UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA RAČUNARSTVO

SEMINARSKI RAD

PARALELNO IZVRŠENJE ALGORITMA MAŠINSKOG UČENJA U DOCKER/KUBERNETES OKRUŽENJU

Lazar Ignjatović

Niš, februar 2023. god.

SADRŽAJ

1 UVOD

Mašinsko učenje predstavlja granu računarstva koja se bavi stvaranjem algoritama koji omogućavaju računarima da uče iz podataka, kako bi se stvorile predikcije i doneli zaključci bez eksplicitnog programiranja. Izuzetno je važno u velikom broju industrija i istraživanja, kao što su medicina, finansije, marketing, transport i mnogi drugi. Međutim, izvršavanje algoritama mašinskog učenja može biti vrlo zahtevno u smislu resursa, zbog velikih količina podataka i složenih algoritama. Zbog toga je paralelno izvršavanje postalo neophodno kako bi se smanjilo vreme izvršavanja, smanjio trošak i povećala efikasnost.

Postoji nekoliko pristupa za postizanje paralelnog izvršavanja algoritama mašinskog učenja, kao što su distribuirano izvršavanje, upotreba grafičkih procesora (GPU) i korišćenje kontejnera. Docker i Kubernetes su se pokazali kao vrlo korisni za upravljanje aplikacijama zasnovanim na kontejnerima. Docker omogućava pakovanje aplikacija i njihovih zavisnosti u kontejnere, dok Kubernetes pruža alate za automatizaciju upravljanja tim kontejnerima. Kombinacija ova dva alata omogućava da se skalira izvršavanje algoritama mašinskog učenja i poboljša efikasnost.

U ovom seminarskom radu, fokus će biti na omogućavanju paralelnog izvršavanja algoritama mašinskog učenja u Docker/Kubernetes okruženju. Za testiranje koristiće se istrenirani Keras model i dataset sa preko 500.000 unosa. Biće data implementacija klijentske i serverske aplikacije. Serverska aplikacija biće upakovana u Docker kontejner i pripremljena za izvršavanje u okviru Kubernetes klastera. Nakon toga biće opisan formirani Kubernetes klaster u okviru fakultetske cloud mreže, kao i dat uvid u deployment serverske aplikacije na dati klaster. Naposletku biće reči o testiranju predloženog sistema i njegovim preformansama.

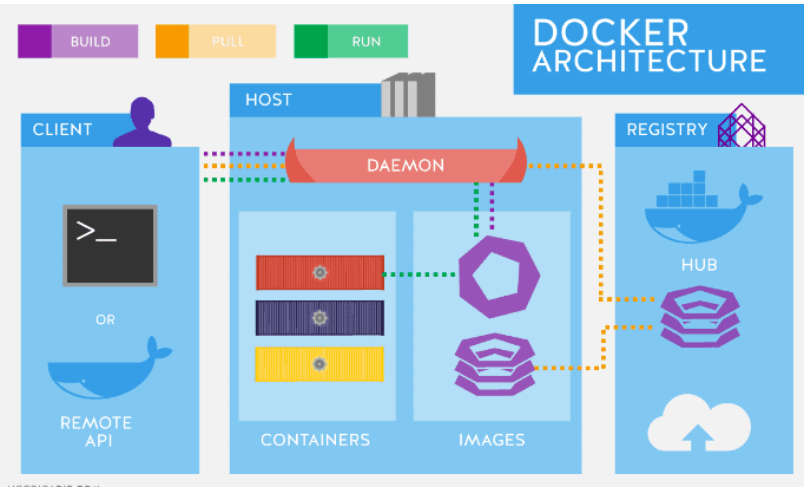
Nakon uvoda, u drugom poglavlju biće data teoretska osnova za implementiranje predloženog sistema. Biće diskutovana Docker i Kubernetes okruženja, njihov princip rada i način konfiguracije. Takođe, biće reči o TensorFlow i Keras bibliotekama, kao i o NFS protokolu. U trećem poglavlju detaljno će biti opisan implementirani sistem. U četvrtom poglavlju biće predočene performanse implementiranog sistema. Zaključak će biti dat u petom poglavlju.

2 TEORETSKA OSNOVA

2.1 Docker

Docker je open-source platforma za upravljanje kontejnerima koja omogućuje izolaciju aplikacija i servisa od host sistema, što je važno za njihovu portabilnost i skalabilnost. Docker koristi koncept kontejnera, koji se mogu smatrati "laganim" virtuelnim mašinama. Kontejneri se mogu posmatrati kao izolovani procesi koji se izvršavaju u okruženju koje omogućuje pristup samo onim resursima koji su potrebni za izvršavanje. Ova izolacija omogućuje jednostavno upravljanje softverskim okruženjima i njihovu distribuciju na različitim platformama [1].

Docker arhitektura (prikazana na slici 1) sastoji se od tri glavna dela: Docker klijenta, Docker hosta i Docker registra. Docker klijent je glavni interfejs prema Docker platformi, dok je Docker host odgovoran za pokretanje i upravljanje kontejnerima. Docker registar služi kao mesto za skladištenje Docker slika, koje predstavljaju osnovu za generisanje kontejnera [1].



Slika 1 – Docker arhitektura

Docker daemon je glavni proces Docker platforme koji omogućava stvaranje, upravljanje i izvršavanje Docker kontejnera. Daemon osluškuje Docker API i obrađuje zahteve korisnika, stvarajući i pokrećući Docker kontejnere na osnovu instrukcija koje prima. On takođe održava lokalni registar s Docker slikama i upravlja mrežnim resursima potrebnim za komunikaciju između kontejnera. Docker daemon je jednostavan za instalaciju i konfiguraciju, a nudi bogat set značajki za upravljanje Docker kontejnerima, uključujući podršku za skaliranje, povezivanje s mrežom i spremanje podataka u volumene. Docker daemon je ključan deo Docker platforme i ključan za njeno efikasno funkcionisanje [1].

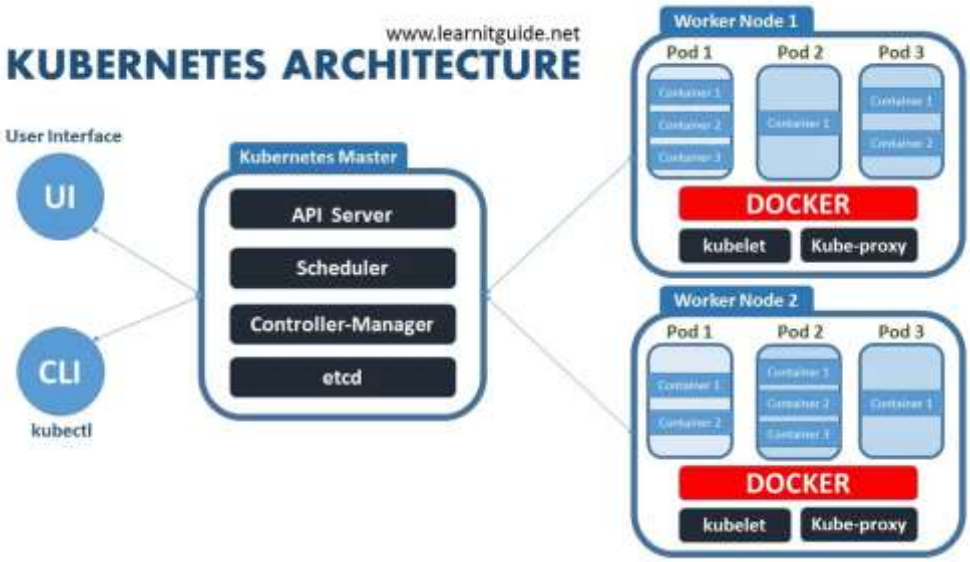
Dockerfile je tekstualni dokument koji sadrži skriptu za automatsko generisanje Docker slike. Ovaj dokument definše sve potrebne korake za instalaciju softvera, postavljanje okruženja i konfigurisanje aplikacija. Dockerfile je važan alat za automatizaciju procesa izgradnje i distribucije aplikacija u kontejneriziranom okruženju [1].

2.2 Kubernetes

Kubernetes je besplatan i open-source sistem za automatizaciju upravljanja kontejnerima. Omogućava programerima da automatizuju raspoređivanje, skaliranje i upravljanje aplikacijama u kontejnerima. Sistem je razvijen od strane Google-a i danas se razvija u okviru Cloud Native Computing Foundation (CNCF) [2].

Kubernetes omogućava automatsko upravljanje aplikacijama upotrebom kontejnera, što omogućava laku i brzu primenu, ali i veliku fleksibilnost u radu sa aplikacijama. Uz pomoć Kubernetes-a, moguće je izvršiti automatsko raspoređivanje kontejnera, balansiranje opterećenja, automatsko skaliranje, ažuriranje i povlačenje kontejnera, upravljanje saobraćajem, sigurnost i praćenje stanja aplikacija [2].

Kubernetes arhitektura (prikazana na slici 2) sastoji se od Master i Worker čvorova. Master čvor upravlja kontrolnom ravni, što podrazumeva upravljanje procesima, raspoređivanje aplikacija, skaliranje i povezivanje sa drugim komponentama sistema, dok Worker čvorovi izvršavaju aplikacije i obezbeđuju resurse potrebne za njihovo izvršavanje. Kubernetes arhitektura ima nekoliko glavnih komponenti, uključujući Kubernetes API server, etcd, kubelet, kubeproxy, Kubernetes kontrolne panele i Kubernetes resurse [2].



Slika 2 – Kubernetes arhitektura

Jedna od ključnih karakteristika Kubernetes-a je mogućnost deklarativnog upravljanja. To znači da programer definiše željeno stanje aplikacije u Kubernetes manifest datoteci, a zatim Kubernetes sistem garantuje da će aplikacija uvek raditi u skladu sa definisanim stanjem. Ovo omogućava efikasnije upravljanje aplikacijama, kao i lakši razvoj i testiranje aplikacija [2].

Kubernetes takođe podržava veliki broj mrežnih i skladišnih sistema, što ga čini idealnim za različite primene. Na primer, moguće je koristiti Kubernetes za upravljanje bazama podataka

u kontejnerima, što značajno olakšava upravljanje složenim bazama podataka u cloud okruženju [2].

2.3 TensorFlow i Keras

TensorFlow je popularna open-source biblioteka za računarsku obradu podataka i mašinsko učenje. Razvijen je od strane Google-a i dostupan je na velikom broju platformi, uključujući Windows, Linux i macOS. TensorFlow se najčešće koristi za razvoj dubokih neuronskih mreža i primenu u oblastima kao što su prepoznavanje govora, prepoznavanje slika, prirodni jezik obrade i drugo [3].

Keras je open-source biblioteka visokog nivoa, napisana u Python-u, koja služi kao interfejs za TensorFlow. Keras omogućava programerima da brzo i lako razvijaju neuronske mreže i druge modele mašinskog učenja, bez potrebe za detaljnim znanjem o matematičkim konceptima koji stoje iza ovih modela. Keras takođe omogućava programerima da brzo eksperimentišu sa različitim modelima i hiperparametrima, što omogućava brži i efikasniji razvoj [3].

TensorFlow i Keras su zajedno postali popularan izbor za razvoj modela mašinskog učenja, zbog njihove efikasnosti, fleksibilnosti i velike zajednice razvijalaca koja ih podržava. Ove biblioteke su takođe pogodne za paralelno izvršavanje algoritama mašinskog učenja, što ih čini idealnim za primenu u Docker i Kubernetes okruženju.

TensorFlow i Keras omogućavaju razvoj različitih modela mašinskog učenja, uključujući neuronske mreže sa konvolucionim slojevima, rekurentne neuronske mreže, generativne modele, i druge. Ove modele je moguće koristiti u različitim primenama, uključujući prepoznavanje slika i govora, predviđanje vrednosti, i drugo [3].

2.4 NFS

NFS (Network File System) je standardni protokol za deljenje datoteka preko mreže. NFS omogućava klijentima da pristupe datotekama na udaljenim serverima kao da se nalaze na lokalnom računaru. NFS je jednostavan za korišćenje i često se koristi za deljenje datoteka između servera i klijenata u okruženjima sa više računara.

NFS je dizajniran da bude nezavisan od platforme i može se koristiti na različitim operativnim sistemima, uključujući UNIX, Linux i Windows. Ovaj protokol se zasniva na klijent-server modelu, gde server obezbeđuje datoteke koje se mogu pristupiti preko mreže, dok klijenti pristupaju tim datotekama preko mreže.

NFS koristi RPC (Remote Procedure Call) za komunikaciju između klijenta i servera. RPC je protokol koji omogućava klijentima da pozovu funkcije koje se izvršavaju na serveru, i da dobiju odgovor nazad preko mreže. NFS koristi RPC za prenos zahteva za pristup datotekama i za slanje odgovora sa sadržajem datoteka nazad klijentima.

3 PREGLED IMPLEMENTIRANOG SISTEMA

3.1 Fizičko okruženje

Za implementaciju sistema na raspolaganje su date tri Ubuntu 22.04 LTS virtuelne mašine u okviru Elektronskog fakulteta. Specifikacija mašina je sledeća:

- Operativni sistem: Ubuntu 22.04 LTS
- Memorija: 8GB + 20GB dodeljena particija
- Radna memorija: 4GB
- Procesor: 2 jezgra Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz

Mašinama je omogućen pristup preko SSH tunela. Adresni prostor mašina je 10.10.3.80-82, gde je prvoj mašini na adresi .80 dodeljen hostname *master-node*, dok su druge dve nazvane *worker1* i *worker2* respektivno. Nazivi su dati shodno ulogama mašina u okviru Kubernetes klastera.

3.2 Logička organizacija sistema.

Implemetacija je organizovana po klijent-server modelu, gde se klijentska aplikacija nalazi na lokalnom računaru, dok je serverska implementacija u okviru udaljenog Kubernetes klastera. Pristup serverskoj strani omogućen je putem SSH tunela do mašine *worker1* koja je deklarisana kao pristupna tačka klasteru. Komunikacija klijent-server vrši se preko HTTP protokola.

Uloga klijenta je da na osnovu skupa test podataka koje poseduje formira HTTP zahteve za predikciju i prosledi ih serveru na izvršenje. Server prima ove zahteve, izvršava testiranje korišćenjem već istreniranog Keras modela i vraća odgovor klijentu. Keras model skladišti se u okviru NFS servera koji se nalazi na *master-node* mašini i dostupan je svim čvorovima u okviru klastera.

3.3 Serverska aplikacija

Kompletan kod serverske aplikacije dat je na slici 3.

```
app = Flask(__name__)

if __name__ != '__main__':
    gunicorn_logger = logging.getLogger('gunicorn.error')
    app.logger.handlers = gunicorn_logger.handlers
    app.logger.setLevel(gunicorn_logger.level)

model_path = os.environ.get("MODEL_PATH", "/app/model")
model = None
```

```
def load_model():
    global model
    if model is None:
        model = tf.keras.models.load_model(model_path)
    return model

@app.route("/test", methods=["POST"])
def test_model():
    model = load_model()
    csv = request.get_json()
    array = json.loads(csv)
    data = np.array(array, np.float64)
    result = model.predict(data)
    return str(result.tolist())
```

Slika 3 – Kod serverske aplikacije

Serverska aplikacija implementirana je kao Flask aplikacija koja se pokreće u okviru Gunicorn web servera. Pro pokretanju definišu se podešavanja za ispisivanje log poruka u okivur Gunicorn servera. Aplikacija ima jedan endpoint „*/test*“ čijim pozivom se inicira testiranje podataka nad istreniranim Keras modelom. Putanja do skladištenog modela definiše se pomoću promenljive okruženja *MODEL_PATH* sa podrazumevanom vrednošću „/app/model“ ukoliko nije podešena. U okviru Flask aplikacije nisu implementirane sesije, te se učitavanje modela vrši pri prvom pozivu testiranja.

3.4 Klijentska aplikacija

Kod klijentske aplikacije dat je na slici 4.

```
chunk_size = 10000
data_chunks = pd.read_csv(testfile_path,chunksize=chunk_size)
logging.basicConfig(level=logging.INFO)

requests=[]
for i, chunk in enumerate(data_chunks):
    logging.info(f"Forming request for chunk {i+1}...")
    test = chunk.drop([x for x in chunk.columns if 'Label_' in x], axis=1)
    json_data = test.to_json(orient='values')
    requests.append(grequests.post(test_url, json=json_data))

for resp in grequests.imap(requests, size=10):
    logging.info(resp)
```

Slika 4 – Kod klijentske aplikacije

Ovaj kod priprema podatke za testiranje modela mašinskog učenja i šalje ih u formi asinhronih zahteva ka određenom URL-u (u ovom slučaju „http://localhost:80/test“). Za slanje zahteva se koristi biblioteka grequests, koja omogućava slanje asinhronih zahteva velikim brzinama.

Ulazni podaci za testiranje modela su čitani iz CSV fajla, „testset.csv“, koji se nalazi na putanji „/home/kali/RUO/data/test/“, koristeći Pandas biblioteku za rad sa podacima. Ulazni podaci se čitaju po delovima čija je veličina definisana promenljivom *chunk_size*, što je korisno u slučajevima kada je ulazni CSV fajl veoma veliki.

Nakon čitanja i podele ulaznih podataka na delove, za svaki chunk podataka se formira zahtev u JSON formatu, koji se zatim dodaje u listu zahteva. JSON format podataka se formira pozivom *to_json* metode nad ulaznim podacima, što se koristi za konverziju Pandas DataFrame objekta u JSON format.

Nakon što su svi zahtevi formirani, koristi se *grequests.imap* metoda za asinhrono slanje zahteva u zadatoj veličini bloka (u ovom slučaju *size*=10). Ova metoda vraća generator koji omogućava čitanje odgovora na zahtev u redosledu u kojem su zahtevi poslati, što omogućava efikasno upravljanje velikim brojem zahteva.

Za upravljanje logovima korišćena je Python-ova ugrađena biblioteka za logovanje.

3.5 Docker slika

Nakon implementacije klijenta i servera izvršena je kontejnerizacija servera u Docker sliku. Sadržaj Dockerfile-a koji se koristi za formiranje slike prikazan je na slici 5.

```
FROM tensorflow/tensorflow

WORKDIR /app

COPY server.py .
COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

CMD ["gunicorn", "-w 4", "-b", "0.0.0.0:8000", "--log-level", "debug",
"server:app"]
```

Slika 5 – serverski Dockerfile

Kao osnova za izgradnju nove Docker slike koristi se *tensorflow/tensorflow* slika koja već ima instaliran TensorFlow. Nakon toga instaliraju se potrebne python biblioteke potrebne za pravilno funkcionisanje servera. Na kraju pokreće se Gunicorn server sa 4 worker-a, koji sluša zahteve na portu 8000.

Izgrađena Docker slika objavljena je javno na DockerHub-u pod nazivom *lazar010/ruo*.

3.6 NFS

Na *master-node* čvoru podignut je NFS server čija svrha je deljenje skladištenog istreniranog Keras modela. Podešavanja za deljenje prikazana su na slici 6.

```
/var/nfs/public *(rw,insecure,sync,no_subtree_check)
```

Slika 6 – sadržaj /etc/exports fajla na *master-node* mašini

Folder /var/nfs/public podeljen je svim čvorovima koji imaju pristup *master-node* mašini. U okviru ovog foldera skladišti se istrenirani Keras model. Čvorovima su date read i write privilegije i omogućena je komunikacija preko porta 2049 koji koristi NFS protokol (insecure fleg). Menjanje modela može se izvršiti montiranjem foldera i ažuriranjem njegovog sadržaja na klijentskom filesystem-u koji ima pristup *master-node* mašini preko SSH tunela. Ostali čvorovi u okviru klastera konfigurisani su u NFS klijent režimu i imaju pristup pomenutom folderu.

3.7 Kubernetes klaster

Kubernetes konfiguracija sastoji se iz dva dela: kofiguracija Deployment-a i konfiguracija Service-a.

Deployment objekat definiše korišćenje 2 replike serverske aplikacije. Specifikacija template-a sadrži definiciju volumena „modelpd“ koji je tipa „nfs“. Ovaj volume će omogućiti deljenje modela između replika. Takođe, specifikacija template-a uključuje definiciju kontejnera "server" koji koristi Docker sliku "lazar010/ruo". Kontejner je definisan sa memorijskim zahtevom od 2GB (sama slika zauzima 1,55GB) i limitiran na maksimalno zauzeće od 5GB. Limiti u pogledu performansi nisu definisani. Kontejner je mapiran na port 8000, a promenljiva okruženja *MODEL_PATH* je podešena na „/app/model“ putanju. Takođe, konfigurisano je mountovanje volume-a „modelpd“ na putanju „/app/model“.

Korišćenjem *kubectl* alata za konfiguraciju u kontrolu Kubernetes klastera, omogućeno je horizontalno autoskaliranje. Minimum replika postavljen je na 2, dok je maksimum postavljen na 10 replika.

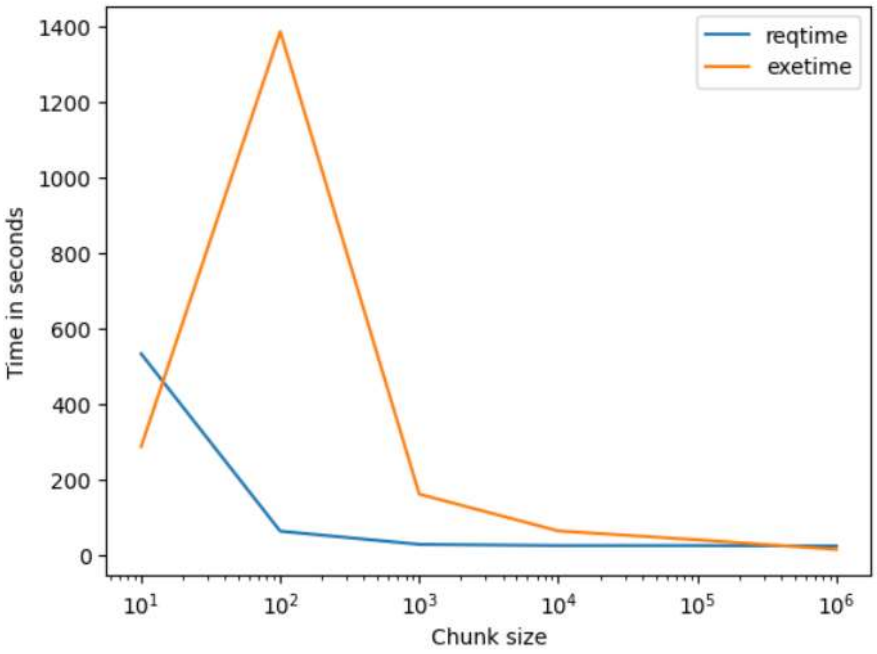
Service objekat definiše LoadBalancer tip usluge sa spoljnom IP adresom 10.10.3.81 (adresa *worker1* mašine) i mapira port 8000 na targetPort 8000 u Deployment objektu. NodePort je postavljen na 30001.

Ovakva konfiguracija omogućava pristup serveru unutar klastera preko adrese čvora *worker1* i otkrivenog porta 30001. Pristup klasteru može se izvršiti otvaranjem SSH tunela do mašine *worker1*.

4 PERFORMANSE IMPLEMENTIRANOG SISTEMA

Klijentski deo implementiranog sistema može se koristiti samostalno, ali se može i inkorporirati u postojeći kod u okviru kojeg se razvija sistem za mašinsko učenje. Takodje može se i montirati deljeni nfs folder na lokalnoj mašini radi izmene i zamene modela koji se testira. Zavisnosti klijenta su biblioteke *Pandas* i *grequests*.

Performanse aplikacije testirane su na već istreniranom Keras modelu i testnom dataset-u koji sadrži preko 500.000 redova. Test scenario podrazumeva deljenje test podataka, formiranje zahteva različitih veličina i merenje performansi u zavisnosti od veličine zahteva. Za svaku veličinu zahteva, ceo dataset je testiran po pet puta i kao vrednost vremena izvršenja uzeta je prosečna vrednost trajanja izvršenja. Praćeno je vreme potrebno za formiranje zahteva, kao i vreme od slanja zahteva do dobijanja svih rezultata. Grafik sa slike 7 prikazuje rezultate testiranja.



Slika 7 – test rezultati

Može se primetiti smanjenje trajanja testiranja sa povećanjem broja torki čije se testiranje jednovremeno zahteva. Takođe, sa manjim brojem zahteva smanjuje se i vreme potrebno za formiranje istih. Najmanja vrednost, odnosno najbrže vreme izvršenja postiže se kada se ceo dataset obuhvati jednim zahtevom.

5 ZAKLJUČAK

Cilj ovog seminarskog rada bila je implementacija Docker/Kubernetes okruženja za distribuirano izvršavanje testiranja algoritma mašinskog učenja. Za testiranje korišćen je istrenirani Keras model i dataset sa preko 500.000 unosa. Implementirana je klijentska i serverska aplikacija. Serverska aplikacija upakovana je u Docker kontejner i pripremljena za izvršavanje u okviru Kubernetes klastera. Na fakultetskoj cloud mreži formiran je Kubernetes klaster sa jednim *master* i dva *worker* čvora. Odrađen je deployment serverske aplikacije i testiranje njenih performansi.

Iz prethodnog poglavlja može se zaključiti da dobijeni rezultati nisu ono što je od ovakvog sistema očekivano, bar ne na ovom nivou zahteva. Performanse testiranja prilikom većih zahteva skroz su zadovoljavajuće, ali ne prikazuju prednosti distribuiranja unurar klastera. Razlog tome može biti razmera između opterećenosti sistema prilikom izvršavanja testa i overhead-a koji unosi mrežna komunikacija. Potencijalno se prednosti mogu uočiti prilikom testiranja nad znatno većim skupovima podataka, gde bi opterećenje worker-a bilo znatno veće. Takođe, bitno je naznačiti da je testiranje vršeno po scenariju gde se klasteru pristupa preko SSH tunela. Dodatna testiranja i podešavanja performansi potencijalni su predmet za dalji rad.

Nakon uvoda, u drugom poglavlju data je teoretska osnova za implementiranje predloženog sistema. Diskutovana su Docker i Kubernetes okruženja, njihov princip rada i način konfiguracije. Takođe, bilo je reči o TensorFlow i Keras bibliotekama, kao i o NFS protokolu. U trećem poglavlju detaljno je opisan implementirani sistem. U četvrtom poglavlju predočene su performanse implementiranog sistema.

6 LITERATURA

[1] FOUDA, Engy. A Complete Guide to Docker for Operations and Development.
[2] BURNS, Brendan, et al. Kubernetes: up and running. " O'Reilly Media, Inc.", 2022.
[3] GULLI, Antonio; PAL, Sujit. Deep learning with Keras. Packt Publishing Ltd, 2017.
[4] NOWICKI, Bill. Nfs: Network file system protocol specification. No. rfc1094. 1989.