Puštanje u pogon aplikacije CosmoLLM s podrškom za distribuirano računanje i višekorisničku komunikaciju

Sočec, Nikola

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1595

PUŠTANJE U POGON APLIKACIJE COSMOLLM S PODRŠKOM ZA DISTRIBUIRANO RAČUNANJE I VIŠEKORISNIČKU KOMUNIKACIJU

Nikola Sočec

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1595

PUŠTANJE U POGON APLIKACIJE COSMOLLM S PODRŠKOM ZA DISTRIBUIRANO RAČUNANJE I VIŠEKORISNIČKU KOMUNIKACIJU

Nikola Sočec

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1595

Pristupnik:	**Nikola Sočec (0036531697)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Mario Brčić

Zadatak: **Puštanje u pogon aplikacije CosmoLLM s podrškom za distribuirano računanje i višekorisničku komunikaciju**

Opis zadatka:

Cilj ovog završnog rada je razvoj sustava za puštanje u pogon aplikacije CosmoLLM, koja omogućava istovremenu komunikaciju s više korisnika i efikasno izvršavanje Monte Carlo Markovljevih lanaca (engl. Monte Carlo Markov Chain, MCMC) eksperimenata koristeći distribuirano računanje. Sustav će se oslanjati na infrastrukturu sastavljenu od čvorova unutar Kubernetes okruženja, gdje će svaki čvor imati specifičnu ulogu. Te uloge uključuju omogućavanje komunikacije korisnika s agentom CosmoLLM preko web sučelja, izvršavanje distribuiranih MCMC izračuna, te upravljanje procesom stvaranja novih čvorova. Rad će istražiti kako se Kubernetes može iskoristiti za izgradnju robusnog sustava koji podržava kompleksne računske zadatke i višekorisničku interakciju s CosmoLLM agentom, te kako se kroz web sučelje može omogućiti intuitivna komunikacija s korisnicima.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

1. Uvod

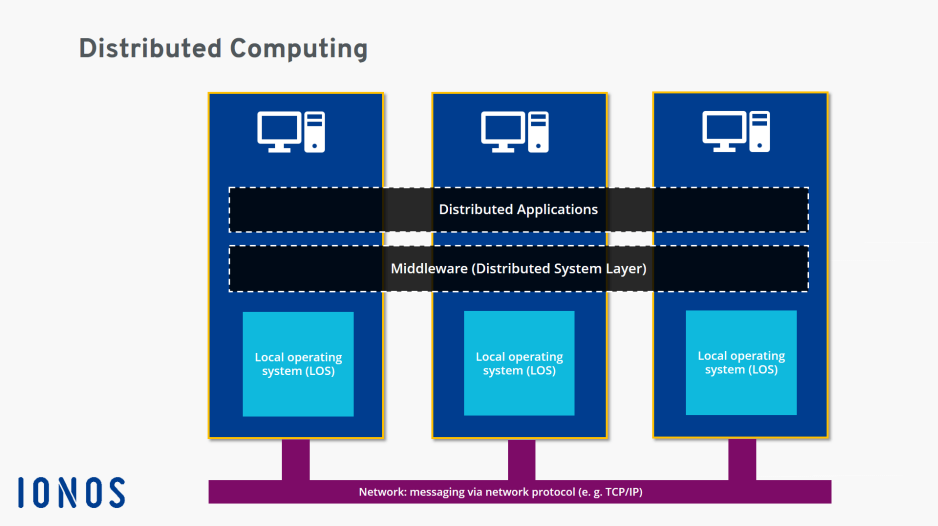
Puštanje u pogon modernih aplikacija, posebice onih s web sučeljima kojima mora pristupati veliki broj korisnika, zahtijeva obraćanje pažnje na skalabilnost sustava vrlo rano u razvoju. Ova potreba postaje još izraženija kada je u rad aplikacije uključeno i računanje koje zahtijeva netrivijalne količine resursa. Baš takav je slučaj s aplikacijom CosmoLLM, u kojoj svaki korisnik razgovara s velikim jezičnim modelom (LLM) te provodi izračune bazirane na metodi Monte Carlo Markovljevih lanaca (engl. Markov Chain Monte Carlo, MCMC). Takvi izračuni zahtijevaju značajne količine resursa i mogu trajati vrlo dugo, ovisno o raznim parametrima koje jezični model u komunikaciji s korisnikom izabere. Zbog toga je za rješenje ovog problema izabrano korištenje distribuiranog računanja i orkestracije. Distribuirano računanje je implementirano koristeći protokol OpenMPI, dok je orkestracija ostvarena korištenjem Kubernetesa.

Cilj ovog završnog rada je implementirati sustav koji će omogućiti pokretanje aplikacije CosmoLLM u Kubernetes okruženju s podrškom za distribuirano računanje. Konkretno, to uključuje tri komponente aplikacije koje ćemo nazvati čvorovima. Ti čvorovi obuhvaćaju čvor za korisničko sučelje, čvor za izračune i čvor za administraciju ostalih čvorova (sa korisničkim sučeljem za administratore).

Rad je strukturiran na sljedeći način. U poglavlju 2., pod nazivom "Distribuirano računanje", objašnjeni su osnovni koncepti distribuiranog računanja i OpenMPI protokola. Također je predstavljen algoritam Monte Carlo Markovljevih lanaca (MCMC) koji se koristi za izračune unutar aplikacije CosmoLLM. U poglavlju 3., "Orkestracija," detaljno je objašnjen Kubernetes, alat koji omogućuje orkestraciju distribuiranih sustava. Posebna pažnja posvećena je integraciji OpenMPI-ja s Kubernetesom. Poglavlje 4., "Definicija problema", precizno definira problem koji se rješava ovim radom, uključujući zahtjeve sustava i izazove s kojima se susrećemo. Poglavlje 5., "Rješenje", opisuje imple-

mentaciju rješenja, uključujući arhitekturu sustava i tehničke detalje o svakoj komponenti. U poglavlju 6., "Analiza skalabilnosti i performansi sustava", prikazani su rezultati izvedbe sustava nakon implementacije, s analizom učinkovitosti i skalabilnosti. Poglavlje 7., "Upute za korištenje", daje korisnicima detaljne upute o korištenju implementiranih alata kako bi mogli pustiti u pogon vlastitu instancu sustava. Na kraju, poglavlje 8., "Zaključak", daje pregled postignutih ciljeva, te prijedloge za budući rad i moguća poboljšanja sustava.

2. Distribuirano računanje



Slika 2.1. Distribuirano računanje, izvor: `https://www.ionos.ca/digitalguide/server/know-how/what-is-distributed-computing/`

Distribuirano računanje predstavlja način obrade podataka i izvršavanja zadataka korištenjem više povezanih računala koja zajednički rade kao jedan sustav. Ovaj pristup omogućuje raspodjelu zadataka na više čvorova, čime se postiže povećanje efektivne računalne snage, poboljšanje skalabilnosti te smanjenje vremena obrade[1].

U distribuiranom sustavu, pojedini zadaci se dijele na manje dijelove, od kojih svaki dio može biti obrađen paralelno na različitim računalima unutar mreže. Ova podjela omogućuje da se zahtjevi za računalnim resursima efikasnije koriste, čime se poboljšava ukupna učinkovitost sustava. Glavna prednost distribuiranog računanja je mogućnost obrade velikih količina podataka i izvršavanja kompleksnih zadataka koji bi na pojedinačnom računalu trajali predugo ili bi zahtijevali previše resursa[1].

Distribuirano računanje se često koristi u različitim područjima, uključujući znanstvena istraživanja, financijske analize i razne simulacije. Primjeri primjene uključuju

analizu genoma, simulacije fizikalnih sustava, financijsko modeliranje, nadziranje naftnih crpki u stvarnom vremenu itd.[1]

Postoji nekoliko modela distribuiranog računanja, od kojih su najčešći "client-server model", "peer-to-peer model", "trorazinski model" i "N-razinski model". Client-server model je ostvaren centraliziranim poslužiteljem koji upravlja zahtjevima klijenata i raspodjeljuje zadatke na raspoložive resurse. Peer-to-peer model, s druge strane, omogućuje svim čvorovima u mreži da ravnopravno dijele zadatke i resurse bez centraliziranog upravljanja. Trorazinski model je client-server model u kojem se poslužitelji dijele na dvije skupine: aplikacijske i podatkovne. Aplikacijski poslužitelji obrađuju zahtjeve klijenata, dok podatkovni poslužitelji upravljaju pohranom i dohvaćanjem podataka. N-razinski model se sastoji od više client-server sustava koji međusobno surađuju[1].

Jedan od ključnih izazova u distribuiranom računanju je osiguravanje pouzdanosti i otpornosti sustava. To uključuje rješavanje problema poput otkaza čvorova, mrežnih problema i sinkronizacije podataka među čvorovima. Kako bi se ti izazovi prevladali, koriste se različite tehnike poput replikacije podataka, detekcije i oporavka od otkaza te koordinacijskih protokola[2].

Osim tehničkih izazova, distribuirano računanje donosi i određene sigurnosne izazove. Zaštita podataka i osiguravanje privatnosti korisnika su od ključne važnosti, posebno u okruženjima gdje više korisnika dijeli resurse. Primjenjuju se različite sigurnosne mjere poput enkripcije, autentifikacije i autorizacije kako bi se osigurala sigurnost distribuiranih sustava[2].

U kontekstu aplikacije CosmoLLM, distribuirano računanje omogućuje raspodjelu zahtjevnih MCMC izračuna na više čvorova, čime se postiže brže i efikasnije izvršavanje zadataka. Implementacija distribuiranog računanja u ovoj aplikaciji koristi OpenMPI protokol za međusobnu komunikaciju čvorova, čime se osigurava skalabilnost i pouzdanost sustava.

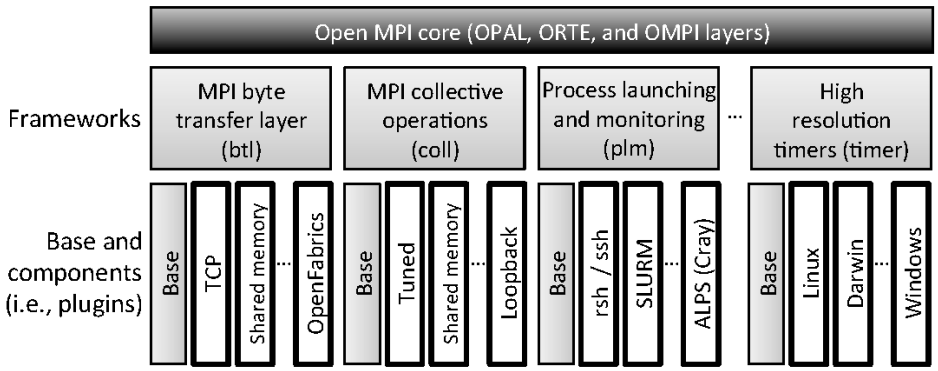
2.1. OpenMPI

OpenMPI (engl. Open Message Passing Interface) je programska biblioteka koja pruža podršku za paralelno računanje koristeći standardizirani model poruka za komunikaciju

Slika 2.2. OpenMPI logo, izvor: `https://avatars.githubusercontent.com/u/2165682?s=280&v=4`

među procesima. Razvijen je s ciljem olakšavanja razmjene podataka među procesima koji se izvršavaju na različitim čvorovima u računalnom clusteru, kao i na višejezgrenim sustavima. OpenMPI je rezultat kolaboracije nekoliko akademskih i industrijskih institucija, što uključuje zajedničke napore istraživača i programera iz različitih područja znanosti i tehnologije[3].

OpenMPI implementira specifikaciju MPI (engl. Message Passing Interface), koja je de facto standard za poruke u distribuiranom računalstvu. MPI standard definira kako se podaci prenose među procesima, osiguravajući konzistentnost i učinkovitost komunikacije. U kontekstu OpenMPI-ja, komunikacija se ostvaruje putem različitih metoda, kao što su point-to-point komunikacija i kolektivne operacije, koje omogućuju razmjenu podataka među većim brojem procesa istovremeno[3].



Slika 2.3. Modularnost OpenMPI arhitekture, izvor: `https://aosabook.org/static/openmpi/open-mpi-mca.png`

Jedna od ključnih prednosti OpenMPI-ja je njegova modularna arhitektura, koja omogućuje jednostavno proširenje i prilagodbu. Ova modularnost omogućuje korisnicima da biraju između različitih mrežnih protokola i komunikacijskih sučelja, te da optimi-

raju performanse sustava prema specifičnim zahtjevima aplikacije. OpenMPI podržava različite mrežne protokole, uključujući InfiniBand, Ethernet i Myrinet, što ga čini vrlo fleksibilnim i prilagodljivim za različite računalne infrastrukture[3].

Osim toga, OpenMPI uključuje napredne mehanizme za uravnoteženje opterećenja (engl. load balancing) i upravljanje resursima, što omogućuje efikasniju distribuciju zadataka među procesima i bolju iskorištenost računalnih resursa. Također, podržava različite razine paralelizacije, od osnovne do napredne, čime se omogućuje skalabilnost aplikacija od malih do velikih sustava[3].

Razvoj OpenMPI-ja prati aktivna zajednica korisnika i programera, što osigurava kontinuirano unapređenje i ažuriranje ove programske biblioteke. Dokumentacija i podrška dostupni su putem brojnih online resursa, uključujući službene web stranice, forume te znanstvene publikacije, što korisnicima omogućuje lakše usvajanje i primjenu OpenMPI-ja u njihovim projektima.

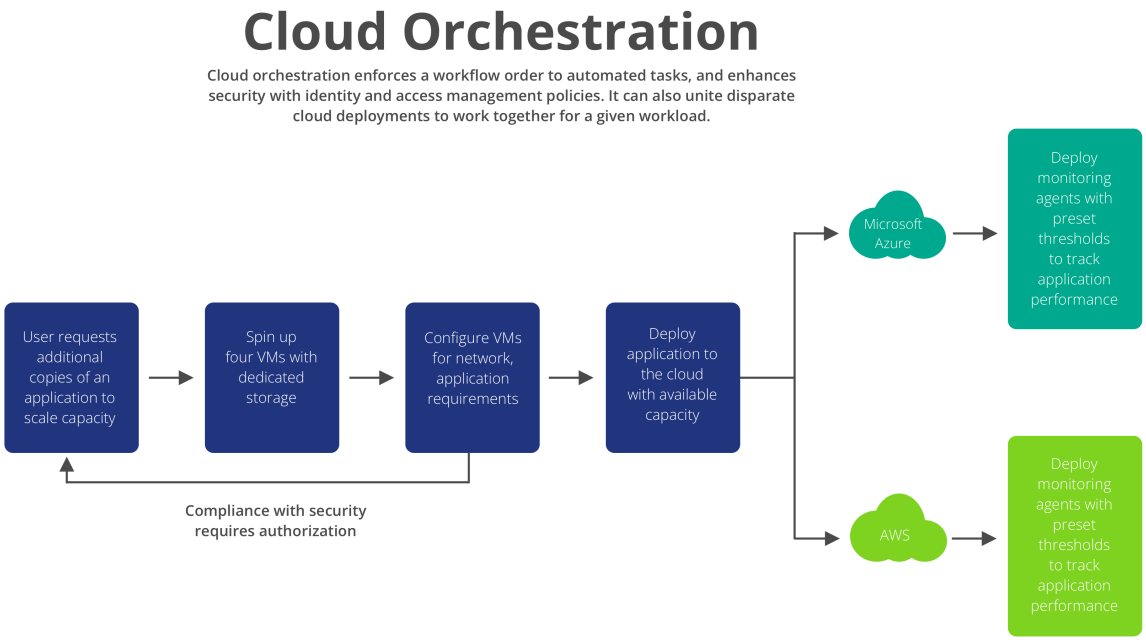
2.2. Monte Carlo Markovljevi lanci (MCMC)

Aplikacija CosmoLLM koristi Python biblioteku “zeus-mcmc“, koja implementira algoritam Monte Carlo Markovljevih lanaca, specifično “Ensemble Slice Sampling (ESS)“ metodu[4].

MCMC metoda je skup algoritama koji koriste uzorkovanje kako bi procijenili distribucije kompleksnih sustava. Njihova primjena je široka, od fizike i kemije do ekonomije i statistike, posebno u slučajevima gdje je izravno izračunavanje distribucija nemoguće zbog visokodimenzionalnosti problema.

Aspekt zeus-mcmc alata koji je posebno važan za ovaj rad je mogućnost paralelizacije izračuna. Naime, zeus-mcmc omogućuje paralelno uzorkovanje distribucija. Svaki proces uzorkovanja se naziva lanac (engl. chain). Više lanaca se može izračunavati istovremeno na različitim čvorovima u paraleli. Ova značajka je ključna za postizanje bržeg i efikasnijeg uzorkovanja distribucija, posebno u slučajevima kada je potreban veliki broj iteracija kako bi se dobila pouzdana procjena. Za ostvarenje paralelnog uzorkovanja, zeus-mcmc koristi OpenMPI protokol.

3. Orkestracija



Slika 3.1. Dijagram orkestracije u oblaku, izvor: `https://pliant.io/wp-content/uploads/2022/10/cloud-orchestration-diagram.png`

Orkestracija je koordinirana izvedba više automatizacijskih zadataka ili procesa u računarstvu. Obično se primjenjuje na više računalnih sustava, aplikacija i usluga kako bi se osiguralo da se implementacija, upravljanje konfiguracijom i drugi procesi izvode u pravilnom slijedu. Iako su automatizacija i orkestracija povezani koncepti, razlikuju se po svojoj svrsi. Automatizacija koristi softver za obavljanje zadataka bez ljudske intervencije, što minimizira pogreške i smanjuje vrijeme potrebno za ručno izvođenje operacija. Orkestracija, s druge strane, koordinira automatizirane zadatke na višim razinama kako bi se postigao specifičan cilj ili proces[5]. U kontekstu puštanja u pogon aplikacije CosmoLLM, orkestracija se koristi za upravljanje distribuiranim sustavom, što uključuje pokretanje, nadzor i skaliranje komponenti (čvorova) aplikacije, pružanje umreženog datotečnog sustava, osiguravanje mrežne komunikacije između čvorova i slično.

U industriji, timovi upravljaju velikim brojem poslužitelja, sustava i aplikacija u privatnim podatkovnim centrima, i sustavima u oblaku. Kako industrijska okruženja postaju složenija, automatizacija zadataka poboljšava učinkovitost i olakšava upravljanje procesima, ali skaliranje automatizacije donosi vlastite izazove. Većina procesa uključuje mnoge pojedinačne zadatke koji trebaju biti automatizirani. Da bi se potpuno automatizirao proces, ti zadaci također moraju međusobno surađivati; kad jedan zadatak završi, treba pokrenuti odgovarajući sljedeći zadatak. Neka automatizacijska rješenja mogu povezati zadatke u logičke cjeline, što uklanja potrebu za ručnim pokretanjem akcija u odgovarajućem trenutku. Izgradnja tih tijekova logičkih cjelina je jedan element orkestracije[5].

Primjer orkestracije u oblaku je proces koji se sastoji od sljedećih koraka:

1. Alokacija virtualnih strojeva (engl. Virtual Machine, VM) u oblaku za izvršavanje aplikacija
2. Instalacija potrebnih programskih paketa na VM-ove
3. Konfiguracija mrežnih postavki i sigurnosnih pravila
4. Pokretanje aplikacija na VM-ovima
5. Nadzor i upravljanje resursima
6. Automatsko skaliranje aplikacija prema promjenama u opterećenju

3.1. Kubernetes

Slika 3.2. Kubernetes logo, izvor: `https://upload.wikimedia.org/wikipedia/commons/thumb/3/39/Kubernetes_logo_without_workmark.svg/2109px-Kubernetes_logo_without_workmark.svg.png`

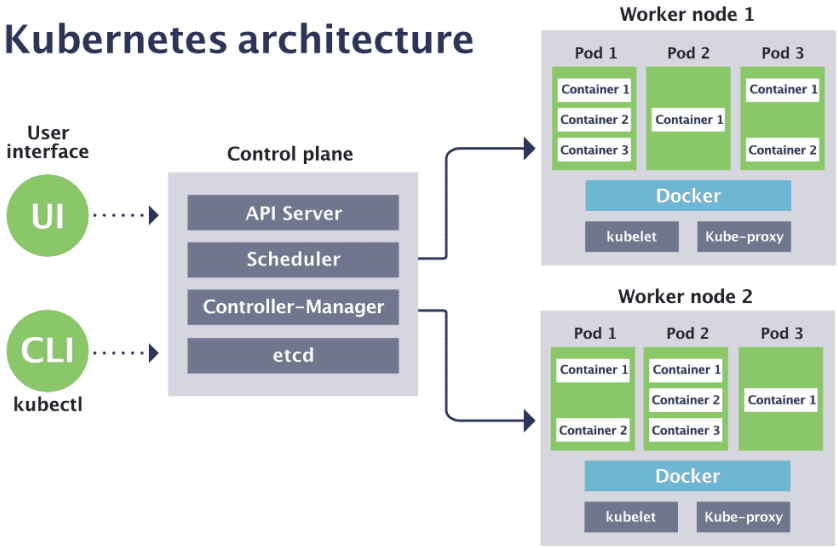
Kubernetes je platforma otvorenog koda za automatizaciju implementacije, skaliranja i upravljanja aplikacijama u kontejnerima, tj. orkestraciju uz pomoć kontejnera. Razvio ju je Google i predstavio javnosti 2014. godine. Ova platforma nudi rješenja za izazove tradicionalnog i virtualiziranog načina implementacije aplikacija, što omogućuje učinkovitije korištenje resursa i skalabilnost sustava.

Tradicionalni način implementacije aplikacija na fizičkim serverima imao je ograničenja u pogledu efikasnosti resursa, gdje su aplikacije često bile prisiljene dijeliti resurse bez mogućnosti izolacije. To je dovodilo do problema s performansama i visokim troškovima održavanja mnogobrojnih servera. Virtualizacija je riješila neke od tih problema omogućujući pokretanje više virtualnih mašina na jednom fizičkom poslužitelju, ali to rješenje nije bilo dovoljno fleksibilno ni učinkovito za sve scenarije[6].

Kontejneri su uvedeni kao lakša alternativa VM-ovima. Omogućuju dijeljenje operacijskog sustava među aplikacijama, čime se poboljšava efikasnost korištenja resursa. Kontejneri uz to omogućuju agilnu kreaciju i implementaciju aplikacija, kontinuirani razvoj, integraciju i isporuku (engl. Continuous Integration / Continuous Deployment, CI/CD), te lakše monitoriranje sustava[6].

3.1.1. Arhitektura Kubernetes clustera

Osnovna jedinica u Kubernetes arhitekturi je “cluster“, koji se sastoji od jednog ili više čvorova (engl. nodes) i jedne komponente “Control Plane“. Control Plane je odgovoran za upravljanje clusterom, što uključuje raspodjelu zadataka te upravljanje resursima,



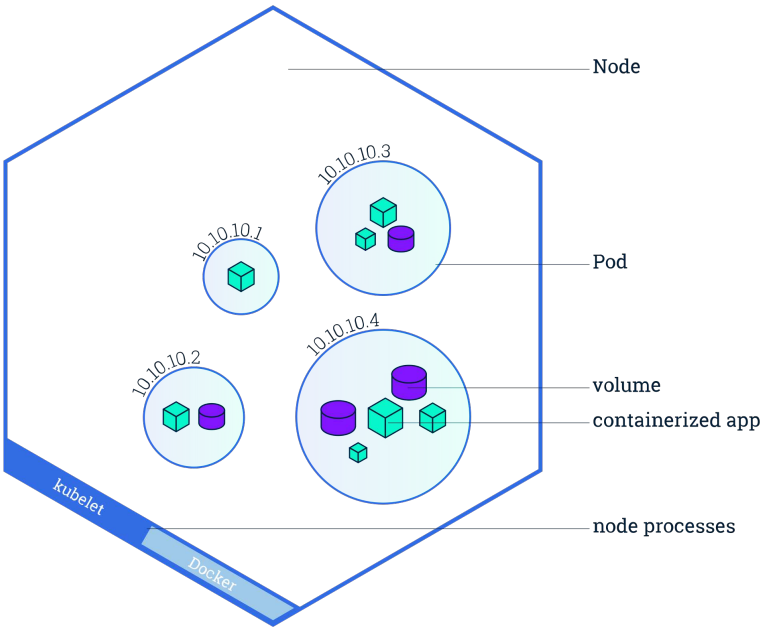
Slika 3.3. Kubernetes arhitektura, izvor: `https://www.cncf.io/wp-content/uploads/2020/08/Kubernetes-architecture-diagram-1-1.png`

mrežom i sigurnošću. Svaki čvor u clusteru je fizički ili virtualni poslužitelj koji izvršava aplikacije u kontejnerima[7]. U kontekstu aplikacije CosmoLLM, elementi aplikacije su također nazvani čvorovima, ali oni su različiti od Kubernetes čvorova. Kubernetes čvorovi su infrastrukturni elementi unutar kojih se pokreću kontejneri sa CosmoLLM čvorovima.

3.1.2. Ključne komponente Kubernetes platforme

Unutar Kubernetes clustera postoje različiti objekti koji omogućuju okrestraciju uz pomoć kontejnera. Najbitniji od tih objekata za puštanje u pogon aplikacije CosmoLLM u Kubernetes okruženju uključuju:

- **Container:** jedinica programske distribucije koja sadrži aplikaciju unutar slike (engl. Docker Image) operacijskog sustava, zajedno s njenim konfiguracijskim datotekama i svim ovisnostima. Kontejneri su izolirani jedni od drugih i dijele operacijski sustav domaćina (Kubernetes čvora)[8].
- **Pod:** najmanja jedinica u Kubernetesu koja sadrži jedan ili više kontejnera. Podovi dijele resurse i mrežu, što omogućuje jednostavno upravljanje grupama kontejnera[9].
- **Deployment:** objekt koji definira željeno stanje aplikacije te upravlja procesom



Slika 3.4. Kubernetes komponente, izvor: `https://kubernetes.io/docs/tutorials/kubernetes-basics/public/images/module_03_nodes.svg`

implementacije i skaliranja. Deploymenti omogućuju deklarativno upravljanje aplikacijama, što znači da korisnici samo trebaju definirati željeno stanje, a Kubernetes će se pobrinuti da se ono ostvari[10].

- **Service:** objekt koji definira pristupnu točku za aplikaciju unutar clustera. Servisi omogućuju komunikaciju između različitih komponenti aplikacije, kao i pristup aplikaciji izvana[11].

- **Ingress:** objekt koji omogućuje pristup aplikaciji izvan clustera. Ingressi omogućuju upravljanje pristupom aplikacijama i SSL certifikatima. Izrađuju se na temelju konkretne domene preko koje će se vanjski klijenti spajati[12].

- **Volume:** objekt koji omogućuje trajno pohranjivanje podataka unutar clustera. Omogućuje dijeljenje podataka između različitih kontejnera i čuvanje podataka između njihovog ponovnog pokretanja[13].

- **ConfigMap:** objekt koji omogućuje definiranje konfiguracijskih podataka unutar clustera. ConfigMapovi omogućuju odvajanje konfiguracijskih podataka od aplikacijskog koda, što olakšava upravljanje konfiguracijom aplikacija. Više različitih

komponenti može čitati konfiguracije iz istog objekta[14].

- **Secret:** objekt donekle sličan ConfigMap objektu koji omogućuje sigurno pohranjivanje osjetljivih podataka unutar clustera. Pružaju podršku za enkripciju i dekripciju podataka, te pristupanje podacima samo autoriziranim korisnicima[15].
- **Namespace:** objekt koji omogućuje grupiranje resursa unutar clustera. Omogućuje organizaciju resursa prema različitim kriterijima, kao što su timovi, okruženja i aplikacije[16].
- **StorageClass:** objekt koji omogućuje definiranje različitih tipova pohrane unutar clustera. Vrste pohrane mogu biti lokalna pohrana, mrežna pohrana itd.[17]

3.2. Integracija s OpenMPI

Kao što je već spomenuto, aplikacija CosmoLLM koristi OpenMPI protokol za distribuirano računanje, a svoje čvorove pokreće unutar Kubernetes clustera. Da bi funkcionirao, OpenMPI zahtijeva da svi čvorovi koji će sudjelovati u izračunima budu povezani s glavnim čvorom (čvor sa grafičkim sučeljem za korisnika) preko SSH protokola. U kontekstu Kubernetesa, to znači da svi čvorovi moraju imati uspostavljen vlastiti servis i imati pristup secret objektu koji sadrži SSH ključeve za autentifikaciju. Uz to, glavni čvor mora unaprijed znati IP adrese svih čvorova koji će sudjelovati u izračunima i uspostaviti inicijalnu vezu s njima kako bi SSH protokol funkcionirao. Zbog toga je bilo potrebno implementirati dodatnu logiku kojom će glavni čvor čekati na sve čvorove za izračun koji si mu dodijeljeni.

4. Definicija problema

4.1. CosmoLLM aplikacija

CosmoLLM je aplikacija koja je dizajnirana za rješavanje problema vezanih uz istraživanje evolucije tamne energije kroz korištenje naprednih statističkih metoda kao što su Bayesova statistika i Monte Carlo Markovljevi lanci. Ideja za implementaciju korisničkog sučelja zasnovanog na prirodnom jeziku proizašla je iz članka "Istraživanja o evoluciji tamne energije", a svrha rada je olakšati korištenje programske biblioteke "zeus-mcmc" (spomenuto u poglavlju 2.2.) za ispitivanje različitih parametrizacija tamne energije. Aplikacija omogućava fizičarima, koji možda nemaju programersko znanje, da koriste sofisticirane alate za istraživanje kozmoloških problema putem sučelja koje interpretira prirodni jezik. CosmoLLM generira i izvršava programski kod na temelju korisničkih upita te vraća rezultate u razumljivom formatu, bilo da se radi o tekstu, grafovima ili tablicama. Cilj aplikacije je omogućiti efikasnije ispitivanje hipoteza, što doprinosi razvoju kozmologije i boljem razumijevanju svemira[18].

4.2. Puštanje u pogon CosmoLLM aplikacije

CosmoLLM aplikacija sama po sebi je zamišljena tako da ju koristi jedan korisnik, koji postavlja upite i dobiva rezultate. Tijekom razvoja aplikacije, bilo je dostatno provoditi testiranje lokalno na osobnom računalu, ali kako bi se aplikacija mogla efektivno koristiti u praksi, potrebno ju je pokrenuti na poslužitelju. S obzirom na to da je sučelje aplikacije dizajnirano kao web stranica, ovo postaje problem puštanja u pogon web aplikacije. U praksi postoje još dva bitna zahtjeva za funkcionalnost aplikacije: višekorisnička podrška i skalabilnost računanja.

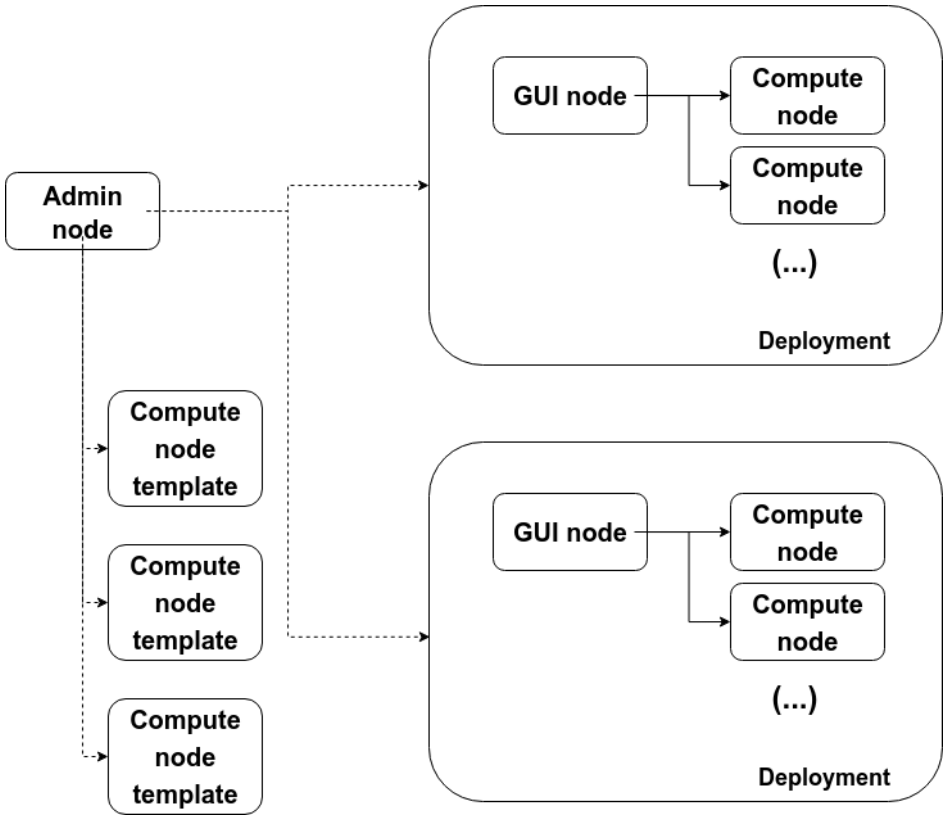
4.2.1. Višekorisnička podrška

Kako bi aplikacija bila korisna u praksi, potrebno je omogućiti višekorisničku podršku. To znači da više korisnika može koristiti aplikaciju istovremeno, svaki s vlastitim korisničkim računom i pristupom. Ovo je posebno važno u okruženjima gdje više korisnika dijeli resurse, kao što su istraživački centri, laboratoriji i slično. Višekorisnička podrška zahtijeva implementaciju autentifikacije i autorizacije korisnika, upravljanje korisničkim računima i pristupom, te izolaciju podataka i resursa među korisnicima.

4.2.2. Skalabilnost računanja

Drugi bitan zahtjev za aplikaciju je skalabilnost računanja. Kako bi se omogućilo brzo i efikasno izvršavanje zahtjevnih statističkih izračuna, potrebno je distribuirati izračune na više čvorova. Ovo se postiže paralelnim izvršavanjem izračuna na različitim čvorovima. Time se može efikasno iskoristiti snaga distribuiranog sustava s mnogo procesora, što će dovesti do bržeg izvršavanja izračuna. Skalabilnost računanja zahtijeva implementaciju distribuiranog računanja, upravljanje resursima i nadzor izvršavanja izračuna. Još jedan bitan zahtjev u kontekstu skalabilnosti računanja je mogućnost dinamičnog dodjeljivanja i oslobađanja komputacijskih resursa za svakog korisnika zasebno.

5. Rješenje

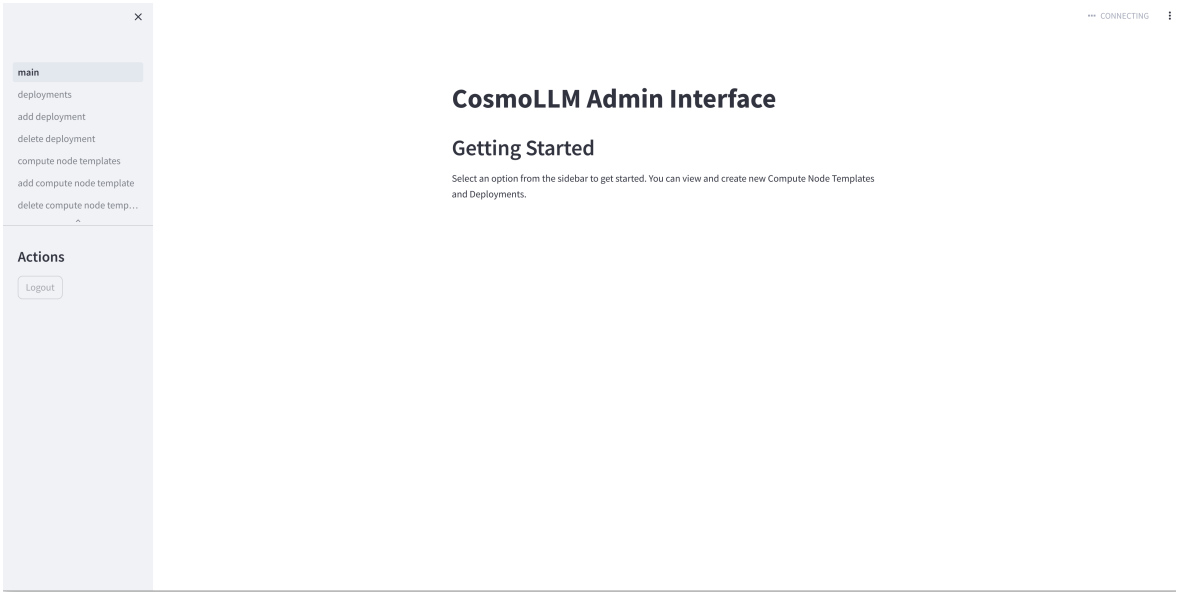


Slika 5.1. Arhitektura rješenja

Rješenje problema je ostvareno kroz 5 elemenata koji zajedno čine sustav upravljanja i korištenja aplikacije CosmoLLM. Kao što je vidljivo na slici 5.1., elementi su redom: administracijski čvor (engl. Admin Node), instanca aplikacije (engl. Deployment), korisnički čvorovi s grafičkim sučeljem (engl. GUI Node), vrsta čvora za izračune (engl. Compute Node Template) i instanca čvora za izračune (engl. Compute node).

5.1. Elementi arhitekture

5.1.1. Administracijski čvor



Slika 5.2. Sučelje administracijskog čvora

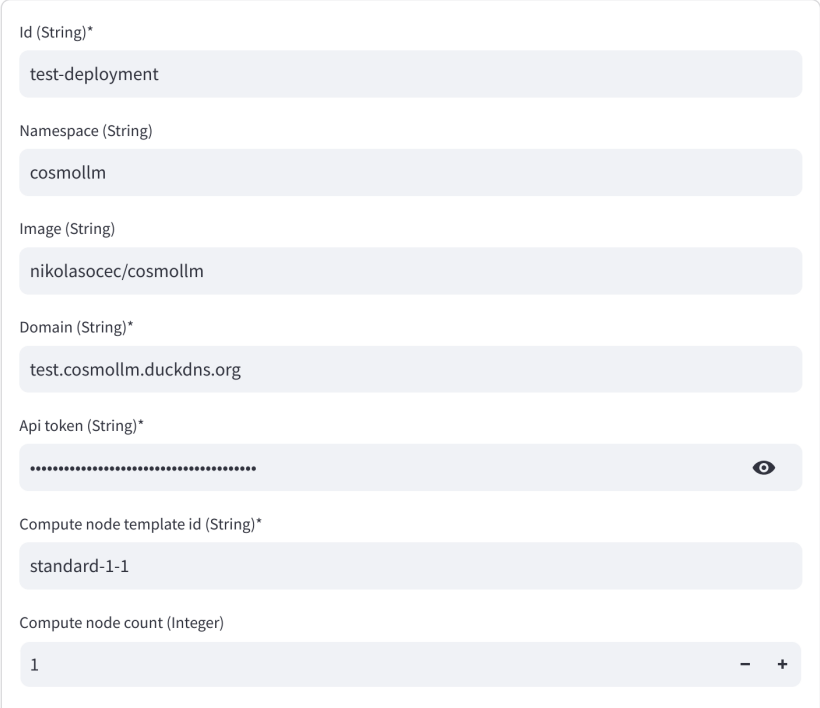
```
NAME↑
cosmollm-admin-54954967f4-2lmhq
mongodb-694cb44f4b-hgh2d
```

Slika 5.3. Prikaz Kubernetes podova - administracijski čvor i MongoDB baza

Administracijski čvor je glavni element sustava koji upravlja stvaranjem svih ostalih elemenata. Nudi grafičko sučelje za administratorskog korisnika kroz web preglednik, preko kojeg se mogu stvarati, modificirati i brisati vrste čvorova za izračune te instance aplikacije. Pristup administracijskom čvoru je ograničen na administratorske korisnike, koji se autentificiraju putem korisničkog imena i lozinke. Za funkcioniranje čvora je potrebna MongoDB baza podataka, koja se koristi za pohranu podataka o korisnicima, vrstama čvorova i instancama aplikacije.

5.1.2. Instanca aplikacije

Instanca aplikacije je apstraktni element koji predstavlja jednu konfiguraciju aplikacije CosmoLLM. Svaka instanca aplikacije ima svoj jedinstveni identifikator, koji se koristi za referenciranje i upravljanje instancom. Instanca aplikacije se sastoji od jednog ili više čvorova za izračune i točno jednog čvora s korisničkim sučeljem, koji se pokreću unutar Kubernetes clustera. Svaki čvor za izračune je zaseban pod u Kubernetes terminologiji. Uz konfiguraciju čvorova, instanca aplikacije se brine o pružanju zajedničkog datotečnog sustava koji povezuje korisnički čvor i čvorove za izračun. U taj datotečni sustav



Slika 5.4. Model instance aplikacije (prvih nekoliko polja)

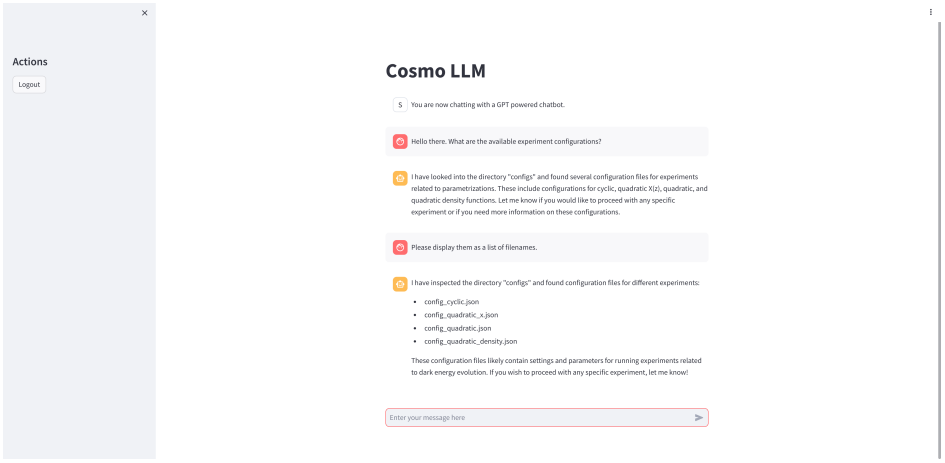
```
NAME↑
cosmollm-admin-54954967f4-2lmhq
cosmollm-compute-test-deployment-0-597bbf4c75-fzf8x
cosmollm-compute-test-deployment-1-fd486f4b6-8v8n6
cosmollm-compute-test-deployment-2-667fbb8ff7-kdkgn
cosmollm-compute-test-deployment-3-64b5bdb696-ncx7z
cosmollm-compute-test-deployment-4-859d5ccdf9-hzzrw
cosmollm-gui-test-deployment-66584d7d96-t9rxk
mongodb-694cb44f4b-hgh2d
```

Slika 5.5. Prikaz Kubernetes podova - instanca aplikacije s 5 čvorova za izračune

se spremaju rezultati izračuna (distribucije i grafičke reprezentacije rezultata), kojima zatim korisnički čvor direktno pristupa i prikazuje ih korisniku prema zahtjevu. Instanca aplikacije se može skalirati dodavanjem ili uklanjanjem čvorova za izračune kroz administracijsko sučelje.

5.1.3. Korisnički čvorovi s grafičkim sučeljem

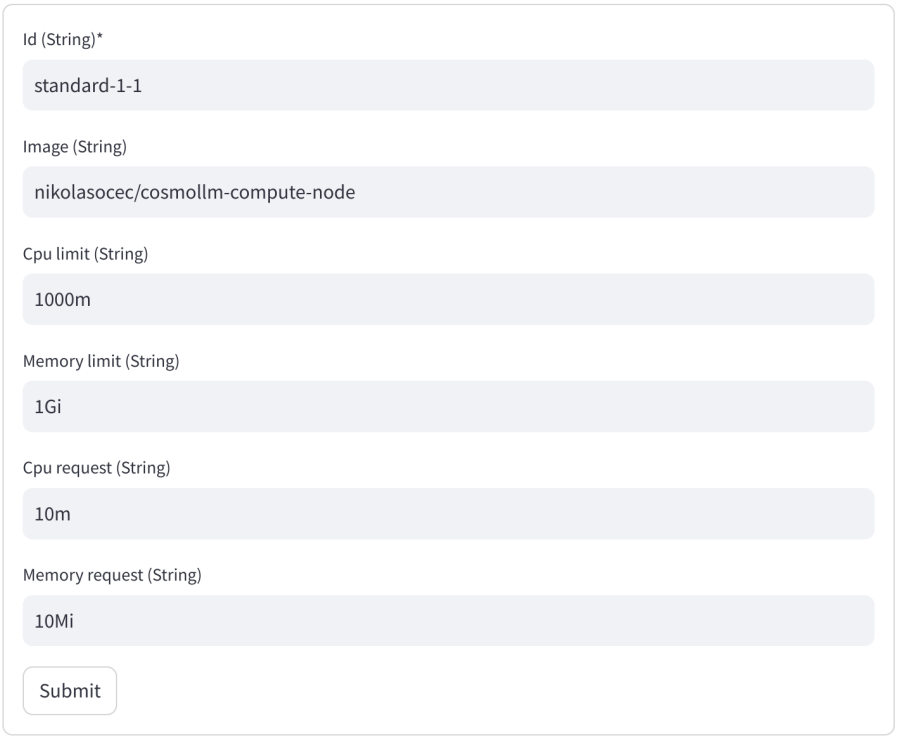
Korisnički čvor s grafičkim sučeljem je element koji omogućuje korisnicima pristup aplikaciji putem web preglednika. Svaki korisnički čvor ima svoje jedinstveno korisničko sučelje. Korisnički čvorovi su izolirani jedni od drugih, što znači da svaki korisnik ima svoj vlastiti korisnički čvor s vlastitim resursima i podacima. Korisnički čvorovi su povezani s instancom aplikacije preko Kubernetes servisa, što im omogućuje komunikaciju s čvorovima za izračune. Autentifikacija korisnika je izvedena istim mehanizmom kao i za



Slika 5.6. Model korisničkog čvora s grafičkim sučeljem

administracijski čvor, preko korisničkog imena i lozinke, što uvodi ovisnost o MongoDB bazi podataka.

5.1.4. Vrsta čvora za izračune



Slika 5.7. Model vrste čvora za izračune

Vrsta čvora za izračune je apstraktni element koji predstavlja jednu konfiguraciju čvora za izračune. Kao što je vidljivo na slici 5.7., vrsta čvora za izračune definira parametre čvora: maksimalno opterećenje procesora, maksimalnu količinu memorije te

zahtjeve za dodijeljenim resursima. Zahtjevi za resursima govore Kubernetes clusteru koliko resursa je potrebno dodijeliti svakom čvoru za izračune, kako bi se mogli dobro rasporediti na Kubernetes čvorove (u sustavima koji imaju samo jedan Kubernetes čvor, zahtjevi nisu previše bitni). Vrsta čvora za izračune se koristi prilikom stvaranja nove instance aplikacije, kako bi se definirali parametri čvorova za izračune koji će se koristiti.

5.1.5. Instanca čvora za izračune

Instanca čvora za izračune je konkretna implementacija vrste čvora za izračune, koja se pokreće unutar Kubernetes clustera. Svaka instanca čvora za izračune je zaseban pod u Kubernetes terminologiji. Ovi čvorovi nemaju nikakvo korisničko sučelje. Instanca čvora za izračune pokreće SSH server, koji omogućuje glavnom čvoru da se poveže s njim preko MPI protokola i pokrene izračune. Kao što je spomenuto u sekciji 5.1.2., instanca čvora za izračune je povezana s korisničkim čvorom preko zajedničkog datotečnog sustava, što omogućuje pohranu i dijeljenje rezultata izračuna.

6. Analiza skalabilnosti i performansi sustava

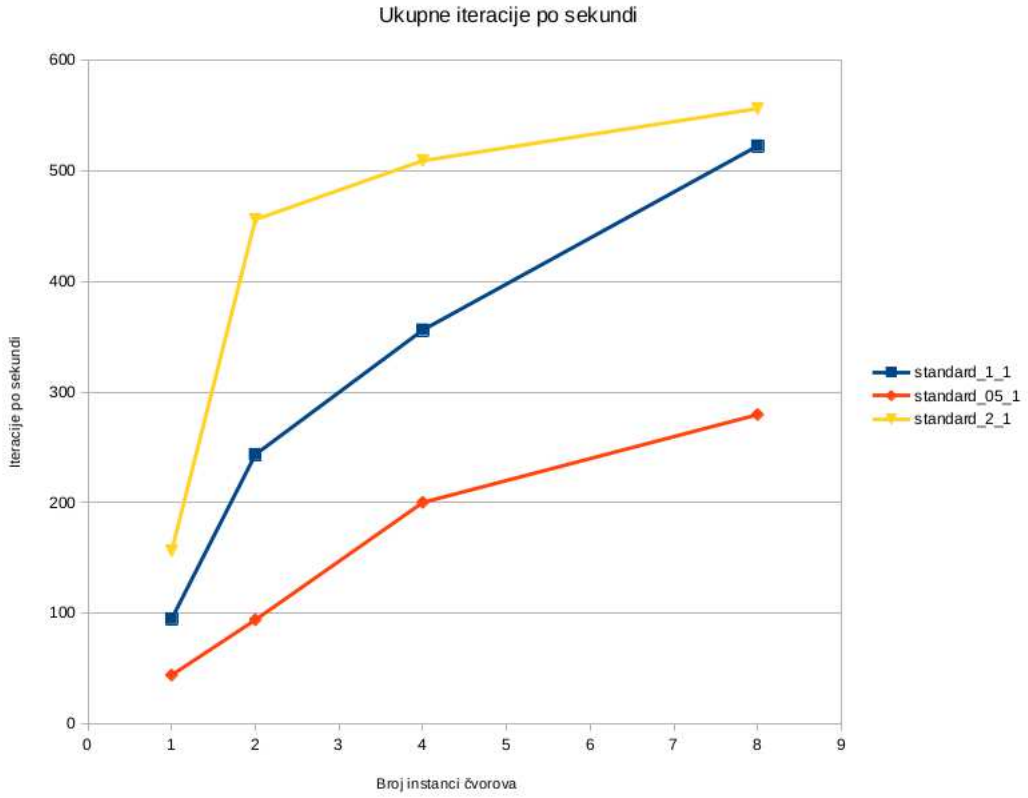
Za analizu skalabilnosti i performansi sustava, proveden je eksperiment u kojem se testiraju različite konfiguracije sustava. Cilj eksperimenta je bio utvrditi kako se sustav ponaša ovisno o tome koliko čvorova za računanje je dodijeljeno korisniku i ovisno o snazi tih čvorova. Eksperimenti su provedeni na lokalnom Kubernetes clusteru, koristeći jedan Kubernetes čvor sa 12 virtualnih jezgri (Intel i7-8700k@5GHz, 6 fizičkih jezgri uz "Intel Hyper-Threading") i 32 GB RAM-a.

6.1. Metodologija

U provedenom eksperimentu se testiraju tri vrste čvora za izračune i po četiri konfiguracije instance aplikacije za svaku vrstu čvora. Vrste čvorova za izračune su: *standard_1_1* (1 jezgra, 1 GB RAM-a), *standard_2_1* (2 jezgre, 1 GB RAM-a) i *standard_05_1* (0.5 jezgri, 1 GB RAM-a). Konfiguracije instance aplikacije za svaku vrstu čvora su: 1, 2, 4 i 8 čvorova za izračune. Za svaku konfiguraciju je pokrenut isti izračun uz pomoć sljedeće naredbe pokrenute unutar konzole čvora s korisničkim sučeljem: "python3 tests/benchmarks/mpi_mcmc_scaling.py -r 5". Konzoli se pristupa kroz Kubernetes alat "kubectl". Ova skripta pokreće iste alate koje bi pokrenuo i LLM tijekom normalnog rada, ali je znatno lakše prikupiti detaljne podatke o trajanju izračuna kroz konzolu. Prilikom svakog pokretanja skripte, broj lanaca u korištenoj MCMC konfiguraciji je promijenjen na broj čvorova za izračune. Koristi se konfiguracija "configs/config_quadratic_density.json", kojoj je pretpostavljena količina lanaca 4. S obzirom na to da se broj lanaca prilagođuje broju čvorova za izračune, onda je npr. u konfiguraciji s 2 čvora za izračune broj lanaca bio 2. Parametar "-r 5" označava da se svaka konfiguracija pokreće 5 puta, kako bi se dobila prosječna vrijednost trajanja izračuna. Pokretanjem naredbe se u konzolu ispisuju

informacije o trajanju izračuna za svaki lanac i za ukupni broj provedenih iteracija u tom lancu. Ovi podatci se zatim prikupljaju te im se računa prosjek. Na temelju prosječnih trajanja i broja iteracija, računa se prosječna brzina izračuna za svaku konfiguraciju. Prosječna brzina izračuna se na kraju množi s brojem lanaca jer se svi lanci izračunavaju paralelno, što daje ukupnu brzinu izračuna za svaku konfiguraciju u iteracijama po sekundi.

6.2. Rezultati



Slika 6.1. Rezultati eksperimenta

Kao što je vidljivo na slici 6.1., rezultati eksperimenta pokazuju da se brzina izračuna povećava otprilike linearno s brojem čvorova za izračune, ali se smanjuje s manjom snagom čvorova. Konkretno, konfiguracija s 2 čvora za izračune tipa *standard_05_1* je dvostruko brža od konfiguracije s 1 čvorom *standard_05_1*, ali je otprilike dvaput sporija od konfiguracije s 2 čvora tipa *standard_1_1*. Ovo pokazuje da se izračuni skaliraju gotovo linearno s brojem jezgri upakiranima u čvorove za izračune. Bitno je napomenuti da konfiguracije od 8 *standard_1_1* čvorova te 4 i 8 *standard_2_1* čvora prelaze ograničenje fizičkog procesora korištenog u eksperimentu. Naime, korišteni procesor ima 6 fizičkih

jezgri, a priroda ovog računskog zadatka je takva da dijeljeni resursi između dvije virtualne jezgre na jednoj fizičkoj jezgri mogu izvršavati samo jedan izračun odjednom, a ne dva. Zato vidimo da konfiguracije koje ukupno očekuju više od 6 jezgri ne dobivaju očekivano linearno povećanje brzine izračuna, već se ponašaju kao da imaju samo 6 jezgri.

7. Upute za korištenje

Za instalaciju CosmoLLM sustava na vlastiti poslužitelj, potrebno je imati instaliranu i konfiguriranu Kubernetes platformu, kao i lokalne alate za pristup clusteru (Kubectl i Helm). Upute za instalaciju ovih alata izlaze izvan okvira ovog rada. U ovom poglavlju će biti opisani koraci za instalaciju i korištenje CosmoLLM sustava, uključujući postavljanje i korištenje administracijskog čvora, iz kojeg se dalje kroz sučelje može upravljati ostalim elementima sustava.

7.1. Instalacija administracijskog čvora i potrebnih alata

Instalacija administracijskog čvora i potrebnih alata uključuje sljedeće korake:

1. **Kloniranje CosmoLLM repozitorija:** Klonirajte CosmoLLM repozitorij s GitHub-a na lokalno računalo pomoću naredbe: “git clone https://github.com/COALA-Lab/CosmoLLM.git“

2. **Pozicioniranje u direktorij s alatima za instalaciju administracijskog čvora:** Terminalom se pozicionirajte u direktorij “./CosmoLLM/deployment/k8s“

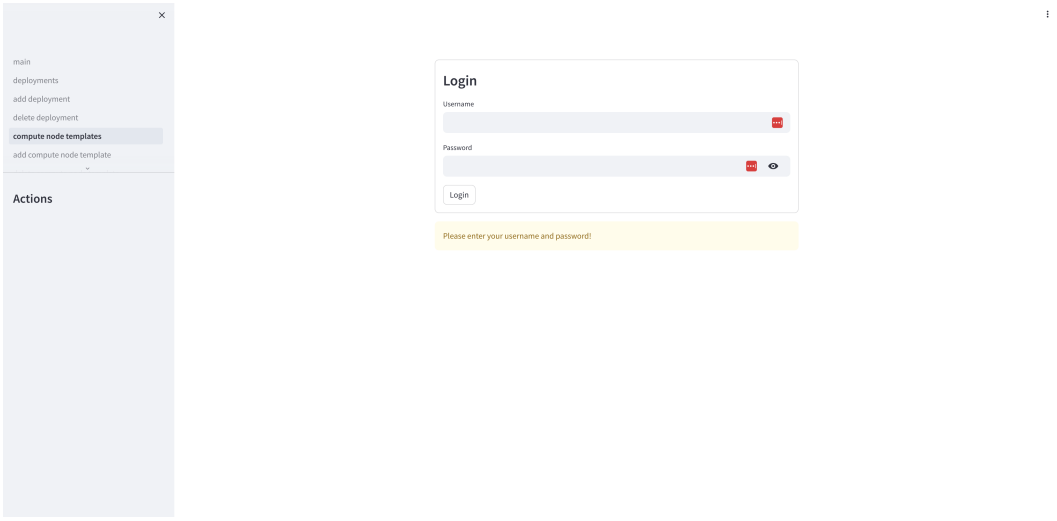
3. **Postavljanje StorageClass objekta:** Pokrenite naredbu “python3 deploy_storage_class.py“. Ovo će instalirati StorageClass objekt u Kubernetes clusteru, koji koristi lokalni disk za pohranu i nazvat će ga “local-storage“. Ukoliko trebate koristiti drugi tip pohrane, možete pokrenuti skriptu s drugim parametrima. Pokretanje “python3 deploy_storage_class.py -h“ će prikazati sve opcionalne parametre.

4. **Instalacija MongoDB baze podataka:** Pokrenite naredbu “python3 deploy_mongo.py --namespace <namespace> --mongo-user root --mongo-password <password>“. <namespace> može biti bilo što, ali razumna vrijednost je “cosmollm“. <password>

je lozinka koju želite postaviti za MongoDB korisnika.

5. **Instalacija administracijskog čvora:** Pokrenite naredbu
   “python3 deploy_admin.py \
   --namespace <namespace> \
   -i nikolasocec/cosmollm-admin-node \
   -d <domena> \
   --kube-config-path <kube config datoteka> \
   --mongo-url mongodb:27017 \
   --mongo-user root \
   --mongo-password <Mongo password> \
   --admin-user admin \
   --admin-password <administratorska lozinka>“
   <namespace> je isti kao i u prethodnom koraku. <domena> je domena na kojoj će biti dostupan administracijski čvor. <kube config datoteka> je putanja do kube config datoteke koja se koristi za pristup vašem Kubernetes clusteru (obično “ /.kube/config“). <Mongo password> je lozinka koju ste postavili za MongoDB korisnika u prethodnom koraku. <administratorska lozinka> je lozinka koju želite postaviti za administratorskog korisnika.

7.2. Korištenje administracijskog čvora

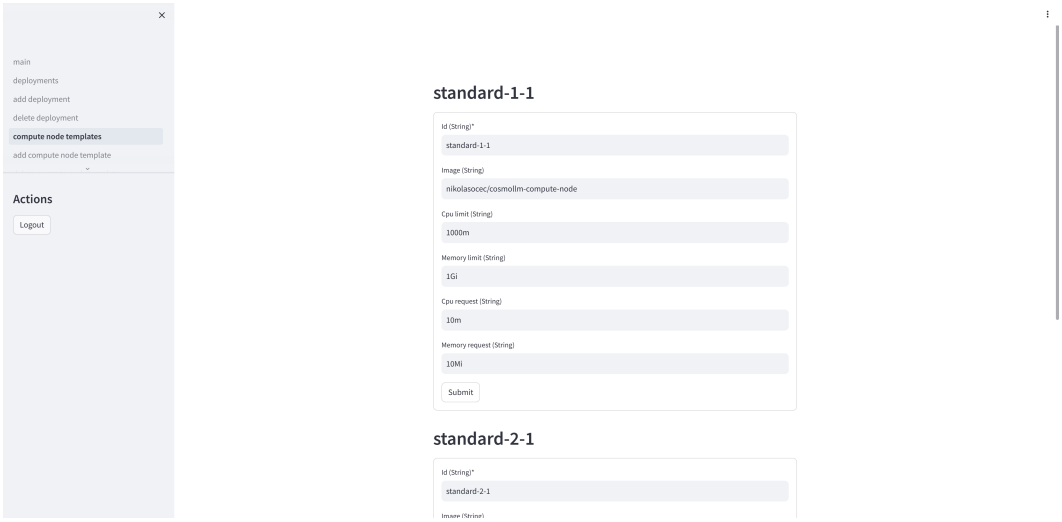


Slika 7.1. Stranica za prijavu u administracijski čvor

Nakon što je administracijski čvor uspješno instaliran, možete pristupiti sučelju preko

web preglednika. Da biste to učinili, otvorite web preglednik i upišite adresu “https://<domena>“. Bit ćete preusmjereni na stranicu za prijavu, gdje možete unijeti korisničko ime i lozinku koje ste postavili prilikom instalacije. Nakon uspješne prijave, bit ćete preusmjereni na početnu stranicu administracijskog čvora, gdje možete upravljati vrstama čvorova za izračune i instancama aplikacije.

7.2.1. Upravljanje vrstama čvorova za izračune

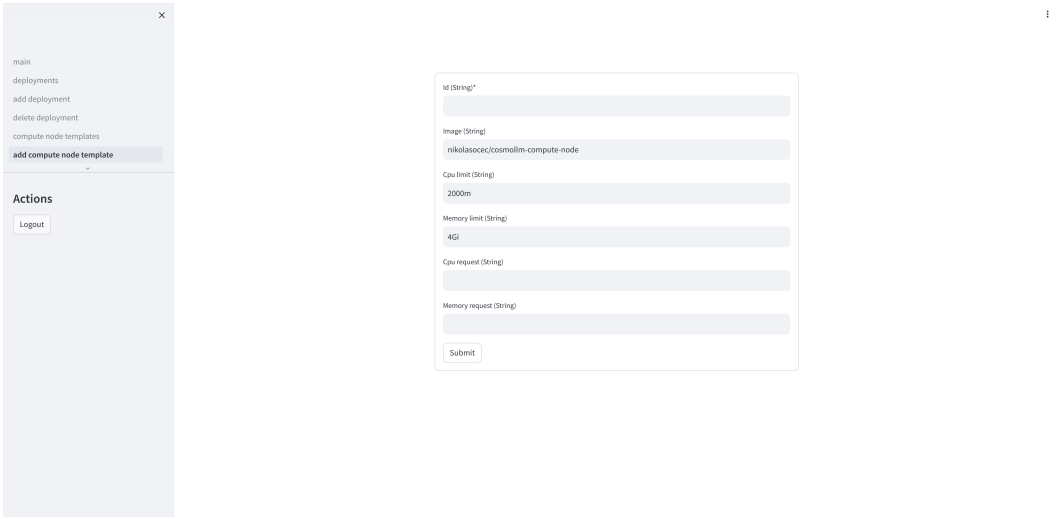


Slika 7.2. Stranica s popisom vrsta čvorova za izračune

Upravljanje vrstama čvorova za izračune je raspoređeno na tri stranice: popis svih vrsta čvorova (“compute node templates“), dodavanje nove vrste čvora (“add compute node template“) i brisanje vrste čvora (“delete compute node template“). Na stranici s popisom vrsta čvorova (slika 7.2.)) možete vidjeti sve trenutno dostupne vrste čvorova za izračune. Za svaku vrstu čvora možete vidjeti i mijenjati njihove parametre. Inicijalno će ova stranica biti prazna, jer nema unaprijed definiranih vrsta čvorova.

Na stranici za dodavanje nove vrste čvora za izračune (slika 7.3.) možete unijeti parametre za novu vrstu čvora. Prilikom unosa parametara, sustav će provjeriti ispravnost unesenih podataka i obavijestiti vas o eventualnim greškama. Parametri su:

- **Id:** jedinstveni identifikator vrste čvora. Mora se sastojati od malih slova, brojeva i crta (-).
- **Image:** Docker image koji će se koristiti za ovu vrstu čvora. Pretpostavljena vrijednost je “nikolasocec/cosmollm-compute-node“.



Slika 7.3. Stranica za dodavanje vrste čvora za izračune

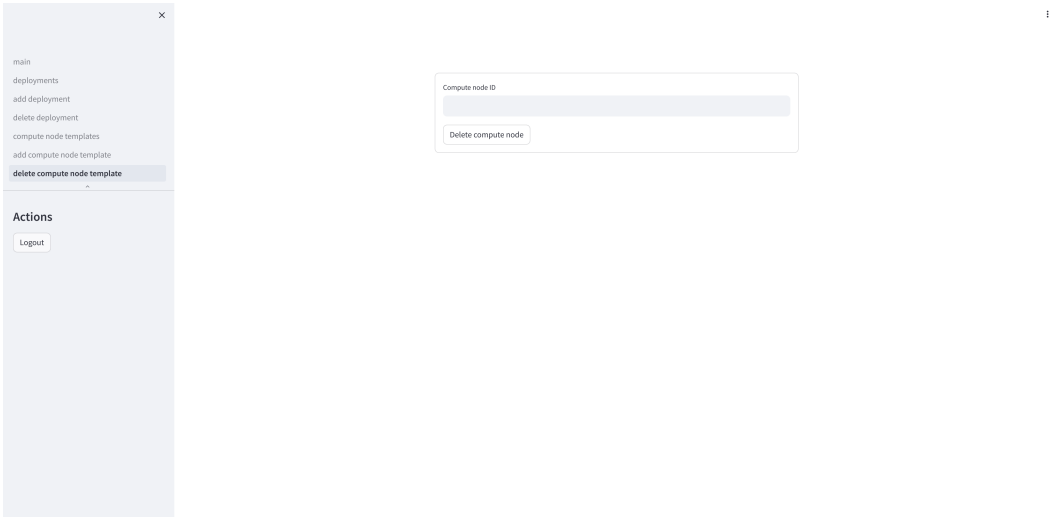
- **Cpu limit:** maksimalno opterećenje procesora koje će čvor moći ostvariti. Ovo efektivno može pretvoriti jedan fizički procesor u više virtualnih procesora. Mjerna jedinica je u tisućinama procesora, tako da npr. vrijednost 1000m znači da čvor može koristiti 100% jedne procesorske jezgre, 2000m 100% dvije jezgre, 2500m dvije i pol jezgre itd.

- **Memory limit:** maksimalna količina memorije koju će čvor moći koristiti. Mjerna jedinica je u gibibajtima, mibibajtima itd. Npr. vrijednost 4Gi znači da čvor može koristiti maksimalno 4 gibibajta memorije. Za razliku od limitiranja procesorskog vremena, gdje je normalno da čvor koristiti onoliko procesorskog vremena koliko mu je maksimalno dozvoljeno, prelazak limitacije memorije će uzrokovati prekid izvršavanja čvora.

- **Cpu request:** minimalno opterećenje procesora koje će čvor imati na raspolaganju. Više detalja o zahtjevima je dano u 5.1.4. U sustavima s jednim Kubernetes čvorom, ovaj parametar može biti zanemaren.

- **Memory request:** minimalna količina memorije koju će čvor imati na raspolaganju. Vrijede ista pravila kao i za zahtjeve procesora.

Na stranici za brisanje vrste čvora za izračune (slika 7.4.) možete izbrisati vrstu čvora ako se ona ne koristi (sustav će to provjeriti). Brisanje se provodi upisom identifikatora vrste čvora u polje za unos i pritiskom na gumb “Delete“. Sustav će vas pitati za potvrdu



Slika 7.4. Stranica za brisanje vrste čvora za izračune

brisanja, a nakon potvrde će izbrisati vrstu čvora.

7.2.2. Upravljanje instancama aplikacije

Upravljanje instancama aplikacije funkcionira na vrlo sličan način kao i upravljanje vrstama čvorova za izračune. Analogna imena stranica su “deployments“, “add deployment“ i “delete deployment“. Jedina bitna razlika su parametri koji se unose prilikom stvaranja nove instance aplikacije:

- **Id:** jedinstveni identifikator instance aplikacije. Vrijede ista pravila kao za identifikator vrste čvora za izračune.
- **Namespace:** Kubernetes namespace u kojem će se pokrenuti nova instanca aplikacije. Pretpostavljena vrijednost je “cosmollm“.
- **Image:** Docker image koji će se koristiti za korisnički čvor. Pretpostavljena vrijednost je “nikolasocec/cosmollm“.
- **Domain:** domena na kojoj će biti dostupan korisnički čvor. Vrijednost mora biti validna domena. Vrijede ista pravila kao kod dodavanja administracijskog čvora.
- **Api token:** OpenAI API token koji će se koristiti za pristupanje GPT modelu potrebnom za funkcioniranje aplikacije.
- **Compute node template id:** identifikator vrste čvora za izračune koja će se koristiti za ovu instancu. Vrijednost mora biti identifikator neke od postojećih vrsta čvorova za izračune.
- **Compute node count:** broj čvorova za izračune koji će se pokrenuti unutar ove instance. Vrijednost mora biti pozitivan cijeli broj.
- **Admin user:** korisničko ime novog administratorskog korisnika koji će imati pristup ovoj instanci.
- **Admin password:** lozinka novog administratorskog korisnika.
- **Mpi host slots:** broj MPI procesa koje svaki čvor za izračune može pokrenuti u paraleli. Vrijednost mora biti pozitivan cijeli broj. Pretpostavljena (i preporučena) vrijednost je 2.

- **Storage class:** StorageClass objekt koji će se koristiti za pohranu podataka. Vrijednost mora biti identifikator postojećeg StorageClass objekta. Pretpostavljena vrijednost je “local-storage“.
- **Storage size:** veličina datotečnog sustava za trajnu pohranu podataka koji će biti dodijeljen aplikaciji. Vrijednost je u gibibajtima. Pretpostavljena vrijednost je 4Gi.
- **Mongo url:** URL MongoDB baze podataka. Pretpostavljena vrijednost je “mongodb:27017“.
- **Mongo user:** korisničko ime MongoDB korisnika. Pretpostavljena vrijednost je “root“.
- **Mongo password:** lozinka MongoDB korisnika.

Konfiguracija instanci aplikacije se može mijenjati kroz administracijsko sučelje, a dozvoljene promjene će se odmah reflektirati na instancu aplikacije.

S obzirom na to da je ova konfiguracija zahtjevna, većina parametara je postavljena na pretpostavljene vrijednosti koje će raditi u većini slučajeva. Najbitniji parametri na koje treba obratiti pažnju su “Id“, “Domain“, “Api token“, “Compute node template id“, “Compute node count“, “Admin user“, “Admin password“, “Mongo user“ i “Mongo password“.

7.3. Korištenje korisničkog čvora

Nakon što je instanca aplikacije uspješno pokrenuta, možete pristupiti korisničkom čvoru preko web preglednika. Da biste to učinili, otvorite web preglednik i upišite adresu “https://<domena>“. Bit ćete preusmjereni na stranicu za prijavu, gdje možete unijeti korisničko ime i lozinku koje ste postavili prilikom konfiguracije instance aplikacije. Nakon uspješne prijave, bit ćete preusmjereni na početnu stranicu korisničkog čvora, gdje možete postavljati upite i dobiti rezultate.

Sučelje korisničkog čvora je dizajnirano tako da omogućava korisnicima jednostavno postavljanje upita i pregledavanje rezultata kroz razgovor s LLM-om. Na početnoj stranici možete vidjeti polje za unos upita, gumb za slanje upita i polje za prikaz rezultata.

Nakon što unesete upit i pritisnete gumb za slanje, aplikacija će izvršiti upit i prikazati rezultate u polju za prikaz u obliku razgovora. Rezultati se mogu sastojati od teksta, grafova, tablica i drugih oblika podataka, a prikazuju se u obliku koji je najprikladniji za prikazivanje. Više detalja o korištenju aplikacije možete pronaći u radu "Razvoj korisničkog sučelja na prirodnom jeziku za kozmološke Monte Carlo simulacije"[18].

8. Zaključak

U ovom završnom radu, prikazali smo proces puštanja u pogon aplikacije CosmoLLM s podrškom za distribuirano računanje i višekorisničku komunikaciju. Cilj je bio implementirati sustav koji omogućuje učinkovito izvršavanje kompleksnih MCMC izračuna korištenjem OpenMPI protokola unutar Kubernetes okruženja.

Kroz poglavlja rada detaljno smo opisali teorijsku podlogu distribuiranog računanja, OpenMPI protokola, te orkestracije uz pomoć Kubernetesa. Definirali smo problematiku vezanu uz skalabilnost i višekorisničku podršku, te smo implementirali rješenje koje se sastoji od više komponenti: administracijskog čvora, instanci aplikacije, korisničkih čvorova s grafičkim sučeljem, vrsta čvorova za izračune i instanci čvorova za izračune.

Provedenom analizom performansi i skalabilnosti sustava pokazali smo kako se sustav ponaša pod različitim konfiguracijama, te potvrdili da distribuirano računanje uz pomoć OpenMPI-ja i orkestracije Kubernetesa omogućuje skalabilno izvršavanje zahtjevnih izračuna.

Primjena ovog sustava omogućuje korisnicima, posebice fizičarima koji istražuju evoluciju tamne energije, da jednostavno i efikasno koriste napredne statističke metode bez potrebe za dubokim tehničkim znanjem o infrastrukturi koja stoji iza aplikacije.

Za budući rad, predlažemo nekoliko smjerova poboljšanja i proširenja sustava. Fokus treba biti na pojednostavljenju korisničkog sučelja kako bi interakcija s korisničkom, ali i administracijskom aplikacijom bila što intuitivnija. Uz to, potrebno je fokusirati se na dodatno povećanje stabilnosti sustava i oporavak od raznih rubnih slučajeva koji još nisu pokriveni. Konačno, potrebno je napraviti detaljnu analizu sigurnosti sustava i implementirati odgovarajuće mjere kako bi se dodatno osiguralo da su podaci korisnika sigurni i privatni.

Literatura

[1] A. W. Services, “What is distributed computing?” https://aws.amazon.com/what-is/distributed-computing/, [posjećeno 2024-06-01].

[2] H. Sankar, “Challenges in distributed systems”, https://www.scaler.com/topics/challenges-of-distributed-system/, Lipanj 2023., [posjećeno 2024-06-01].

[3] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, i T. S. Woodall, *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*. Springer, Berlin, Heidelberg, 2004.

[4] M. Karamanis, F. Beutler, i J. A. Peacock, “zeus: A python implementation of ensemble slice sampling for efficient bayesian parameter inference”, *arXiv preprint arXiv:2105.03468*, 2021.

[5] R. Hat, “What is orchestration?” https://www.redhat.com/en/topics/automation/what-is-orchestration, Veljača 2024., [posjećeno 2024-06-02].

[6] T. K. Authors, “Kubernetes documentation - concepts - overview”, https://v1-26.docs.kubernetes.io/docs/concepts/overview/, 2022., [posjećeno 2024-06-02].

[7] ——, “Kubernetes documentation - concepts - overview - kubernetes components”, https://v1-26.docs.kubernetes.io/docs/concepts/overview/components/, 2022., [posjećeno 2024-06-02].

[8] ——, “Kubernetes documentation - concepts - containers”, https://v1-26.docs.kubernetes.io/docs/concepts/containers/, 2022., [posjećeno 2024-06-02].

[9] ——, “Kubernetes documentation - concepts - workloads - pods”, https://v1-26.docs.kubernetes.io/docs/concepts/workloads/pods/, 2022., [posjećeno 2024-06-02].

[10] ——, “Kubernetes documentation - concepts - workloads - workload resources - deployments”, https://v1-26.docs.kubernetes.io/docs/concepts/workloads/controllers/deployment/, 2022., [posjećeno 2024-06-02].

[11] ——, “Kubernetes documentation - concepts - services, load balancing, and networking - services”, https://v1-26.docs.kubernetes.io/docs/concepts/services-networking/service/, 2022., [posjećeno 2024-06-02].

[12] ——, “Kubernetes documentation - concepts - services, load balancing, and networking - ingress”, https://v1-26.docs.kubernetes.io/docs/concepts/services-networking/ingress/, 2022., [posjećeno 2024-06-02].

[13] ——, “Kubernetes documentation - concepts - storage - volumes”, https://v1-26.docs.kubernetes.io/docs/concepts/storage/volumes/, 2022., [posjećeno 2024-06-02].

[14] ——, “Kubernetes documentation - concepts - configuration - configmaps”, https://v1-26.docs.kubernetes.io/docs/concepts/configuration/configmap/, 2022., [posjećeno 2024-06-02].

[15] ——, “Kubernetes documentation - concepts - configuration - secrets”, https://v1-26.docs.kubernetes.io/docs/concepts/configuration/secret/, 2022., [posjećeno 2024-06-02].

[16] ——, “Kubernetes documentation - concepts - overview - working with kubernetes objects - namespaces”, https://v1-26.docs.kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/, 2022., [posjećeno 2024-06-02].

[17] ——, “Kubernetes documentation - concepts - storage - storage classes”, https://v1-26.docs.kubernetes.io/docs/concepts/storage/storage-classes/, 2022., [posjećeno 2024-06-02].

[18] A. Rakocija, “Razvoj korisničkog sučelja na prirodnom jeziku za kozmološke monte carlo simulacije”, *Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva*, 2024.

Sažetak

Puštanje u pogon aplikacije CosmoLLM s podrškom za distribuirano računanje i višekorisničku komunikaciju

Nikola Sočec

Ovaj završni rad prikazuje postupak puštanja u pogon aplikacije CosmoLLM koja podržava distribuirano računanje i višekorisničku komunikaciju. CosmoLLM omogućuje korisnicima interakciju s velikim jezičnim modelom (LLM) za provođenje izračuna temeljenih na Monte Carlo Markovljevim lancima (MCMC). Aplikacija je implementirana koristeći OpenMPI za distribuirano računanje i Kubernetes za orkestraciju. Sustav je dizajniran s tri glavne komponente: administracijski čvor, čvor za korisničko sučelje i čvorovi za izračune. Analizirane su performanse i skalabilnost sustava te su potvrđene prednosti distribuiranog pristupa. Rad završava prijedlozima za buduća poboljšanja i proširenja sustava.

Ključne riječi: UI; LLM; NLP; Kubernetes; Distribuirano računanje; OpenMPI

Abstract

Puštanje u pogon aplikacije CosmoLLM s podrškom za distribuirano računanje i višekorisničku komunikaciju

Nikola Sočec

This thesis presents the process of deploying the CosmoLLM application with support for distributed computing and multi-user communication. CosmoLLM enables users to interact with a large language model (LLM) to perform calculations based on Monte Carlo Markov Chains (MCMC). The application is implemented using OpenMPI for distributed computing and Kubernetes for orchestration. The system is designed with three main components: an administrative node, a user interface node, and compute nodes. The performance and scalability of the system are analyzed, confirming the benefits of the distributed approach. The thesis concludes with suggestions for future improvements and system expansions.

Keywords: AI; LLM; NLP; Kubernetes; Distributed computing; OpenMPI