pokretanje novih zadataka koji premašuju dostupnu memoriju mogu zahtijevati nadogradnju memorije ili redistribuciju zadataka na druge čvorove s većim memorijskim kapacitetom. Na primjer, u računalnim klasterima koji obrađuju velike skupove podataka aplikacije mogu zahtijevati nekoliko terabajta memorije za izvođenje proračuna u razumnom vremenu. Ako se pokaže da ukupna dostupna

memorija na postojećim čvorovima nije dovoljna, može se donijeti odluka o nadogradnji fizičke memorije ili dodavanju novih čvorova kako bi se podržale aplikacije s visokim memorijskim zahtjevima.

2. **Iskorištena memorija** (eng. *Used Memory*) – ova mjerena veličina prikazuje količinu memorije koja je trenutno u upotrebi od strane aplikacija, operacijskog sustava i drugih procesa. Visoka razina iskorištenosti memorije može ukazivati na potencijalna uska grla u performansama računalnih klastera. Na primjer, ako je aplikacija koja obavlja simulacije konstantno na visokom stupnju iskorištenosti memorije, to može signalizirati da aplikacija zahtijeva optimizaciju algoritama ili bolje upravljanje resursima kako bi se spriječilo zagušenje memorije. Također, u scenarijima gdje više korisnika paralelno pokreće zadatke unutar računalnog klastera, praćenje iskorištene memorije pomaže u alokaciji zadataka na manje opterećene čvorove, kako bi se izbjegla prekomjerna potrošnja memorijskih resursa na pojedinim čvorovima.
3. **Slobodna memorija** (eng. *Free Memory*) – ova mjerena veličina pokazuje količinu memorije koja je trenutno dostupna za nove zadatke, procese ili aplikacije. Ako slobodna memorija padne na vrlo nisku razinu, to može značiti da je sustav iscrpio većinu svojih memorijskih resursa i da je prisiljen koristiti *swap* memoriju, što znatno usporava performanse sustava zbog sporijeg pristupa podacima. Na primjer, u sustavima koji obrađuju podatke u stvarnom vremenu, kao što su sustavi za praćenje financijskih transakcija ili analiza podataka sa senzora, nedostatak slobodne memorije može dovesti do kašnjenja u obradi, što može imati ozbiljne posljedice, uključujući gubitak podataka ili prekid u operacijama.

U izrađenom programskom rješenju prikupljaju se podaci o ukupnoj dostupnoj memoriji te iskorištenoj i slobodnoj memoriji. Kontinuirano praćenje ovih veličina omogućuje pravovremeno prepoznavanje situacija u kojima memorija postaje usko grlo u radu računalnog klastera. Ako se utvrdi da su čvorovi blizu maksimalne iskorištenosti memorije, to može signalizirati potrebu za proširenjem memorijskog kapaciteta. Na primjer, u situacijama kada RAM memorija postaje kritično opterećena, administrator može odlučiti proširiti fizičku memoriju na čvorovima ili dodati nove čvorove s većim kapacitetom kako bi se izbjegla uska grla i omogućilo skaliranje aplikacija.

Pored fizičke RAM memorije, operacijski sustavi često koriste *swap* memoriju kao dio virtualne memorije. *Swap* memorija omogućuje premještanje manje korištenih podataka iz RAM-a na disk kada fizička memorija postane previše zauzeta. Iako to omogućuje nastavak rada sustava, prekomjerna upotreba *swap* memorije može drastično smanjiti performanse zbog puno sporijeg

pristupa podacima na disku u odnosu na RAM. Na primjer, ako aplikacija koristi *swap* memoriju umjesto RAM-a, vrijeme izvršenja zadataka može se značajno povećati, što može ugroziti cjelokupnu učinkovitost računalnog klastera. Praćenje iskorištenosti *swap* memorije omogućuje administratorima pravovremeno identificiranje situacija u kojima sustav počinje koristiti disk kao proširenje memorije. Prekomjerno oslanjanje na *swap* memoriju često signalizira potrebu za dodavanjem fizičke memorije na čvorovima ili optimizacijom aplikacija kako bi se smanjila memorijska potrošnja. Izrađeno programsko rješenje prikuplja podatke o iskorištenosti *swap* memorije.

Moderni operacijski sustavi pružaju razne alate za praćenje mjerenih veličina memorije. Na Linux i Unix sustavima alati poput *top, htop, vmstat* omogućuju praćenje osnovnih mjerenih veličina memorije u stvarnom vremenu. Za prikazivanje informacija o ukupnoj i slobodnoj memoriji koriste se *top* i *htop,* dok *vmstat* pruža detaljne informacije o virtualnoj memoriji i *swap* prostoru. Windows operacijski sustav nudi pregled osnovnih mjerenih veličina memorije unutar Upravitelja zadataka, uključujući ukupnu, iskorištenu i slobodnu memoriju. Upravitelj performansi (eng. *Performance Monitor*) omogućuje detaljnije praćenje memorije i može se konfigurirati za praćenje specifičnih mjerenih veličina i njihovih povijesnih trendova.

Praćenje memorijskih resursa ključno je za održavanje optimalnih performansi računalnih klastera. Precizna analiza iskorištenosti memorije, zajedno s praćenjem *swap* memorije, omogućuje pravovremeno prepoznavanje potencijalnih problema i optimizaciju resursa, čime se osigurava stabilnost i učinkovitost sustava. Ove mjerne veličine pružaju administratorima ključne informacije za donošenje informiranih odluka o budućim nadogradnjama i raspodjeli resursa.

3.3. Mjerene veličine vezane uz sustave za pohranu podataka

Praćenje mjerenih veličina vezanih uz sustave za pohranu podataka ključno je za održavanje optimalnih performansi, pouzdanosti i kapaciteta pohrane u računalnim klasterima. Diskovi su osnovna komponenta za pohranu podataka, te njihova učinkovitost izravno utječe na performanse sustava. U računalnim klasterima, gdje se obrađuju ogromne količine podataka, poput simulacija, analiza velikih podataka (eng. *Big Data*) ili strojnih učenja, pravilno praćenje kapaciteta i performansi diskova omogućuje pravovremeno prepoznavanje problema i donošenje odluka o proširenju ili optimizaciji resursa. Ako diskovi nisu optimalno upravljani, to može dovesti do značajnih usporavanja rada sustava, gubitka podataka ili čak prekida u operacijama. Bitne mjerene veličine za praćenje performansi diska uključuju trenutno zauzeće diska, slobodni prostor diska i

ukupni prostor diska. Ove veličine pomažu u osiguravanju optimalnog korištenja diskovnog prostora i omogućuju planiranje kapaciteta prema budućim potrebama.

Osnovne mjerene veličine vezane uz sustave za pohranu podataka:

1. **Trenutno zauzeće diska** (eng. *Disk Usage*) – mjeri koliko je prostora na disku trenutno zauzet podacima. Visok postotak zauzeća diska može signalizirati potrebu za proširenjem kapaciteta kako bi se izbjegli problemi s nedostatkom prostora za pohranu novih podataka. Na primjer, u računalnim klasterima koji obrađuju slike visoke rezolucije ili velike znanstvene podatke, poput satelitskih snimki ili genetskih sekvenci, diskovi se brzo pune. Ako zauzeće diska postigne visok postotak, to može dovesti do usporavanja aplikacija koje često čitaju i zapisuju podatke. Praćenje zauzeća diska omogućuje donošenje odluka o proširenju kapaciteta, kao što su nadogradnja diskova na veće ili premještanje podataka na vanjske sustave za pohranu.
2. **Slobodni prostor diska** (eng. *Free Disk Space*) – pokazuje koliko prostora na disku je trenutno dostupno za pohranu novih podataka. Niska razina slobodnog prostora može usporiti rad aplikacija koje zahtijevaju učestalo zapisivanje i čitanje podataka, poput baza podataka ili aplikacija koje obrađuju velike datoteke. Na primjer, u računalnom klasteru koji koristi distribuirane sustave za pohranu podataka, poput HDFS-a (eng. *Hadoop Distributed File System*, HDFS), nedostatak slobodnog prostora može uzrokovati neuspješno repliciranje podataka između čvorova, čime se ugrožava integritet podataka. Također, sustavi prisiljeni na automatsko brisanje ili premještanje podataka kako bi oslobodili prostor mogu postati neefikasni, što dovodi do značajnih problema s performansama.
3. **Ukupni prostor diska** (eng. *Total Disk Space*) – označava ukupni kapacitet diska, uključujući zauzeti i slobodni prostor. Praćenje ukupnog prostora pomaže u planiranju kapaciteta diska, što je od posebne važnosti u sustavima koji se kontinuirano šire i gdje se volumen podataka stalno povećava. Na primjer, računalni klasteri koji rade na znanstvenim projektima, poput istraživanja klimatskih promjena, svakodnevno akumuliraju terabajte podataka. Stalnim praćenjem ukupnog kapaciteta diska moguće je planirati buduće nadogradnje ili migracije podataka na vanjske sustave za pohranu kako bi se izbjegla zagušenja i preopterećenja diskovnih jedinica.

U izrađenom programskom rješenju prikupljaju se podaci o ukupnom prostoru diska te slobodnom i zauzetom prostoru diska. Praćenje zauzeća diska omogućuje detaljan uvid u brzinu

kojom se diskovi pune, što je ključna informacija za donošenje odluka o proširenju kapaciteta ili redistribuciji podataka. Na primjer, ako se pokaže da je zauzeće diska na nekoliko čvorova neprestano iznad 90 %, potrebno je hitno poduzeti korake kako bi se osigurao dodatan prostor, bilo proširenjem postojećih diskova ili dodavanjem novih particija.

Ako se tijekom praćenja otkrije da su neki čvorovi računalnih klastera vrlo blizu punjenja svojih diskova, mogu se primijeniti različite mjere. Jedna opcija je premještanje podataka na čvorove s većim slobodnim kapacitetom, ili pak dodavanje novih diskova, čime se sprječava zastoj u radu aplikacija. Također, ako se uoči pad performansi zbog niske brzine čitanja i pisanja, može se analizirati potreba za prelazak s tradicionalnih tvrdih diskova na brže SSD diskove, koji omogućuju brži pristup podacima i optimiziraju performanse aplikacija.

Linux i Unix sustavi pružanju alate poput *df* i *du*, koji omogućuju pregled zauzeća diska i slobodnog prostora. Informacije o ukupnom, slobodnom i zauzetom prostoru na diskovima prikazuju se naredbom *df*, dok *du* pruža detaljan uvid u zauzeće prostora po direktorijima. Na Windows operacijskom sustavu postoje dva alata za praćenje mjerenih veličina diska. Istraživač datoteka (eng. *File Explorer*) prikazuje osnovne informacije o slobodnom i ukupnom prostoru diska, dok Upravljanje diskom (eng. *Disk Management*) omogućuje detaljan pregled svih particija i njihovih veličina, kao i dodatne mogućnosti upravljanja diskovima, poput proširenja ili smanjenja particija.

Praćenje mjerenih veličina diska ključna je aktivnost za održavanje stabilnosti i performansi računalnih klastera. Razumijevanje koliko brzo se diskovi pune, koliko je slobodnog prostora ostalo te koliko je ukupnog kapaciteta raspoloživo, omogućuje administratorima da pravovremeno reagiraju i spriječe potencijalne probleme s pohranom podataka. Ove mjerne veličine omogućuju učinkovito planiranje budućih nadogradnji diskovnog sustava i osiguravaju da sustav radi optimalno čak i uz stalni rast količine podataka.

3.4. Mjerene veličine ulazno-izlaznih operacija diska

Kontinuirano praćenje ulazno-izlaznih mjerenih veličina diska od presudne je važnosti za razumijevanje trenutnih performansi i opterećenja diskovne infrastrukture unutar računalnih klastera. Ulazno-izlazne operacije diska izravno utječu na brzinu pristupa podacima i ukupnu učinkovitost rada računalnog klastera, posebno kod sustava koji obrađuju velike količine podataka ili zahtijevaju visoku propusnost, poput sustava za analizu velikih podataka, financijskih simulacija ili znanstvenih istraživanja. Optimizacija diskovnih resursa ključna je za održavanje

visoke razine performansi i sprječavanje zastoja, a praćenje ovih mjerenih veličina omogućuje pravovremeno prepoznavanje i rješavanje potencijalnih problema. Ulazno-izlazne mjerene veličine diska obuhvaćaju različite aspekte rada diska, uključujući trenutni broj operacija pisanja i čitanja, brzine tih operacija, kao i broj ulazno-izlaznih operacija po sekundi (eng. *input/output operations per second*, IOPS). Redovno praćenje ovih mjerenih veličina omogućuje administratorima sustava da donesu informirane odluke o optimizaciji aplikacija, prilagodbi resursa ili nadogradnji diskovne infrastrukture.

Osnovne ulazno-izlazne mjerene veličine diska:

1. **Trenutni broj pisanja i čitanja u sekundi** (eng. *Read/Write Throughput*) – predstavlja intenzitet operacija čitanja i pisanja podataka na disk u određenoj jedinici vremena, obično sekundi. Praćenje broja operacija čitanja i pisanja daje uvid u to koliko se disk koristi u stvarnom vremenu, što je ključno za razumijevanje opterećenja diskovnih resursa. U računalnim klasterima koji koriste distribuirane sustave za pohranu podataka, poput Apache Hadoop, kontinuirano visoke vrijednosti propusnosti mogu signalizirati da su diskovi preopterećeni i da je potrebno proširenje kapaciteta ili optimizacija aplikacija. S druge strane, niske vrijednosti mogu signalizirati neiskorištenost resursa, što otvara mogućnost za smanjenje dimenzije sustava ili bolju raspodjelu zadataka kako bi se osiguralo efikasnije korištenje diskovnih kapaciteta. Na primjer, na računalnom klasteru gdje svaki zadatak uključuje često pisanje i čitanje velikih setova podataka s diska, ako broj ulazno-izlaznih operacija konstantno raste, sustav može postati zagušen, što ukazuje na potrebu za skaliranjem ili nadogradnjom diskova s većom propusnošću.
2. **Brzina pisanja i čitanja** (eng. *Read/Write Speed*) – izražava brzinu kojom podaci mogu biti zapisani na disk ili pročitani s diska u jedinici vremena. Visoka brzina čitanja i pisanja ukazuje na dobru izvedbu diskovnih resursa, dok niske brzine često upućuju na uska grla u sustavu. U računalnim klasterima gdje je pristup podacima ključan, poput sustava za strojno učenje koji kontinuirano obrađuju velike skupove podataka, smanjena brzina pisanja ili čitanja može drastično utjecati na cjelokupne performanse. Praćenjem ovih veličina moguće je prepoznati kada sustav doseže svoje granice, te razmotriti prelazak na brže diskovne tehnologije, poput SSD-ova. Na primjer, ako se unutar računalnog klastera uoči pad brzine čitanja, to može signalizirati potrebu za prelaskom s tradicionalnih tvrdih diskova na brže SSD diskove, čime bi se značajno ubrzao proces obrade podataka i omogućilo efikasnije izvršavanje simulacija.

3. **Broj ulazno-izlaznih operacija po sekundi** (IOPS) - mjeri koliko pojedinačnih operacija čitanja ili pisanja disk može obaviti u jednoj sekundi. IOPS je ključna mjerena veličina za ocjenu sposobnosti diska da se nosi s višestrukim paralelnim zahtjevima za pristupom podacima. Veći broj IOPS-a znači da disk može obaviti veći broj operacija u kraćem vremenskom razdoblju, što rezultira boljim ukupnim performansama sustava. Ovo je posebno važno u scenarijima gdje je potrebna visoka propusnost podataka, poput virtualiziranih okruženja ili baze podataka visoke propusnosti. Praćenje IOPS-a pomaže u optimizaciji resursa i izboru odgovarajućih diskovnih rješenja koja će zadovoljiti specifične potrebe računalnog klastera. Na primjer, u računalnom klasteru koji koristi virtualizirane aplikacije ili bazu podataka, visoki IOPS omogućava brzu obradu velikog broja malih operacija čitanja ili pisanja, što je kritično za performanse tih aplikacija. Ako IOPS ne zadovoljava potrebne performanse, moguće je odlučiti o nadogradnji na diskove s većim IOPS-om ili optimizaciji aplikacija kako bi se smanjilo nepotrebno opterećenje diska.

U izrađenom programskom rješenju prikupljaju se podaci o trenutnom broju pisanja i čitanja u sekundi, brzini pisanja i čitanja te broju ulazno-izlaznih operacija po sekundi. Kontinuirano praćenje ovih mjerenih veličina omogućuje optimizaciju diskovnih resursa i pruža pravovremene podatke koji pomažu u identifikaciji potencijalnih problema s performansama. Na primjer, konstantno visoki broj operacija pisanja i čitanja može ukazivati na potrebu za skaliranjem računalnog klastera kako bi se nosio s povećanim opterećenjem, dok niska brzina čitanja i pisanja može signalizirati uska grla u sustavu ili potrebu za prelaskom na diskove s većim performansama. Ako se praćenjem otkrije da su brzine pisanja i čitanja ispod očekivanih vrijednosti, to može ukazivati na preopterećenost diskova ili probleme s optimizacijom aplikacija. Na primjer, ako IOPS podaci pokazuju da trenutni diskovi nisu u stanju obraditi zahtjeve aplikacija, može se odlučiti o nadogradnji na diskove s višim IOPS-om ili optimizaciji aplikacija kako bi se smanjilo opterećenje na diskovnu infrastrukturu.

Linux i Unix sustavi pružaju alate kao što su *iostat, iotop* i *dstat* koji omogućuju praćenje ulazno-izlaznih operacija i performansi diska. Informacije o brzini IO operacija i IOPS-u pruža *iostat,* dok *iotop* i *dstat* omogućuju uvid u ulazno-izlazne aktivnosti i opterećenje diska u stvarnom vremenu. Windows operacijski sustav pruža alat Upravitelj performansi (eng. *Performance monitor*) koji nudi mogućnost praćenja različitih ulazno-izlaznih mjerenih veličina, uključujući brzine pisanja i čitanja te IOPS, dok Upravitelj resursima (eng. *Resource Monitor*) pruža detaljne informacije o ulazno-izlaznim operacijama na disku i performansama u stvarnom vremenu.

Praćenje ulazno-izlaznih mjerenih veličina diska ključno je za održavanje optimalnih performansi i opterećenja diskovne infrastrukture u računalnim klasterima. Razumijevanje trenutnog broja pisanja i čitanja, brzina tih operacija te broja ulazno-izlaznih operacija po sekundi omogućuje pravovremeno prepoznavanje problema i omogućuje donošenje informiranih odluka o optimizaciji ili nadogradnji diskovnih resursa. Ove mjerene veličine ključne su za osiguranje stabilnosti računalnih klastera i njegovih aplikacija, kao i za dugoročnu održivost sustava u okruženjima s visokim zahtjevima za performansama.

3.5. Mjerene veličine mrežnih uređaja

Mjerene veličine mrežnih uređaja od presudne su važnosti za razumijevanje i optimizaciju performansi komunikacije među čvorovima u računalnim klasterima. Učinkovitost mreže izravno utječe na cjelokupne performanse računalnog klastera, budući da se pouzdana i brza razmjena podataka između čvorova temelji na mrežnoj infrastrukturi. Kod aplikacija koje koriste paralelno računalstvo ili distribuirane sustave, mrežni resursi postaju ključni faktor za osiguravanje visokih performansi. Pravovremeno praćenje mrežnih performansi omogućuje rano prepoznavanje mogućih uskih grla i problema s komunikacijom, čime se održava optimalna stabilnost sustava. Ključne mjerne veličine za analizu mrežnih performansi obuhvaćaju propusnost mreže, latenciju i gubitak paketa, koji su nužni za identifikaciju uskih grla, kašnjenja ili problema u prijenosu podataka. Te veličine omogućuju detaljan uvid u rad mreže i pomažu u donošenju odluka o potrebnim nadogradnjama ili optimizaciji mrežne infrastrukture.

Osnovne mjerene veličine mreže:

1. **Propusnost** (eng. *Throughput*) – predstavlja stvarnu brzinu prijenosa podataka kroz mrežu i obično se izražava u megabitima po sekundi (Mbps) ili gigabitima po sekundi (Gbps). Visoka propusnost je ključna za računalne klastere koji obrađuju velike količine podataka, poput znanstvenih simulacija, analize velikih podataka ili strojnog učenja. Na primjer, u sustavima koji razmjenjuju velike datoteke između čvorova, niska propusnost može dovesti do zagušenja mreže, što usporava obradu podataka i može negativno utjecati na cijelu aplikaciju. Praćenje propusnosti pomaže u otkrivanju potencijalnih uskih grla u mreži i omogućuje pravovremeno proširenje mrežne infrastrukture, kao što su dodavanje novih mrežnih veza ili nadogradnja mrežnih preklopnika, kako bi se povećala brzina prijenosa podataka..

2. **Latencija** (eng. *Latency*) – predstavlja vrijeme potrebno da paket podataka putuje od izvora do odredišta. Izražava se u milisekundama (ms) ili mikrosekundama (μs). Niska latencija od presudne je važnosti za aplikacije koje zahtijevaju pravovremenu komunikaciju, kao što su simulacije u stvarnom vremenu, paralelno računalstvo ili distribuirane baze podataka. Na primjer, u računalnim klasterima koji izvode simulacije, visoka latencija može značiti kašnjenja u sinkronizaciji podataka među čvorovima, što može usporiti cijeli izračun i dovesti do nepreciznih rezultata. Praćenjem latencije moguće je identificirati problematične mrežne segmente i optimizirati mrežnu konfiguraciju kako bi se osigurala pravovremena i brza komunikacija između čvorova.
3. **Gubitak paketa** (eng. *Packet Loss*) – nastaje kada jedan ili više paketa ne stignu do odredišta i obično se izražava kao postotak od ukupnog broja poslanih paketa. Čest gubitak paketa može značajno utjecati na performanse mreže, jer dovodi do ponovnih prijenosa podataka i povećava latenciju. Na primjer, u klasterima koji koriste distribuirane sustave za pohranu podataka, poput HDFS-a, praćenje mrežnog prometa može ukazivati na probleme u izvođenju aplikacije, kao što je preopterećenje mreže. Ako se otkrije visoka latencija ili značajan gubitak paketa, to može sugerirati da mrežni promet unutar aplikacije nije optimalno konfiguriran ili da je sama aplikacija mrežno zahtjevna. Ovo praćenje omogućuje administratorima da identificiraju potrebu za optimizacijom mrežnog prometa ili razmatranje nadogradnje mrežne infrastrukture kako bi se poboljšale performanse aplikacije i osigurala bolja učinkovitost sustava.

U izrađenom programskom rješenju prikupljaju se samo podaci o mrežnoj propusnosti. Kontinuirano praćenje ovih podataka omogućuje pravovremeno prepoznavanje i rješavanje potencijalnih problema u mrežnoj infrastrukturi. Na primjer, ako se uoči da propusnost mreže ne zadovoljava potrebe aplikacija, to može signalizirati potrebu za nadogradnjom mrežne opreme ili povećanjem broja mrežnih veza. Nadalje, ako se praćenjem otkrije visoka latencija ili značajan gubitak paketa, administrator sustava će koristiti specijalizirane mrežne alate za analizu mrežnog prometa i infrastrukture. Ovi alati omogućuju detaljno ispitivanje mrežnih postavki, preklopnika i kablova kako bi se identificirali problematični segmenti i osigurao nesmetan rad sustava.

Ako se, primjerice, konstantno bilježe visoki postoci gubitka paketa između čvorova, moguće je da je riječ o preopterećenju mrežne opreme ili lošoj konfiguraciji mrežnih postavki. U takvim slučajevima, nadogradnja mrežnih preklopnika ili optimizacija mrežnih konfiguracija može poboljšati performanse. Na sličan način, ako se propusnost mreže pokaže kao glavni

ograničavajući faktor, može se razmotriti dodavanje dodatnih mrežnih veza ili nadogradnja na brže mrežne standarde kako bi se osigurala brža komunikacija između čvorova i smanjila latencija.

U Unix i Linux sustavima, alati poput *netstat*, *iftop*, *ping* i *traceroute* omogućuju korisnicima da prate propusnost, latenciju i gubitak paketa. Na primjer, *netstat* i *iftop* omogućuju pregled mrežnog prometa i propusnosti, pružajući detaljan uvid u trenutne mrežne performanse, dok se *ping* i *traceroute* koriste za mjerenje latencije i identificiranje problematičnih mrežnih segmenata. S druge strane, Windows operacijski sustav nudi alate kao što su Upravitelj performansi i Upravitelj resursima koji omogućuju nadzor mrežnih mjerenih veličina. Upravitelj performansi nudi mogućnost praćenja mrežnih veličina, uključujući propusnost, latenciju i gubitak paketa, dok Upravitelj resursima daje detaljan pregled mrežnog prometa i omogućuje analizu performansi sustava u stvarnom vremenu.

Praćenje mrežnih mjerenih veličina, kao što su propusnost, latencija i gubitak paketa, ključno je za održavanje optimalnih performansi komunikacije unutar računalnih klastera. Pravilno razumijevanje i praćenje ovih mjerenih veličina omogućuje pravovremeno prepoznavanje uskih grla i problema s mrežom, čime se osigurava stabilnost mrežne infrastrukture i optimiziraju performanse računalnih klastera. Na taj način, mrežni resursi mogu biti učinkovito iskorišteni, a računalni klaster može raditi nesmetano i na maksimalnoj razini performansi.

3.6. Ostale mjerene veličine

Pored osnovnih mjerenih veličina poput opterećenja procesora, zauzeća memorije, performansi diskova i mrežnog prometa, računalni klasteri generiraju i druge mjerne veličine koje, iako nisu od presudne važnosti u svakodnevnim operacijama, mogu biti ključne u specifičnim scenarijima. Ove mjerne veličine, koje nadziru fizička svojstva hardverskih komponenti ili pružaju dodatne detalje o operacijama sustava, omogućuju administratorima dublje razumijevanje kompleksnog ponašanja računalnih klastera. U praksi, ove veličine mogu otkriti potencijalne probleme s toplinskim opterećenjem, energetskom učinkovitošću, stabilnošću napajanja, ili mrežnom infrastrukturom, što može biti presudno za optimizaciju i održavanje stabilnosti cijelog sustava.

U velikim distribuiranim sustavima, poput HPC sustava, računalna infrastruktura može uključivati stotine ili tisuće čvorova. Praćenje samo osnovnih mjerenih veličina nije dovoljno za osiguranje optimalnih performansi i prevenciju kvarova. U stvarnim scenarijima, aplikacije koje zahtijevaju intenzivne računalne resurse, kao što su simulacije fizikalnih procesa, analize

genetskih podataka ili velike financijske simulacije, mogu lako opteretiti procesore, memoriju i diskove. U takvim slučajevima, nadzor dodatnih mjerenih veličina može pomoći u održavanju ravnoteže između performansi i pouzdanosti. U heterogenim računalnim klasterima, gdje se koristi različit hardver unutar istog sustava, ove manje važne mjerene veličine mogu otkriti razlike u načinu na koji različite komponente reagiraju na opterećenje, omogućujući programerima i administratorima da precizno prilagode konfiguracije kako bi optimizirali performanse u cijelom sustavu. Iako ostale mjerene veličine imaju veliku korist pri održavanju i upravljanju računalnim klasterima, u razvijenom programskom rješenju neće se prikupljati i vizualizirati zbog složenosti implementacije i manje važnosti pri upravljanju jednostavnim računalnim klasterima.

Ostale mjerene veličine koje pružaju dodatne korisne informacije uključuju:

1. **Temperatura procesora** – procesori često rade na visokim frekvencijama i pod velikim opterećenjem, što može dovesti do značajnog zagrijavanja. Iako moderni procesori imaju ugrađene mehanizme zaštite poput toplinskog prigušivanja (eng. *Thermal throttling*), gdje se smanjuje radna frekvencija procesora kako bi se spriječilo pregrijavanje, visoke temperature mogu uzrokovati degradaciju performansi i smanjiti životni vijek hardvera. Primjerice, u HPC okruženjima gdje se provode simulacije ili izračuni koji traju danima, temperatura procesora može postati kritičan faktor. Ako jedan ili više čvorova konstantno radi na graničnim temperaturnim vrijednostima, sustav može usporiti zbog automatskih mehanizama hlađenja. Kontinuirano praćenje ove mjerne veličine omogućava pravovremenu intervenciju, poput optimizacije sustava hlađenja ili redistribucije zadataka na manje opterećene čvorove.
2. **Frekvencija procesora** – računalni klasteri često koriste dinamičku prilagodbu frekvencije procesora, odnosno DVFS (eng. *Dynamic Voltage and Frequency Scaling*, DVFS), kako bi smanjili potrošnju energije kada nisu potpuno opterećeni. U situacijama kada su zahtjevi za računalnom snagom visoki, procesori će raditi na maksimalnoj frekvenciji, ali ako je opterećenje niže, frekvencija se automatski smanjuje kako bi se uštedjela energija i smanjila toplinska disipacija. Praćenje promjena u frekvenciji procesora može otkriti probleme u performansama koje inače ne bi bile očite iz drugih mjerenih veličina, osobito u heterogenim okruženjima gdje različiti čvorovi mogu imati različite frekvencijske kapacitete. Na primjer, u scenarijima gdje jedan čvor kontinuirano radi na smanjenoj frekvenciji, a ostali na maksimalnoj, to može signalizirati problem s napajanjem, hlađenjem ili opterećenjem tog čvora.

3. **Napon procesora** – direktno utječe na stabilnost i performanse, osobito pri velikim opterećenjima i visokim frekvencijama. U računalnim klasterima koji koriste *overclocking* za povećanje performansi, precizno praćenje napona je ključno za sprječavanje sistemskih nestabilnosti i hardverskih kvarova. Nagli padovi ili oscilacije napona mogu dovesti do pada sustava ili oštećenja hardvera, što je kritično u računalnim klasterima s velikim brojem čvorova, gdje gubitak jednog čvora može negativno utjecati na cijeli zadatak.
4. **Temperatura diska** – diskovi, bilo da se radi o SSD-ovima ili klasičnim HDD-ovima, pod velikim opterećenjem mogu postati osjetljivi na pregrijavanje, osobito u okruženjima s konstantnim zapisivanjem ili čitanjem velikih količina podataka. Visoka temperatura diska može uzrokovati degradaciju performansi, povećanu latenciju ili čak trajne gubitke podataka. Na primjer, u računalnim klasterima koji koriste diskove za spremanje privremenih podataka pri simulacijama, pregrijavanje može dovesti do povećanog broja ulazno-izlaznih grešaka ili sporog odgovora aplikacija. Praćenjem temperature diska, administratori mogu poduzeti preventivne mjere, kao što su optimizacija pohrane podataka ili poboljšanje hlađenja.
5. **Broj otvorenih datoteka** – računalni klasteri često izvode aplikacije koje otvaraju velik broj datoteka. Ako broj otvorenih datoteka premašuje kapacitet operacijskog sustava, to može uzrokovati uska grla i padove aplikacija. U distribuiranim sustavima, poput onih koji koriste MPI (eng. *Message Passing Interface*, MPI) za paralelnu komunikaciju među čvorovima, praćenje ove mjerene veličine može otkriti situacije u kojima aplikacije neispravno zatvaraju datoteke, što rezultira curenjem resursa. Ovo je posebno važno za dugotrajne procese, gdje bi curenje resursa moglo ugroziti stabilnost cijelog računalnog klastera.

3.7. Razlika mjerenih veličina pojedinog čvora i kumulativnih mjerenih veličina računalnog klastera

Prikupljanje mjerenih veličina za pojedinačne čvorove i kumulativnih mjerenih veličina za cijeli računalni klaster pruža komplementarne informacije koje su ključne za efikasno upravljanje i održavanje računalnog klastera. Svaka od ovih vrsta mjerenih veličina ima svoje specifične prednosti i primjene te pruža uvid u to kako pojedinačni čvorovi doprinose ukupnim performansama računalnog klastera, omogućujući optimizaciju iskorištenosti resursa radi postizanja maksimalne učinkovitosti i pouzdanosti sustava.

Mjerene veličine prikupljene za pojedinačne čvorove odnose se na specifične performanse i resurse svakog čvora unutar računalnog klastera. Ove mjerene veličine uključuju podatke poput opterećenja procesora, iskorištenosti memorije, brzine mrežnih veza te brzine I/O operacija na disku. Detaljno praćenje ovih mjerenih veličina omogućuje identifikaciju i rješavanje problema koji se odnose na specifične čvorove, kao što su preopterećenje procesora ili nedostatak memorije. Na primjer, visoko opterećenje procesora na jednom čvoru može ukazivati na potrebu za optimizacijom aplikacija koje se izvode na tom čvoru ili redistribucijom radnog opterećenja na druge, manje opterećene čvorove.

S druge strane, kumulativne mjerene veličine predstavljaju agregirane podatke prikupljene sa svih čvorova u računalnom klasteru. Ove mjerene veličine uključuju ukupnu iskorištenost procesora, prosječnu količinu memorije koju svi čvorovi koriste, ukupnu brzinu čitanja i pisanja podataka na diskovima te prosječno vrijeme kašnjenja u mrežnoj komunikaciji. Takvi agregirani podaci omogućuju pregled ukupnih performansi računalnih klastera i pomažu u otkrivanju globalnih problema, kao što su uska grla u mrežnoj komunikaciji ili potreba za povećanjem kapaciteta diska. Na primjer, ako kumulativne mjerene veličine pokazuju da ukupno opterećenje procesora na svim čvorovima približava maksimalnom kapacitetu, to može signalizirati potrebu za dodavanjem novih čvorova ili redistribucijom zadataka kako bi se poboljšale ukupne performanse.

Metodologija za prikupljanje i analizu ovih mjerenih veličina može se razlikovati, s posebnim alatima i tehnikama za agregaciju podataka na razini računalnih klastera. Ključna prednost praćenja oba tipa mjerenih veličina je mogućnost korelacije podataka s pojedinih čvorova i kumulativnih mjerenih veličina cijelog računalnog klastera. Na taj način administratori mogu brzo identificirati koji čvorovi uzrokuju uska grla ili pogoršavaju ukupne performanse. Na primjer, ako određeni čvor konstantno bilježi visoku potrošnju memorije, to može negativno utjecati na kumulativnu vrijednost iskorištenosti memorije u cijelom računalnom klasteru, što može biti znak da je potrebna redistribucija memorijskih resursa ili optimizacija zadataka koji troše previše memorije. Praćenje ovih podataka u različitim vremenskim intervalima također je važno za detaljnu analizu performansi. Kumulativni podaci prikupljeni kroz dulje vremenske periode mogu sakriti povremena uska grla, dok praćenje podataka u kraćim intervalima omogućuje precizniji uvid u trenutne performanse i stanje računalnog klastera. To omogućuje administratorima da identificiraju uzroke povremenih problema koji možda nisu vidljivi u dugoročnim analizama.

Kontinuirano praćenje pojedinačnih i kumulativnih mjerenih veličina omogućuje učinkovito upravljanje resursima i pravovremeno prepoznavanje problema, čime se osigurava optimalna funkcionalnost i stabilnost računalnog klastera. U izrađenom programskom rješenju mjerene veličine pojedinog čvora prikupljaju se direktno s čvorova, dok se kumulativne mjerene veličine agregiraju na *frontend* računalu, što omogućuje sveobuhvatnu analizu i optimizaciju računalnog klastera u realnom vremenu. Ukupna dostupna memorija i iskorištena memorija računalnog klastera dobiva se sumom ukupne, odnosno iskorištene, memorije pojedinog čvora te se preračunava u gigabajte. Trenutna propusnost mreže računalnog klastera dobiva se zbrajanjem vrijednosti očitanih s pojedinog čvora. Slično tome, ukupne ulazno-izlazne operacije diska dobivaju se zbrajanjem pojedinih vrijednosti s čvorova. Opterećenje procesora određuje se računanjem prosječnog opterećenja čvorova, odnosno zbroj vrijednosti pojedinog opterećenja podijeljen s brojem čvorova.

4. PROGRAMSKO RJEŠENJE

Razlog izrade programskog rješenja izrađenog u okviru ovog rada je potreba za jednostavnim i lako primjenjivim alatom za praćenje opterećenja računalnih klastera. Glavna zadaća programskog rješenja je efikasno prikupljanje mjerenih veličina s čvorova te jasna vizualizacija prikupljenih mjerenih veličina. Programsko rješenje se mora jednostavno instalirati i postaviti te samo korisničko iskustvo mora biti intuitivno. Mjerene veličine koje programsko rješenje prikuplja su opterećenost procesora, korisničko i sustavno vrijeme procesora, prosječno opterećenje sustava, zauzeće i dostupnost memorije i diska te osnovne ulazno-izlazne mjerene veličine diska i propusnost mreže. Prikupljene mjerene veličine detaljno su objašnjenje u poglavlju 3.

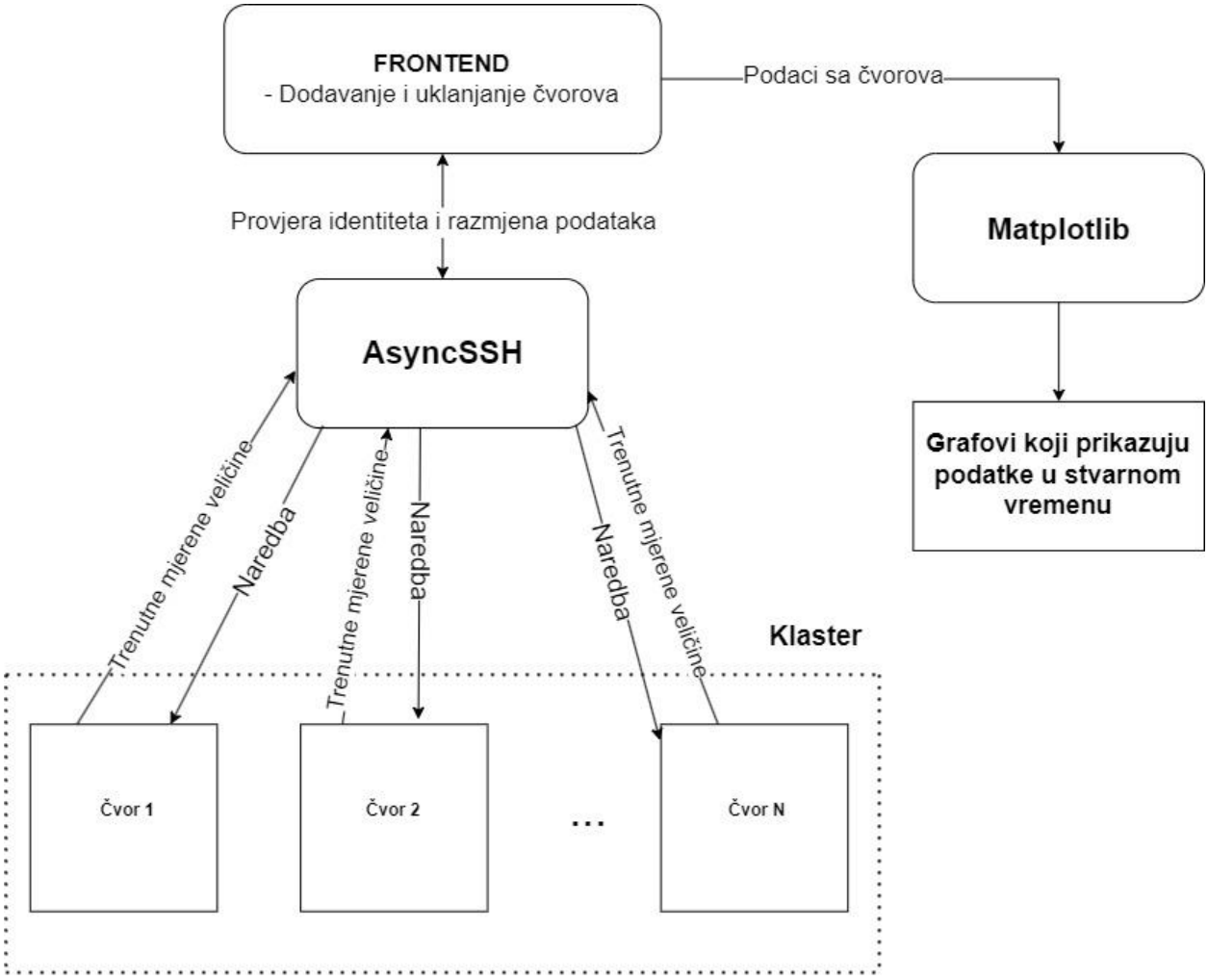
Zahtjevi na programsko rješenje uključuju:

- Periodičko prikupljanje mjerenih veličina za svaki čvor u računalnom klasteru.
- Računanje kumulativnih mjerenih veličina računalnog klastera.
- Vizualizaciju prikupljenih mjerenih veličina putem grafičkog korisničkog sučelja.
- Osvježavanje prikaza prikupljenih mjerenih veličina u stvarnom vremenu.
- Dodavanje, uklanjanje i konfiguriranje čvorova unutar korisničkog sučelja.
- Detektiranje neispravnih čvorova te prikladan prikaz neispravnih čvorova.
- Definiranje različitih računalnih klastera kroz konfiguracijske datoteke.
- Dinamičku promjenu konfiguracijske datoteke bez potrebe za ponovnim pokretanjem sustava.

Arhitektura rješenja temelji se na klijent-poslužitelj modelu, gdje se mjerene veličine prikupljaju iz svakog čvora putem SSH protokola, a centralni poslužitelj obrađuje i vizualizira podatke. Svaki čvor unutar računalnog klastera dostupan je putem SSH protokola, što omogućuje aplikaciji da im daljinski pristupi i prikuplja mjerene veličine bez potrebe za direktnim fizičkim pristupom.

Programsko rješenje razvijeno je u programskom jeziku Python. Inspiracija za značajke aplikacije i vizualizaciju podataka dolazi od sustava Ganglia, koji je detaljno opisan u potpoglavlju 2.1. Cilj aplikacije je omogućiti korisnicima uvid u performanse i opterećenje pojedinih čvorova računalnog klastera, kao i cijelog računalnog klastera, putem intuitivnog i informativnog korisničkog sučelja bez dodatnog *overheada*. Razvijena aplikacija namijenjena je za nadziranje čvorova klastera računala na kojim je instaliran operacijski sustav Linux. Aplikacija je kompatibilna sa svim distribucijama Linuxa koje podržavaju osnovne Linux ili Unix naredbe,

poput *top, df*, i drugih sličnih. Također, aplikacija zahtijeva standardni Linux datotečni sustav za pravilno funkcioniranje. Izvršavanje aplikacije zahtijeva *frontend* računalo s pristupom čvorovima računalnog klastera, koje omogućuje prikupljanje i obradu podataka. Izvorni kod programskog rješenja zajedno s uputama za pokretanje dostupan je u prilogu. Slika 4.1. predstavlja sistematični prikaz programskog rješenja na kojoj su vidljivi osnovni elementi aplikacije i kako su povezani.



Slika 4.1. Sistematični prikaz programskog rješenja

4.1. Korištene tehnologije i biblioteke

Za komunikaciju s čvorovima računalnog klastera koristi se SSH protokol. SSH omogućuje sigurno udaljeno prijavljivanje i izvršavanje naredbi na udaljenim računalima koristeći kriptografiju za zaštitu podataka. Komunikacija između uređaja je šifrirana, što sprječava presretanje podataka i neovlašteni pristup. Šifriranje koristi simetrične ključeve za zaštitu poruka, dok se za inicijalnu razmjenu ključeva i provjeru identiteta koriste asimetrični ključevi ili lozinke.

Asimetrični ključevi, koji uključuju javni i privatni ključ, pružaju veću sigurnost od lozinki. Javni ključ se dijeli i pohranjuje na udaljenim čvorovima, dok privatni ključ ostaje na lokalnom računalu. U programskom rješenju izrađenom u okviru ovog rada implementirana su oba načina provjere identiteta.

Prednost korištenja Python programskog jezika je bogat ekosustav biblioteka za obradu i vizualizaciju podataka. Omogućava razvoj složenih funkcionalnosti s relativno malo koda, dok istovremeno osigurava visoku razinu pouzdanosti i performansi. Jedan od glavnih izazova pri razvoju aplikacije za nadziranje klastera računala u stvarnom vremenu je održavanje performansi aplikacije rastom broja nadziranih čvorova. Python podržava razne metode za poboljšanje performansi, a jedna od njih je biblioteka *asyncio* koja je korištena u aplikaciji. Kao što je navedeno pod [10] *asyncio* je biblioteka za pisanje koda koji se izvršava istovremeno pomoću asinkronih funkcija. Funkcije prikupljanja mjerenih veličina s udaljenih čvorova su zapravo ulazno-izlazne operacije te je *asyncio* napravljen za istodobno izvršavanje ulazno-izlaznih operacija bez blokiranja, što znači da programu omogućava da nastavi s radom dok čeka izvršenje ulazno-izlazne operacije. Za implementaciju SSH protokola korištena je *asyncssh* biblioteka. Prema [11], *asyncssh* omogućuje asinkronu upotrebu SSH protokola u kombinaciji s *asyncio* bibliotekom. Uspostavljanje veza s udaljenim čvorovima te izvršavanje naredbi na asinkron način vrlo je korisno u razvijenoj aplikaciji zbog mogućeg velikog broja čvorova i istovremenih poziva funkcija prikupljanja mjerenih veličina.

Korisničko sučelje aplikacije pruža uvid u čvorove računalnog klastera te grafove za pojedine mjerene veličine u stvarnom vremenu. Biblioteka koja je korištena za izradu korisničko sučelja je *tkinter*. Kao što je navedeno pod [10] *tkinter* je standardna biblioteka Pythona za izradu grafičkog sučelja aplikacije. U biblioteci se nalaze razne predefinirane komponente poput okvira, gumbova, oznaka, teksta i slično. Pomoću tih komponenti jednostavno je izraditi raznovrsna i funkcionalna korisnička sučelja. *Tkinter* također podržava teme, koje komponentama mogu dati moderan izgled ili jedinstven stil, no nisu korištene u razvijenoj aplikaciji. Pored samog korisničko sučelja potrebno je izraditi i grafove prikupljenih mjerenih veličina kako bi korisnici imali jasan uvid u ponašanje čvorova klastera računala. U svrhu kreiranja grafova korištena je biblioteka *matplotlib*. Prema [12], *matplotlib* je biblioteka u Pythonu koja omogućava izradu raznovrsnih grafova, dijagrama i vizualizacija podataka. Pruža širok spektar mogućnosti za prikaz podataka u obliku linija, histograma, stupčastih dijagrama, raspršenih dijagram i mnogih drugih tipova grafova. U razvijenoj aplikaciji najčešće se koriste linijski grafovi, uz izuzetak dva kružna grafa za prikazivanje zauzeća memorije i diska.

4.2. Princip rada prikupljanja mjerenih veličina

Kao što je prije navedeno za komunikaciju s udaljenim čvorovima računalnog klastera koristi se SSH protokol. Protokol omogućuje izvođenje naredbi na čvorovima računalnog klastera. Korištenjem prikladnih naredbi moguće je dohvatiti trenutne mjerene veličine čvora, odnosno prije navedene prikladne mjerene veličine koje ukazuju na sveukupne performanse čvora. Razvijena aplikacija periodično prikuplja mjerene veličine procesora, mjerene veličine memorije, mjerene veličine vezane uz sustave za pohranu podataka, ulazno-izlazne mjerene veličine diska te mjerene veličine mreže. Funkcije koje se koriste za prikupljanje mjerenih veličina dio su korisničkih alata koji dolaze s gotovo svim distribucijama Linuxa. Rezultati funkcija šalju se na *frontend* računalo i obrađuju, čime je teži posao prepušten *frontend* računalu.

Za prikupljanje mjerenih veličina procesora koristi se naredba *top*. Prema [13], naredba *top* daje dinamički uvid u sustav. Prikazuje popis procesa koji se trenutno izvode, detalje opterećenja procesora te prosjeke opterećenja u periodu vremena. Razvijeno programsko rješenje periodično izvršava naredbu na čvorovima računalnog klastera. Izlistanje naredbe se vraća *frontend* računalu, gdje se obrađuje kako bi se odredile mjerene veličine poput opterećenja procesora, postotaka vremena provedenog za izvršavanje korisničkih i sustavskih procesa, te prosječnog opterećenja procesora tijekom različitih vremenskih perioda (1, 5 i 15 minuta). Ove mjerene veličine su detaljno objašnjene u potpoglavlju 3.1. Kako je izlistanje naredbe uvijek istog formata, na *frontend* računalu se čitaju potrebne linije izlistanja s čvora i iz njih se dobiju trenutne mjerene veličine procesora. Slika 4.2. prikazuje primjer rezultata izvršavanja naredbe *top*. U prvom redu nalaze se prosječna opterećenja tijekom određenog vremenskog perioda dok treći red prikazuje mjerene veličine opterećenja procesora. Ispod bijele linije se nalaze trenutni procesi koji se izvršavaju te njihov utjecaj na procesor i memoriju. U rezultatu se također nalazi i zauzeće memorije, no razvijena aplikacija koristi drugu naredbu za prikupljanje mjerenih veličina memorije.

```
top - 21:17:39 up 12 days,  5:52,  1 user,  load average: 0.18, 0.09, 0.06
Tasks: 134 total,   1 running, 133 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :    922.0 total,    576.6 free,     68.9 used,    276.5 buff/cache
MiB Swap:    100.0 total,    100.0 free,      0.0 used.    794.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
21004 rpi       20   0   11212   3160   2628 R   0.7   0.3   0:00.16 top
 3275 rpi       20   0   14512   4324   3416 S   0.3   0.5   0:00.09 sshd
    1 root      20   0   33872   8720   6916 S   0.0   0.9   0:35.86 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:02.31 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
   10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
   11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
   12 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthread
   13 root      20   0       0      0      0 S   0.0   0.0   0:36.26 ksoftirqd/0
   14 root      20   0       0      0      0 I   0.0   0.0   1:00.50 rcu_sched
   15 root      rt   0       0      0      0 S   0.0   0.0   0:00.67 migration/0
   16 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   17 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
   18 root      rt   0       0      0      0 S   0.0   0.0   0:02.46 migration/1
   19 root      20   0       0      0      0 S   0.0   0.0   0:23.03 ksoftirqd/1
   22 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
   23 root      rt   0       0      0      0 S   0.0   0.0   0:02.42 migration/2
   24 root      20   0       0      0      0 S   0.0   0.0   0:07.53 ksoftirqd/2
   27 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/3
   28 root      rt   0       0      0      0 S   0.0   0.0   0:02.34 migration/3
```

Slika 4.2. Rezultat naredbe *top*

Prikupljanje mjerenih veličina memorije izvršava se pomoću naredbe *free*. Kao što je navedeno pod [13] naredba *free* služi za prikazivanje slobodne i zauzete memorije sustava. Prilikom izvršavanja naredbe zadana mjerna jedinica za prikaz memorije je *kibibyte*. Programsko rješenje periodično izvršava naredbu *free* na čvorovima te izlistanje naredbe šalje na *frontend* računalo. Dobiveno izlistanje se obrađuje korištenjem regularnih izraza te se određuju mjerene veličine memorije. Slika 4.3. prikazuje primjer rezultata izvršavanja naredbe *free*. Prikazane su vrijednosti za fizičku i *swap* memoriju sustava, a u aplikaciji se koriste vrijednosti ukupne, iskorištene i slobodne memorije.

```
rpi@rpi:~ $ free
              total        used        free      shared  buff/cache   available
Mem:         944092       70580      590404        3356      283108      813740
Swap:        102396           0      102396
```

Slika 4.3. Rezultat izvršavanja naredbe free

Mjerene veličine vezane uz sustave za pohranu podataka prikupljaju se naredbom *df*. Prema [13], naredba *df* prikazuje informacije o datotečnom sustavu čvora. Za svaki priključeni tip pohrane koji postoji na čvoru prikazuje podatak o zauzeću, slobodnom prostoru te postotku zauzeća. Programsko rješenje periodično izvršava *df* naredbu na čvorovima računalnog klastera. Izlistanje naredbe se šalje na *frontend* računalo gdje se obrađuje kao tekst liniju po liniju. Za svaki priključeni tip pohrane određuju se mjerene veličine. Slika 4.4. prikazuje primjer rezultata

izvršavanja naredbe *df*. U ovome primjeru *mmcblk0p1* predstavlja memorijsku karticu, dok se ostali tipovi pohrane mogu prikazivati drugim identifikatorima.

```
rpi@rpi:~ $ df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/root       14627712 4241684   9759152  31% /
devtmpfs          340460       0    340460   0% /dev
tmpfs             472044       0    472044   0% /dev/shm
tmpfs             188820     964    187856   1% /run
tmpfs               5120       8      5112   1% /run/lock
/dev/mmcblk0p1    261108   51636    209472  20% /boot
tmpfs              94408       0     94408   0% /run/user/1000
```

Slika 4.4. Rezultat naredbe df

Prikupljanje ulazno-izlaznih mjerenih veličina diska razlikuje se od ostalih zato što samo prikupljanje mjerenih veličina traje barem jednu sekundu. Ulazno-izlazne mjerene veličine diska izražavaju se u mjernoj jedinici po sekundi te je zato potrebno dva puta izvršiti naredbu i izračunati razliku. Za dohvaćanje mjerenih veličina koristi se *cat* naredba koja, prema [13], služi za čitanje datoteka. Datoteka koja sadrži nalazi se pod */proc/diskstat* te je standardna datoteka operacijskog sustava Linux. Programsko rješenje iz datoteke prikuplja broj čitanja i pisanja po sekundi, brzinu čitanja i pisanja po sekundi te IOPS.

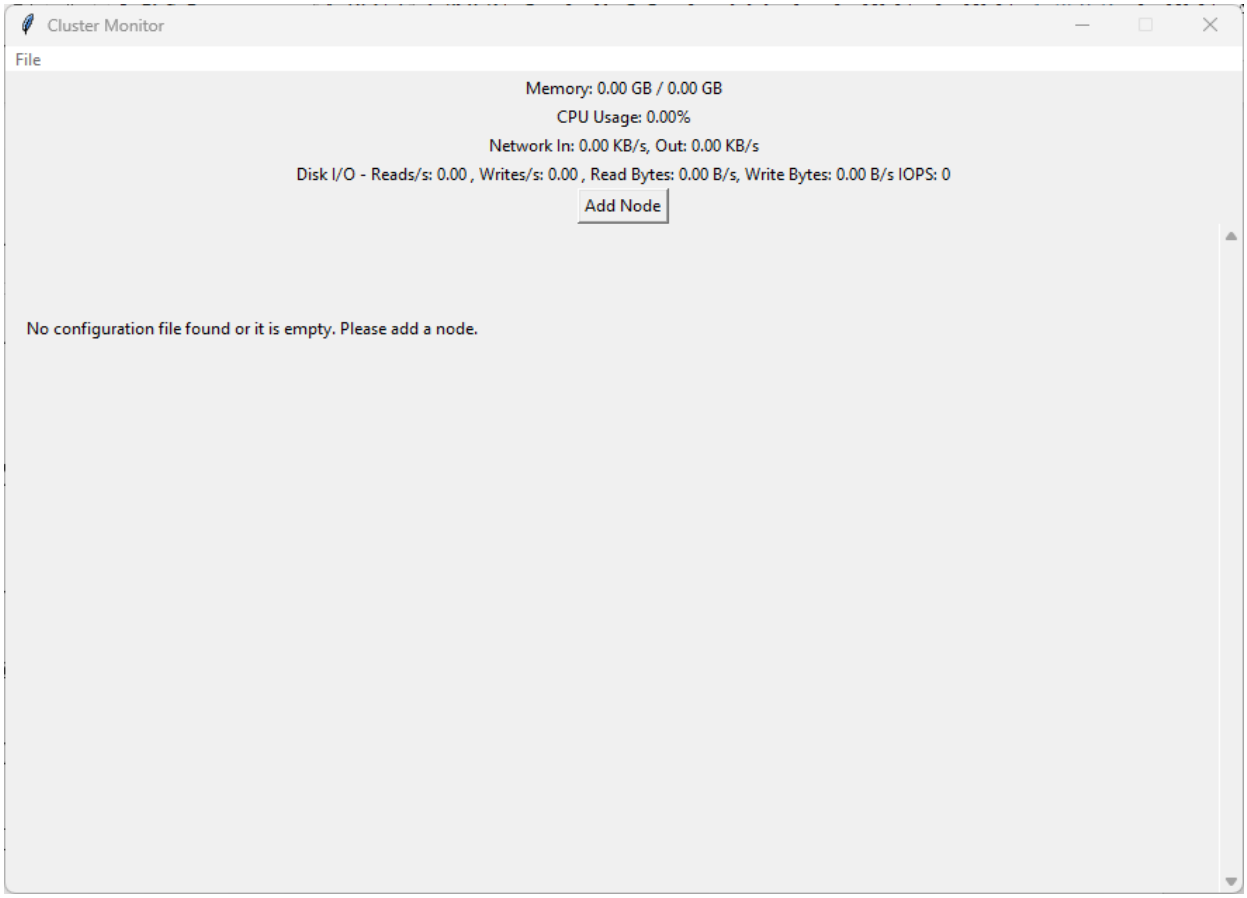
Mjerene veličine mrežnih uređaja, isto kao i ulazno-izlazne mjerene veličine diska, je potrebno prikupljati u vremenskom intervalu od jedne sekunde. Naredbu za prikupljanje je potrebno izvršiti dva puta i izračunati razliku zato što se mjerene veličine mrežnih uređaja izražavaju u mjernoj jedinici po sekundi. Koristi se ista *cat* naredba no datoteka koja sadrži potrebne mjerene veličine je */proc/net/dev*. Izračunata razlika izvršavanja naredbi u vremenskim periodima predstavlja potrebne mjerene veličine. Aplikacija prikuplja mjerene veličine propusnosti mreže, odnosno brzine kojima se podaci šalju van i primaju na čvor.

Pored periodičnih mjerenih veličina koje se prikupljaju u stvarnom vremenu za svaki čvor se jednom prikupljaju osnovne informacije sustava. Te informacije uključuju ime čvora, ukupnu količinu memorije i prostora na disku, ime distribucije Linuxa, verziju jezgre, ime procesora, broj jezgri te arhitekturu procesora.

4.3. Značajke aplikacije

Razvijena aplikacija ima veći broj korisnih značajki. Slika 4.5. prikazuje glavni zaslon aplikacije kada se prvi puta pokrene. Kao što je vidljivo prikazana je poruka da trenutno nisu dodani čvorovi računalnog klastera. Jedna od značajki aplikacije je mogućnost konfiguracije

datoteke koja sadrži potrebne informacije o računalnom klasteru. Datoteka je u standardnom JSON formatu i sadrži ime čvora, adresu čvora te odabrani način SSH provjere identiteta. Prednost ovakvog načina upravljanja čvorovima je mogućnost kreiranje više konfiguracijskih datoteka za pojedini računalni klaster. Konfiguracijsku datoteku moguće je napraviti kroz aplikaciju ili ručno pisanjem informacija za svaki čvor. Slika 4.6. prikazuje primjer JSON konfiguracije za jedan čvor. U njemu su navedene potrebne informacije o čvoru poput imena i adrese čvora, korisničkog imena te načina SSH provjere identiteta. U ovome primjeru odabrana je provjera identiteta ključem te je definirana putanja do istoga.

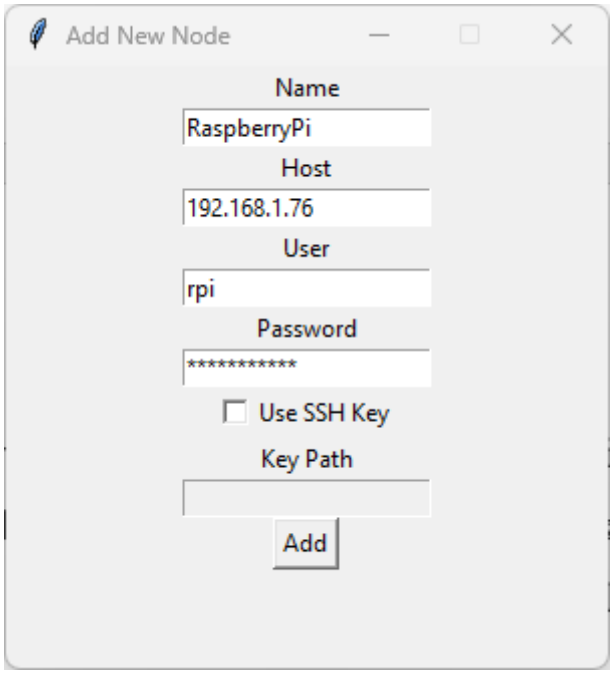


Slika 4.5. Početni zaslon aplikacije

```
[
    {
        "name": "rocky",
        "host": "192.168.1.76",
        "user": "rockylinux",
        "password": "",
        "use_key": true,
        "key_path": "/home/tomislav/.ssh/id_rsa"
    }
]
```

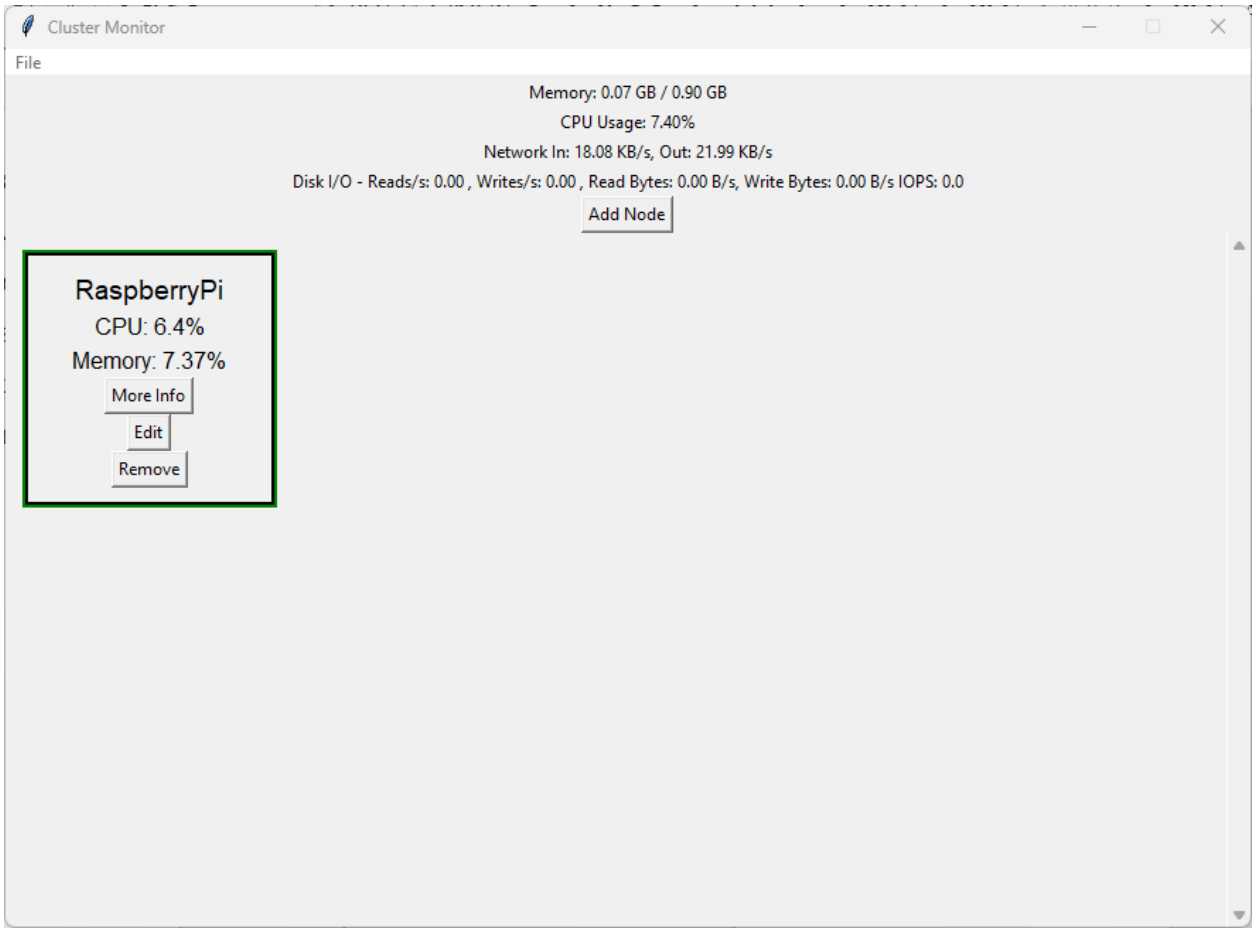
Slika 4.6. Primjer JSON konfiguracije

Kada se pritisne gumb za dodavanje čvora otvara se konfiguracijski prozor kao što je prikazano na slici 4.7. U konfiguracijskom prostoru potrebno je unijeti unikatno ime čvora i adresu te korisničko ime za SSH provjeru identiteta. Također je potrebno unijeti lozinku ili putanju do ključa koji će se koristiti za SSH protokol. Nakon potvrde dodavanja u konfiguracijsku datoteku se upisuju potrebne informacije. Aplikacija također ima i ugrađenu provjeru kako se ne bi dodali čvorovi s istim imenima ili adresama.



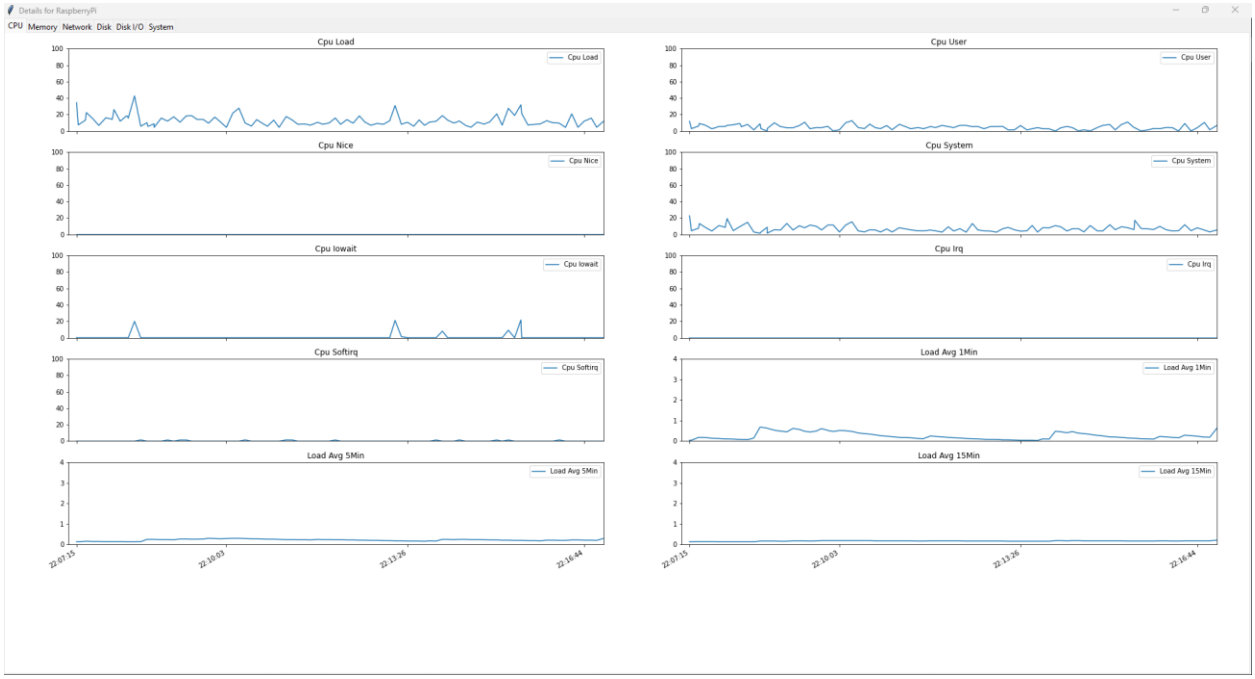
Slika 4.7. Dodavanje čvora

Nakon dodavanja čvora na glavnom zaslonu pojavljuje se zelena kartica ukoliko je čvor uspješno dodan i SSH konekcija je uspostavljena. Slika 4.8. prikazuje uspješnu konekciju čvora te trenutne mjerene veličine. Također u gornjem dijelu korisničkog sučelja nalaze se kumulativne mjerene veličine za cijeli računalni klaster koje se periodično ažuriraju.



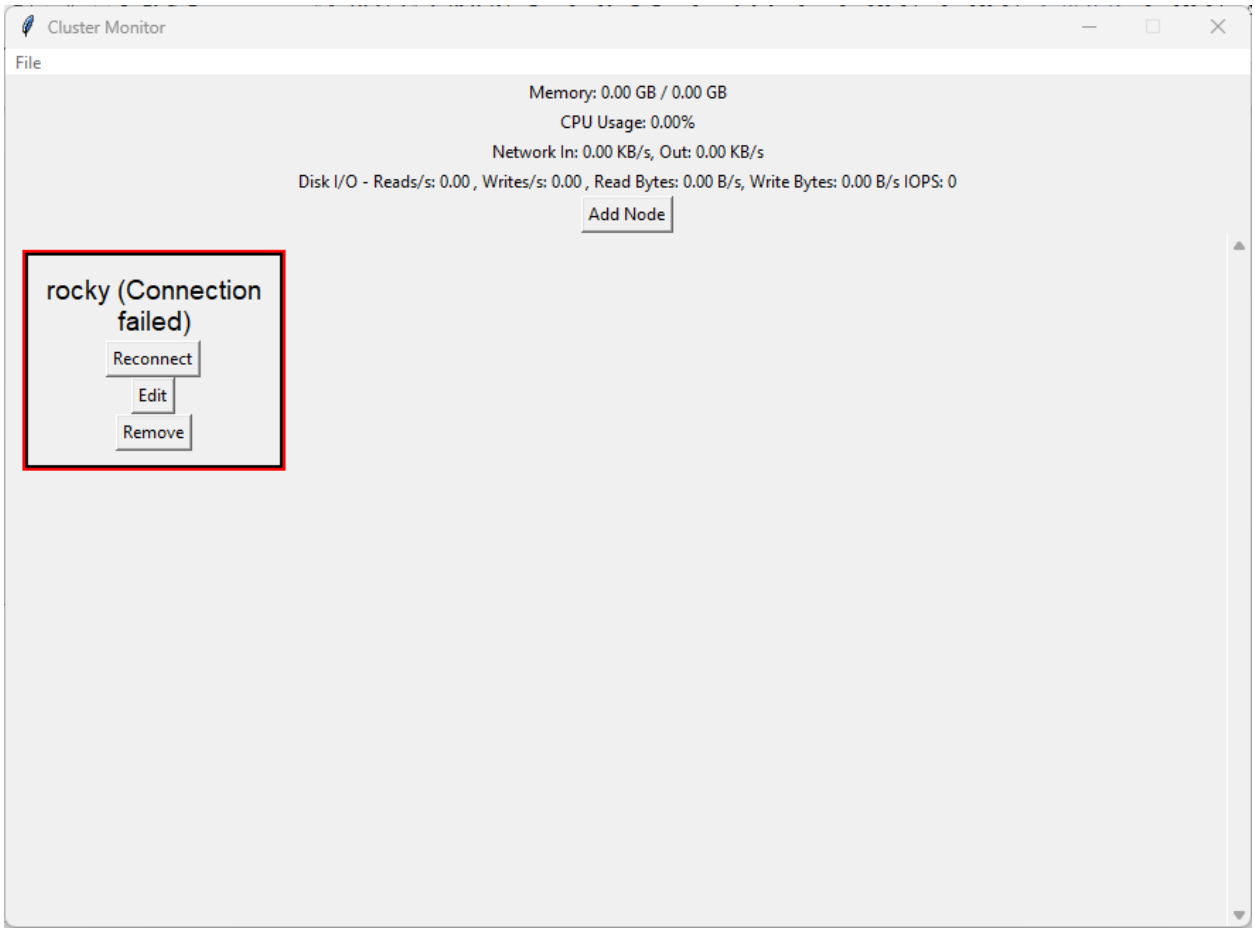
Slika 4.8. Uspješno spojen čvor

Za svaki novi dodani čvor pojavljuje se nova kartica. Kartice je moguće uređivati i uklanjati. Uređivanje se vrši pritiskom gumba *Edit* čime se otvara identičan prozor kao kod dodavanja čvora. Sve promijenjene informacije spremaju se u konfiguracijsku datoteku. Čvor se uklanja pritiskom gumba *Remove* gdje se uklanja kartica čvora te se iz konfiguracijske datoteke brišu informacije za uklonjeni čvor. Pritiskom gumba *More info* otvara se novi prozor prikazan slikom 4.9. U novootvorenom prozoru prikazani su grafovi prikupljenih mjerenih veličina te je moguće pratiti promjene u stvarnom vremenu i vidjeti prijašnje vrijednosti mjerenih veličina. Na slici su prikazane mjerene veličine procesora.



4.9. Grafovi prikupljenih mjerenih veličina

Ukoliko je SSH povezivanje neuspješno umjesto zelene kartice pojavljuje se crvena kartica kao što je prikazano na slici 4.10. Neuspješno spojeni čvor na svojoj kartici ima tri gumba. Gumbovi *Edit* i *Remove* rade isto kao i na normalnoj kartici, dok gumb *Reconnect* pokušava ponovno uspostaviti SSH konekciju te ukoliko je uspješno pojavljuje se normalna zelena kartica.



Slika 4.10. Neuspješno spojen čvor

5. VREDNOVANJE PROGRAMSKOG RJEŠENJA

Vrednovanje programskog rješenja ključno je za procjenu njegove učinkovitosti i utjecaja na performanse računalnog klastera. Svaki program koji se izvodi na računalnom klasteru troši resurse poput procesora, memorije i mreže, što može smanjiti ukupne performanse sustava. Ovaj dodatni trošak resursa, poznat kao *overhead*, može značajno utjecati na HPC sustave, gdje su resursi optimizirani za maksimalnu računalnu snagu. Cilj vrednovanja programskog rješenja izrađenog u okviru ovog rada je izmjeriti koliki *overhead* rješenje uvodi na računalni klaster i koliki je utjecaj na performanse sustava kada je aktivno. Mjerenje *overheada* se postiže usporedbom rezultata HPL (eng. *High-Performance Linpack*, HPL) mjerila. Mjerilo HPL prvo se izvodi na sustavu bez pokretanja programskog rješenja kako bi se dobila referentna vrijednost performansi računalnog klastera. Nakon toga, HPL se pokreće dok programsko rješenje radi, s različitim intervalima prikupljanja podataka (5, 10 i 15 sekundi). Mjerilo HPL je odabrano kao alat za vrednovanje jer je široko korišteno u industriji i istraživanjima za mjerenje performansi HPC sustava te omogućuje preciznu procjenu računalne snage.

5.1. HIGH PERFORMANCE LINPACK BENCHMARK (HPL)

Mjerilo HPL predstavlja standardno mjerilo za procjenu performansi računalnih klastera i superračunala. Prema [14], HPL je specijaliziran za mjerenje sposobnosti sustava u rješavanju sustava linearnih jednadžbi velikih razmjera pomoću metoda faktorizacije matrice, konkretno LU faktorizacije. Glavna mjerena veličina koja se koristi za ocjenu performansi je broj operacija s pomičnim zarezom po sekundi, odnosno FLOPS (eng. *Floating Point Operations Per Second*, FLOPS). HPL je široko prepoznat kao standardni alat za rangiranje najbržih svjetskih superračunala na TOP500 listi, što ga čini relevantnim za mjerenje utjecaja programskog rješenja.

Prema [14], za pokretanje HPL mjerila potrebno je podesiti HPL.dat datoteku koja sadrži parametre potrebne za postavljanje testa. Neki od parametara koje datoteka definira su informacije o veličini problema, broj procesora i način raspodjele podataka te značajke algoritma koje će koristi izvršna datoteka. Slika 5.1. prikazuje primjer HPL.dat datoteke u kojoj je navedeno da će se izvesti jedan test s veličinom problema 13440, što je parametar *N*, koristit će se jedna veličina bloka od 64, parametar *NB*, te da će se test izvoditi na jednoj mreži procesora koja je definirana parametrima *P* i *Q*, odnosno 2 procesa u smjeru retka te 2 u smjeru stupca, koji se mapiraju na četiri procesorske jezgre.

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
13440        Ns
1            # of NBs
64           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
2            Qs
16.0         threshold
1            # of panel fact
1            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
64           NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg, 1=1rM, 2=2rg, 3=2rM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Slika 5.1. Primjer HPL.dat datoteke

Prema [14], izvršavanje testa uključuje sljedeće korake:

1. Generiranje slučajne matrice prema definiranim parametrima.
2. Faktorizacija matrice na blokove pomoću LU faktorizacije.
3. Rješavanje sustava linearnih jednadžbi.
4. Provjera točnosti rješenja i usporedba s očekivanim ispravnim rezultatima.

Slika 5.2. prikazuje rezultate izvršavanja HPL mjerila prema navedenoj HPL.dat datoteci. Na slici su vidljivi parametri dobiveni iz HPL.dat datoteke te sami koraci testa. Nakon toga ispisan je dobiveni rezultat u GFLOPS zajedno s vremenom trajanja testa. U ovom konkretnom primjeru dobiveni rezultat je 1.034 GFLOPS. Na kraju se ispisuje provjera točnosti rješenja.

```
================================================================================
HPLinpack 2.3  --  High-Performance Linpack benchmark  --   December 2, 2018
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   13440
NB     :      64
PMAP   : Row-major process mapping
P      :       2
Q      :       2
PFACT  :    Left    Crout    Right
NBMIN  :       2        4
NDIV   :       2
RFACT  :    Left    Crout    Right
BCAST  :   1ring
DEPTH  :       0
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be               1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00L2L2       13440    64     2     2            1564.36             1.0348e+00
HPL_pdgesv() start time Tue Sep 10 10:03:39 2024

HPL_pdgesv() end time   Tue Sep 10 10:29:43 2024



--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   1.18200856e-03 ...... PASSED
```

Slika 5.2. Primjer rezultata izvršavanja HPL testa

5.2. Analiza rezultata mjerila

Kako bi se detaljno razumio utjecaj programskog rješenja na performanse računalnog klastera, potrebno je usporediti rezultate mjerila bez pokrenutog programskog rješenja s rezultatima dobivenim kada je programsko rješenje aktivno. Cilj ove analize je izmjeriti utjecaj koji programsko rješenje ima na rezultate mjerila. Inicijalno vrednovanje programskog rješenja provedeno je na računalnom klasteru sastavljenom od četiri Raspberry Pi 1B jednopločnih računala (eng. *Single Board Computer*, SBC), koji predstavlja konfiguraciju s ograničenim resursima u usporedbi s modernim računalnim klasterima. SBC je kompaktno računalo koje se sastoji od jedne pločice na kojoj su integrirani svi ključni dijelovi potrebni za rad, uključujući procesor, memoriju i ulazno-izlazne priključke. Prema [15], Raspberry Pi 1B sastoji se od procesora s jednom jezgrom i 512 MB memorije.

Tablica 5.1. prikazuje rezultate vrednovanja programskog rješenja na Raspberry Pi računalnom klasteru. U početnom testu, bez pokrenutog programskog rješenja, računalni klaster je postigao rezultat od 1.034 GFLOPS. Ovaj rezultat predstavlja maksimalne performanse sustava, gdje sva dostupna računalna snaga ide na izvršavanje HPL mjerila bez dodatnih opterećenja. U testovima s pokrenutom aplikacijom za praćenje opterećenja, dolazi do pada performansi na približno 0.78 GFLOPS, što je smanjenje od otprilike 27 % u odnosu na početni test.

Tablica 5.1. Rezultati vrednovanja programskog rješenja na Raspberry Pi računalnom klasteru

Interval prikupljanja podataka (s)	Rezultat mjerila (GFLOPS)	Postotak smanjenja (%)
Bez programskog rješenja	1.034	0
5	0.7873	27.09
10	0.7727	28.92
15	0.78288	27.643

Kod slabijih sustava, poput Raspberry Pi 1B, procesorski kapacitet i memorijski resursi su značajno ograničeni. Svako dodatno opterećenje na procesor, kao što je prikupljanje i obrada podataka, može imati relativno velik utjecaj na performanse jer sustav ima manju rezervu resursa. Kako bi se dodatno ispitao i potvrdio utjecaj programskog rješenja na performanse, testovi su ponovljeni na virtualnom stroju koji ima procesor s četiri jezgre i 2 GB memorije. Svaki test pokrenut je pet puta, a konačni rezultat predstavlja prosjek dobivenih vrijednosti. Inicijalni test

proveden je s intervalima prikupljanja podataka od 5, 10 i 15 sekundi, dok su u novim testovima korišteni veći razmaci, i to 5, 15, 30, 60, 120 i 300 sekundi. U tablici 5.2 prikazani su rezultati mjerenja na virtualnom stroju. Vidljivo je da su početni rezultati bez pokrenutog programskog rješenja veći nego rezultati Raspberry Pi računalnog klastera. U intervalu od 5 do 30 sekundi programsko rješenje ima značajan utjecaj na rezultat testa, iznosivši od 17 % do 25 %. Rezultati za ove intervale pokazuju da, iako virtualni stroj ima značajno veću procesorsku snagu od Raspberry Pi 1B računalnog klastera, aplikacija i dalje uzrokuje sličan relativan pad performansi. Intervali prikupljanja od 60 do 300 sekundi pokazuju manji utjecaj na performanse u vrijednosti između 6 % i 7 %. Te vrijednosti su očekivanije zbog konstantno otvorene SSH veze i prikupljanja mjerenih veličina.

Tablica 5.2. Rezultati vrednovanja programskog rješenja na virtualnom stroju

Interval prikupljanja podataka (s)	Rezultat mjerila (GFLOPS)	Postotak smanjenja (%)
Bez programskog rješenja	3.5256	0
5	2.761	24.32
15	2.744	24.93
30	2.9558	17.58
60	3.273	7.43
120	3.314	6.18
300	3.32	6

Novi rezultati jasno pokazuju da interval prikupljanja podataka ima izravan utjecaj na performanse sustava. Kraći intervali prikupljanja, poput 5 i 15 sekundi, uzrokuju značajan pad performansi oko 25 %, dok duži intervali, poput 60, 120 i 300 sekundi, uzrokuju pad performansi od otprilike 7 %. To je posljedica dodatnog opterećenja koje programsko rješenje stvara na sustavu, osobito kada je procesor već maksimalno opterećen. HPL test maksimalno opterećuje procesor jer koristi sve dostupne resurse, uključujući sve jezgre procesora, kako bi izvodio zahtjevne linearne izračune. U takvim uvjetima, procesor je konstantno zauzet, a svaki dodatni zadatak, poput prikupljanja podataka, uzrokuje preusmjeravanje resursa. Kada se podaci prikupljaju svakih 5 ili 15 sekundi, programsko rješenje mora često pristupati sustavu, što uključuje pokretanje dodatnih ulazno-izlaznih operacija i slanje podataka preko mreže, čime se

stvara dodatni promet i opterećenje na procesor, memoriju i mrežu, odnosno dolazi do zagušenja sustava. S obzirom na to da HPL test dovodi do opterećenja procesora od 100 %, procesor nema viška resursa za rukovanje dodatnim operacijama poput prikupljanja i prijenosa podataka. Svaki put kada se podaci prikupljaju, procesor mora privremeno prekinuti izvođenje glavnog zadatka kako bi obradio zahtjev programskog rješenja. Ovi prekidi uzrokuju zastoj u izvršavanju izračuna, što znači da procesor dio vremena troši na rukovanje prikupljanja podataka. To posebno dolazi do izražaja kod kratkih intervala prikupljanja, kada procesor mora vrlo često prekidati izračune, što dovodi do kumulativnog gubitka performansi.

S povećanjem intervala prikupljanja podataka, *overhead* se značajno smanjuje jer procesor i ostali resursi imaju više vremena za dovršavanje izračuna prije nego što ponovno moraju obaviti prikupljanje podataka. Duži intervali prikupljanja, poput 60 sekundi ili više, omogućuju sustavu da koristi svoje resurse učinkovitije, jer programsko rješenje rjeđe šalje zahtjeve, smanjujući broj ulazno-izlaznih operacija i mrežnih zahtjeva. Na ovaj način, procesor može duže vremena neprekidno obavljati glavne zadatke, što rezultira manjim padom performansi.

Kako bi se smanjio *overhead* i negativni utjecaj na performanse sustava uzrokovan čestim prikupljanjem podataka, postoji nekoliko pristupa koji se mogu implementirati. Jedan od načina je prelazak na prikladniji programski jezik poput C-a ili C++-a, koji nudi bolju kontrolu nad memorijom i paralelnim operacijama, čime se smanjuje opterećenje procesora i ubrzava obrada podataka. Osim toga, korištenje učinkovitijih komunikacijskih metoda poput Apache Kafke moglo bi smanjiti opterećenje mreže i omogućiti brže prikupljanje i prijenos podataka. Kafka, koja je poznata po svojoj skalabilnosti i sposobnosti obrade velikih količina podataka u stvarnom vremenu, bila bi izvrsno rješenje za distribuirano prikupljanje podataka s više čvorova, jer bi omogućila asinkrono slanje podataka bez prevelikog opterećenja mreže i procesora. Kombiniranjem tih metoda s dinamičkim intervalima prikupljanja, gdje se intervali prilagođavaju ovisno o opterećenju sustava, mogli bi se znatno smanjiti negativni utjecaji na performanse, omogućujući optimalno korištenje resursa čak i tijekom zahtjevnih zadataka poput HPL testa. Ovim pristupom postigla bi se bolja ravnoteža između detaljnog praćenja u stvarnom vremenu i učinkovitosti sustava. S obzirom na dobivene rezultate, programsko rješenje prikladno je za manje računalne klastere, poput klastera jednopločnih računala ili klastera namijenjenih edukativnim svrhama.

6. ZAKLJUČAK

Računalni klasteri postali su ključni elementi suvremenih informacijskih sustava, posebno u područjima koja zahtijevaju veliku računalnu snagu i visoke performanse, poput znanstvenih istraživanja u području simulacija klime, bioinformatike i analize velikih podataka, te industrijskih aplikacija kao što su obrada podataka u financijskim sustavima, modeliranje u inženjerskim procesima i razvoj umjetne inteligencije. Jedan od glavnih izazova u upravljanju računalnim klasterima je osigurati optimalno korištenje resursa uz održavanje stabilnosti i performansi sustava. Stoga je praćenje programskog opterećenja u stvarnom vremenu od iznimne važnosti, jer omogućuje administratorima da pravovremeno identificiraju potencijalne probleme prije nego što eskaliraju i uzrokuju pad performansi ili nepredviđene kvarove. Osim toga, učinkovito praćenje pomaže optimizirati iskoristivost dostupnih resursa, čime se postiže veća učinkovitost i stabilnost sustava. U ovom radu obrađena je problematika praćenja opterećenja računalnih klastera u stvarnom vremenu, s naglaskom na praćenje ključnih komponenti poput procesora, memorije, diska, mreže i ulazno-izlaznih operacija diska.

S obzirom na postojanje brojnih alata koji nude slična rješenja, kao što su Ganglia, Prometheus, Zabbix, Netdata i Nagios, u ovom radu istražene su njihove značajke i prednosti, uz identifikaciju njihovih ograničenja. Iako su ovi alati vrlo funkcionalni, većina njih zahtijeva instalaciju agenata na svaki čvor u računalnom klasteru, što može povećati administrativne zadatke i kompleksnost sustava. Stoga je razvijeno programsko rješenje koje koristi SSH protokol za prikupljanje mjerenih veličina s čvorova računalnih klastera, čime se eliminira potreba za dodatnim agentima. Ovo rješenje pruža jednostavniji pristup nadzoru čvorova i smanjuje administrativne zahtjeve, što ga čini prikladnim za manje i jednostavnije računalne klastere, poput klastera jednopločnih računala, hobi klastera ili klastera namijenjenih edukativnim svrhama. U takvim okruženjima obično nema sistemskih administratora koji bi održavali složenije alate i instalirali agente na svaki čvor, pa bi primjena takvih rješenja bila pretjerano zahtjevna i neisplativa. Programsko rješenje periodično prikuplja i vizualizira mjerene veličine procesora, memorije, diska, mreže i ulazno-izlaznih operacija diska, omogućujući administratorima brz uvid u stanje sustava i potencijalne probleme.

Iako izrađeno programsko rješenje uspješno prati opterećenje računalnog klastera u stvarnom vremenu, testovi su pokazali da programsko rješenje uvodi *overhead*, posebno kod kraćih intervala prikupljanja podataka. U intervalima prikupljanja od 5 do 30 sekundi vrijednost *overhead* je značajan i iznosi između 15 % i 25 %. Kod većih intervala prikupljanja, poput 60 sekundi i naviše, vrijednost *overheda* iznosi oko 7 %, što je i očekivano za programsko rješenje. Korištenje

učinkovitijih metoda za prijenos podataka, poput Apache Kafke, ili čak prelazak na učinkovitiji programski jezik mogli bi značajno smanjiti ovaj utjecaj. Iako programsko rješenje već sadrži brojne funkcionalnosti, postoji prostor za daljnje nadogradnje. Jedna od ključnih nadogradnji je lokalno spremanje povijesti prikupljenih podataka, što bi omogućilo kasniju analizu bez dodatnog opterećenja računalnog klastera. Također, dodavanje mogućnosti konfiguracije parametara unutar samog rješenja, poput podešavanja intervala prikupljanja podataka i odabira specifičnih mjerenih veličina, dodatno bi povećalo fleksibilnost i korisnost aplikacije. Pored toga dodavanje dinamičkih intervala prikupljanja moglo bi pozitivno utjecati na *overhead* koji programsko rješenje uvodi u sustav. Uz ove nadogradnje i smanjenje *overheada*, sveukupna učinkovitost i funkcionalnost rješenja bila bi značajno poboljšana.

LITERATURA

[1] Ganglia [online], Github, dostupno na: https://github.com/ganglia [16.6.2024.]

[2] simplegeo: Ganglia [online], Github, dostupno na: https://github.com/simplegeo/ganglia/blob/master/README [16.6.2024.]

[3] Prometheus dokumentacija [online], The Linux Foundation, 2024., dostupno na: https://prometheus.io/docs [16.6.2024]

[4] Zabbix dokumentacija [online], Zabbix SIA., 2024., dostupno na: https://www.zabbix.com/documentation/current/en/manual/ [16.6.2024.]

[5] Netdata repozitorij [online], Github, dostupno na: https://github.com/netdata/netdata [16.6.2024.]

[6] Netdata dokumentacija [online], Netdata, Inc., 2024., dostupno na: https://learn.netdata.cloud/docs/ [16.6.2024.]

[7] Nagios dokumentacija [online], Nagios Enterprises, LLC., dostupno na: https://assets.nagios.com/downloads/nagioscore/docs/ [16.6.2024.]

[8] Cloudflare, „What is SSH?“ [online], Cloudflare, 2024., dostupno na: https://www.cloudflare.com/learning/access-management/what-is-ssh/ [18.6.2024.]

[9] Ellingwood J., „Understanding the SSH Encryption and Connection Process“ [online], DigitalOcean, 2022., dostupno na: https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process [11.9.2024.]

[10] Python Software Foundation, „The Python Standard Library“ [online], Python Software Foundation, 2024., dostupno na: https://docs.python.org/3/library [18.6.2024.]

[11] Frederick R., „AsyncSSH: Asynchronous SSH for Python“ [online], 2023., dostupno na: https://asyncssh.readthedocs.io/en/latest/ [18.6.2024.]

[12] Hunter J., Darren D., Firing E., Droettboom M., Matplotlib [online], The Matplotlib development team, 2024., dostupno na: https://matplotlib.org/stable/users/explain/quick_start.html [18.6.2024.]

[13] Kerrisk M., „Linux manual page“ [online], 2024., dostupno na: https://man7.org/linux/man-pages/man1 [18.6.2024.]

[14] Petitet A., Whaley R. C., Dongarra J., Cleary A., „ HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers“ [online], University of Tennessee, 2018., dostupno na: https://www.netlib.org/benchmark/hpl/index.html [12.9.2024.]

[15] Raspberry Pi Foundation, „Raspberry Pi Resources“ [online], IBEX UK Ltd., dostupno na: https://raspberry-projects.com/pi/category/pi-hardware/raspberry-pi-model-b [12.9.2024.]

SAŽETAK

U računalnim klasterima, pravovremeno praćenje programskog opterećenja ključno je za održavanje stabilnosti i optimizaciju resursa, no postojeća rješenja često zahtijevaju kompleksne postavke i dodatne agente. Ovaj rad bavi se razvojem aplikacije za praćenje programskog opterećenja računalnih klastera u stvarnom vremenu, što je ključan zadatak u održavanju optimiziranosti i performansi računalnih klastera. U radu su istražena rješenja poput Ganglia, Prometheus i Zabbix, a identificirani su njihovi prednosti i nedostaci, kao što je potreba za dodatnim agentima. Izrađeno programsko rješenje omogućuje nadzor ključnih resursa računalnih klastera, uključujući procesore, memoriju, diskove i mrežu, bez potrebe za instaliranjem dodatnih agenata na pojedinačne čvorove. Korištenjem SSH protokola, aplikacija prikuplja podatke u stvarnom vremenu, dok vizualizacija prikupljenih podataka omogućava administratorima brz uvid u stanje sustava. Iako aplikacija uvodi značajan *overhead* ovisno o intervalu prikupljanja podataka, njegova jednostavnost implementacije i preciznost podataka čini ga prikladnim za manje i jednostavnije računalne klastere, poput klastera jednopločnih računala ili klastera namijenjenih edukativnim svrhama. Rješenje je dizajnirano za jednostavne nadogradnje i optimizacije, osiguravajući dugoročnu skalabilnost i prilagodljivost infrastrukturi klastera.

Ključne riječi: nadzor računalnih resursa, praćenje opterećenja, računalni klasteri, SSH protokol, vizualizacija podataka

ABSTRACT

Application for Real-time Monitoring of Computing Cluster Workload

Real-time monitoring of computational load is essential for maintaining stability and optimizing resources in computer clusters. However, existing solutions often require complex configurations and additional agents. This thesis focuses on the development of a real-time monitoring application for computer cluster loads, a critical task for maintaining stability and performance. The study analyzes current solutions, such as Ganglia, Prometheus, and Zabbix, highlighting their advantages and drawbacks, like the need for agent installation. The developed software solution allows for the monitoring of key computer cluster resources, including processors, memory, disks, and network, without the need for installing additional agents on individual nodes. By utilizing the SSH protocol, the application collects real-time data, providing system administrators with a clear overview of system status through visualizations. Although the solution introduces some overhead depending on the data collection interval, its ease of implementation and data accuracy make it well-suited for smaller and simpler clusters, such as single-board computer clusters or educational environments. The system is designed to support future upgrades and optimizations, ensuring long-term scalability and adaptability to changes in computer cluster infrastructure.

Keywords: computer clusters, data visualization, load monitoring, resource monitoring, SSH protocol

ŽIVOTOPIS

Tomislav Barbarić rođen je u Slavonskom Brodu 11.4.2001. godine gdje je završio osnovnoškolsko obrazovanje i srednju tehničku školu, smjer Tehničar za elektroniku. Na Erasmus mobilnosti u Španjolskoj uči osnove razvoja Android aplikacija. Nakon srednje škole upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku gdje dalje unapređuje svoje znanje o računarstvu i elektrotehnici. Nakon preddiplomskog studija, 2022. upisuje diplomski studij Informacijske i podatkovne znanosti na Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku gdje nastavlja svoje obrazovanje.

PRILOZI

Prilog 1: Izvorni kod programskog rješenja s uputama za pokretanje na *Githubu*: https://github.com/tomislav4545/computer_cluster_monitoring_software