Emulator autonomnog vozila u oblaku

Mihailo Žarković, Stefan Stefanović, Goran Ferenc
Syrmia d.o.o.
Beograd, Srbija
mihailo.zarkovic@syrmia.com, stefan.stefanovic@syrmia.com, goran.ferenc@syrmia.com

Sažetak — **Razvoj i testiranje na udaljenim računarima ne podrazumevaju posedovanje ciljanog hardvera i višestruko štede vreme i resurse za prikupljanje test primera. Platforma koja je tema rada nudi automatsko testiranje algoritama vezanih za autonomna vozila. Osnovna funkcionalnost takvih algoritama je obrada podataka sa internih senzora ili uređaja poput lidara, kamere ili radara. Korisnik ima mogućnost da otpremi svoju aplikaciju, posmatra tok testiranja i dobije izveštaj o testiranju. Aplikacije se izvršavaju na specijalizovanom hardveru, dizajniranom za upotrebu u pomenutoj industriji. Platforma korisnika ne ograničava u izboru tehnologije koju će koristiti.**

Ključne reči – Podsistem orijentisan ka klijentu, Podsistem za izvršavanje aplikacija, Podsistem za podatke, Software development kit (SDK), Redis, Docker, Kontejner, HIL, Automotive, Autonomous Driving

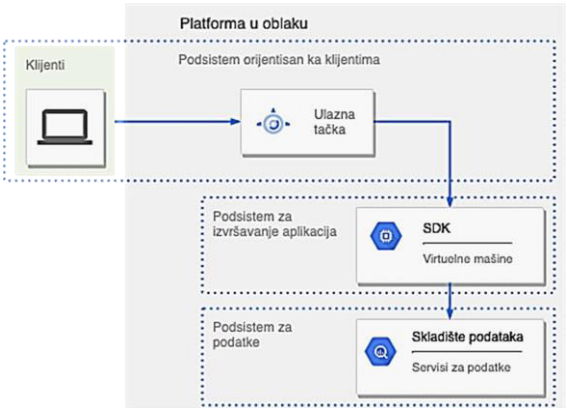
I. UVOD

Razvoj današnjih vozila ide u smeru potpune ili delimične autonomije u vožnji. Postoji više pravaca razvoja, a za sve je zajedničko oslanjanje na senzore koji sakupljaju informacije o okruženju. Kombinovanjem podataka sa senzora, kamere i lidara vozilo može na sigurniji i optimalniji način da prilagodi svoje kretanje uslovima na putu. Detekcijom rupa na putu moguće je prilagotiti sistem oslanjanja i time omogućiti udobniju i bezbedniju vožnju. U uslovima smanjene vidljivosti lidar se može koristiti za detekciju prepreka što podiže stepen bezbednosti učesnika u saobraćaju. S obzirom da proizvođači žele da postignu što viši nivo autonomije, proizvedeni sistemi postaju sve složeniji, samim tim je neophodno da se sistem ponaša adekvatno u svim mogućim situacijama. Neželjeno ponašanje ovakvih vozila može ugroziti živote svih učesnika u saobraćaju i zbog toga se posebna pažnja mora povesti pri testiranju i verifikaciji [1].

Za potrebe ove vrste testiranja postoji više pristupa [2]. Jedan pristup je virtuelna simulacija okruženja gde su svi senzori, dinamički model vozila, kontroler vozila i vozač virtuelni i simulirani kako bi što bliže interpretirali realnu sliku okruženja. Drugi pristup je vožnja u realnom saobraćaju. Cilj testiranja u virtuelnom okruženju je što ranije otkrivanje situacija u kojima se vozilo neželjeno ponaša i podizanje nivoa paralelizacije u proizvodnji. Pored temeljnog testiranja u simuliranom okruženju, neophodno je i temeljno testiranje u realnom saobraćaju. Platforma razvijena u ovom radu koristi virtuelni pristup sa zapisima stvarnih vožnji. Na bazi jednog realnog testa generiše se više testova u kojima su modifikovani snimci sa kamera. Pomoću dubokog učenja veštački se modifikuju vremenski uslovi na videima.

II. ARHITEKTURA PLATFORME

Arhitektura platforme je organizovana u više mikroservisa podeljenih po različitim celinama. Pojam mikroservisa [3] se odonosi na pristup u projektovanju i razvoju aplikacija koji favorizuje dekompoziciju aplikacije na veoma male funkcionalnosti. Svaka dekomponovana funkcionalnost se razvija potpuno nezavisno i ponaša se kao nezavisan deo sistema. S obzirom da je jedan mikroservis zadužen za vrlo usku funkcionalnost to omogućava da jedan deo tima razvija jedan servis bez preterane potrebe za koordinacijom sa ostatkom tima. Samim tim nije neophodna ni sinhronizacija oko programskog jezika ili tehnologije. Ovakav pristup razvojnom timu obezbeđuje visok stepen agilnosti. Još jedna, jednako bitna prednost ovakve organizacije je činjenica da se isti servis može ponovo upotrebiti u drugim aplikacijama koje zahtevaju istu funkcionalnost. Krajnji cilj je napraviti lanac servisa koji kao skup manjih nezavisnih funkcionalnosti čine celu aplikaciju.



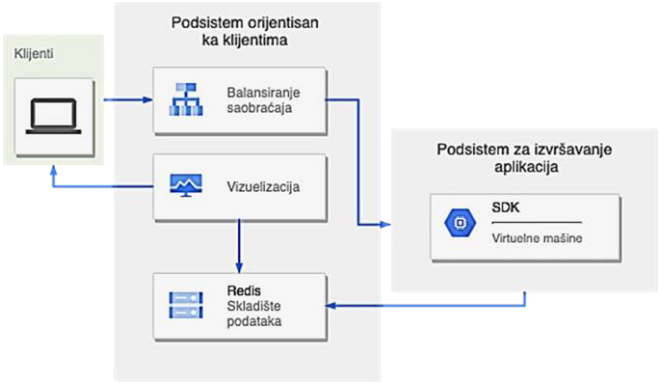
Slika 1. Arhitektura platforme

Sl. 1 prikazuje platformu koja je organizovana u tri celine:

- Podsistem orijentisan ka klijentima
  - Balansiranje saobraćaja
  - Vizuelizacija rezultata
- Podsistem za izvršavanje aplikacija
- Podsistem za podatke

III. PODSISTEM ORIJENTISAN KA KLIJENTIMA

Sl. 2 prikazuje deo platforme orijentisane ka klijentu koga čine servis zadužen za balansiranje saobraćaja (engl. *Cloud Load Balancing*) i servis za generisanje izveštaja i praćenje



Slika 2. Podsistem orijentisan ka klijentima

testiranja.

A. *Balansiranje saobraćaja*

Zadatak ovog servisa je maksimizacija protoka podataka kroz sistem i minimizacija proteklog vremena do dobijanja odgovora od platforme [4]. Postoje više različitih algoritama ove namene, ali se veći deo može klasifikovati u dve glavne kategorije. Statički algoritmi dele saobraćaj podjednako između istih servera dok dinamički algoritmi, po određenom kriterijumu, favorizuju neke servere. Iako su statički bazirani na jednostavnim pravilima otkriveno je da izazivaju veće opterećenje servera samim tim i neuravnotežen saobraćaj u sistemu [5]. S obzirom da platforma nudi test podatke sa različitih uređaja (senzori, kamere, lidari), a u korisničkim algoritmima se ne moraju svi uređaji iskoristiti što nam pruža šansu ka dinamičkom balansiranju zadataka. U podsistemu za samo izvršavanje algoritama postoji više servera različitih konfiguracija koji se upošljavaju u zavisnosti od složenosti samog algoritma. Algoritmi koji zahtevaju intenzivnu obradu podataka sa više uređaja biće dodeljivani najjačim serverima.

Pored hardverske podrške kreatori platforme moraju da obezbede i softverske pakete (engl. SDK - *Software Development Kit*) za različite tehnologije. SDK služi kao interfejs ka veštačkim podacima nalik na interfejse stvarnih uređaja. Pored upotrebe za dobijanje podataka SDK nudi interfejs za čuvanje različitih rezultata izvršavanja koji kasnije mogu biti vizuelizovani.

B. *Vizuelizacija rezultata*

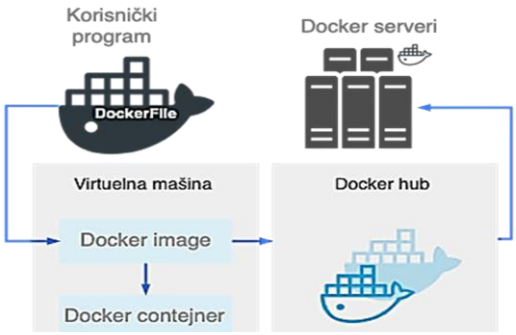
Ova funkcionalnost je projektovana kroz jedan servis zadužen za prikaz samih rezultata i *Redis* [6] skladišta podataka. Servis treba da pročita podatke iz *Redis*-a, jer se on ponaša kao keš memorija u ovakvoj arhitekturi i prikazuje ih korisniku. *Redis* je skladište otvorenog koda bazirano na čuvanju podataka u memoriji (engl. *in-memory database*). *In-memory* baze podataka se oslanjaju primarno na čuvanje u memoriji dok klasične baze čuvaju podatke na diskovima. Ovakve baze dizajnirane su da minimizuju vreme dobijanja odgovora eliminisanjem potrebe za razmenom podataka sa diskom. Većina korisnika *Redis* koristi za potrebe brzog skladištenja ili za skladištenje podataka koji će biti često korišćeni. S obzirom da su svi podaci smešteni u memoriji ovakav tip skladištenja može dovesti do gubitka podataka usled kvara ili prestanka rada servera, međutim *Redis* obezbeđuje mehanizam oporavka i tada učitava podatke sa diska prilikom ponovnog pokretanja.

Obezbeđeni SDK implicitno pruža mogućnost praćenja izvršavanja algoritama. Rezultati izvršavanja se preko internet interfejsa enkapsuliranog u SDK skladište na *Redis* i spremini su da ih servis prikaže korisniku.

IV. PODSISTEM ZA IZVRŠAVANJE APLIKACIJA

Platforma je sposobna da izvršava sve vrste programskih jezika i tehnologija pakujući aplikacije u kontejnere [7]. Pakovanjem u kontejnere dobijamo mogućnost izvršavanja više softverskih aplikacija na istoj mašini. Svaka pokrenuta instanca programa je nezavisna i naziva se kontejnerom. Kontejner je zatvoren iz ugla same aplikacije i obezbeđuje joj sve fajlove i biblioteke potrebne za ispravno izvršavanje. Sl. 3 prikazuje jedan od alata za virtuelizaciju i kontejnerizaciju, a to je *Docker* [8]. Ovaj alat je sposoban da virtuelizuje izvršavanje aplikacija koristeći *DockerFile* u kojem se nalaze instrukcije za pakovanje projekta u kontejner. Sposoban je da napravi i pokrene više virtuelnih instanci iste aplikacije. *Docker Image* se sastoji od kolekcije fajlova koje spajaju sve osnovne stvari potrebne za pokretanje i formiranje samog kontejnera, a to su instalacija, kod aplikacije i zahtevane zavisnosti. Jedan od načina za kreiranje *Docker Image* je pravljenjem gore pomenutog *DockerFile*-a. Dodatno *Docker Image* može biti indeksiran i sačuvan na *Docker*-ovom internet skladištu tako da se može koristiti i na drugim računarima.

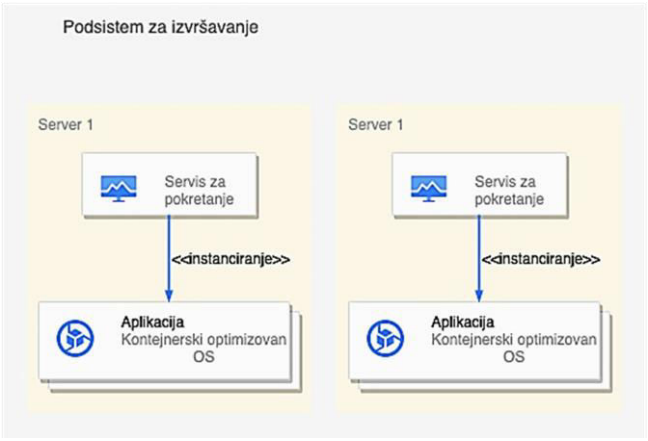
Ovaj podsistem se sastoji od više servera (Sl. 4) i na svakom je podignut po jedan servis čija je namena da instancira i pokrene kontejner. Podsistem za balansiranje saobraćaja upošljava jedan od servera tako što im šalje



Slika 3. Docker kontejnerizacija

podatke otpremljene na klijentskoj strani. Korisnik može otpremiti već napravljen *Docker Image* ili uz otpremljenu aplikaciju dostaviti sve zavisnosti i instrukcije za korišćenje

aplikacije što bi servisu za pokretanje sugerisalo da sam



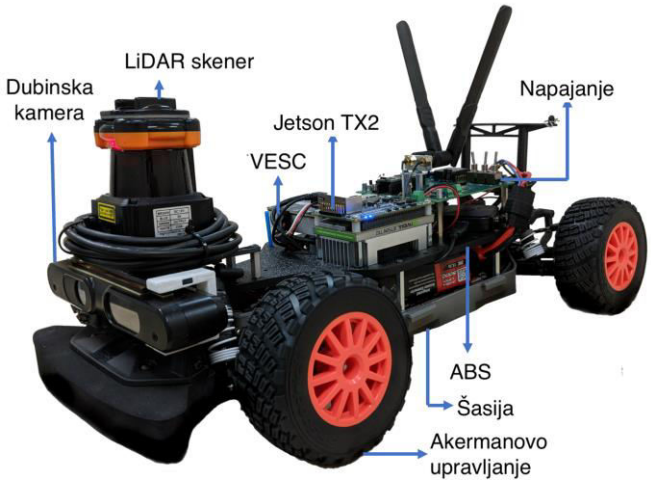
Slika 4. Podsistem za izvršavanje korisničkih aplikacija

formira *DockerFile*, instancira i pokrene kontejner.

Kao što je gore već pomenuto koristeći ovakav pristup programerima se ostavlja mogućnost da svoja rešenja implementiraju u svim tehnologijama i jednostavno ih testiraju na jednom mestu. Rezultate izvršavanja programa prikuplja podsistem za vizuelizaciju rezultata.

V. PODSISTEM ZA PODATKE

Za početak je potrebno prikupiti ili na neki način generisati podatke potrebne za testiranje. Za prikupljanje stvarnih zapisa



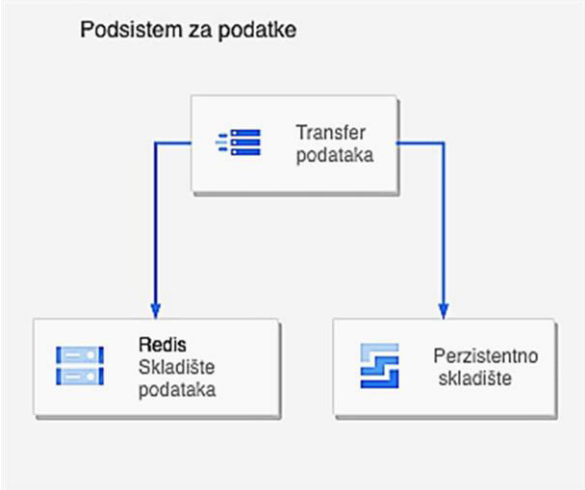
Slika 5. Platforma za prikupljanje podataka

okruženja moguće je koristiti uređaj poput uređaja sa Sl. 5 [9].

Na uređaju se nalaze lidar, stereo kamera, senzor pozicije točkova itd. Kretanjem uređaja formiraju se zapisi sa uredjaja i senzora koji će na kraju biti skladišteni u bazu podataka. Da bi se broj testova povećao moguće je i veštački modifikovati snimke sa kamere. Jedno od rešenja ovog problema su nenadgledane mreže koje vrše transformacije slika [10]. Svi dobijeni podaci sa uređaja su vremenski sinhronizovani i na isti način skladišteni u bazu. Bitna je vremenska sinhronizacija, jer će samo na taj način algoritam stvoriti adekvatnu virtuelnu sliku okruženja u kojem se uređaj nalazio dok je snimao.

Sistem je projektovan kao HIL (engl. *Hardware-in-the-loop*) model. HIL predstavlja apstrakciju željenog uređaja koji je sposoban da generiše signale u realnom vremenu takve da simuliraju stvarno stanje okruženja. Skraćenje proizvodnog procesa, paralelni razvoj, testiranje i ograničenja fizičke prirode pri testiranju na realnim stvarima su neki od razloga zašto se ni jedan kompleksniji sistem ne može razvijati bez upotrebe HIL-a.

Arhitekturu ovog podsistema čini baza podataka sa senzora i uređaja, *Redis* i servis koji dovlači skup podataka potrebnih za testiranje iz perzistentnog skladšta u *Redis* (Sl. 6). Pre početka izvršavanja korisničkog programa servis treba da napuni *Redis* i time su se ispunili uslovi za korišćenje tog skladišta po principu HIL-a. Kao što je gore pomenuto, vodeća osobina *Redis*-a je brz upis i čitanje podataka što



Slika 6. Podsistem za podatke

stvara privid da se izvršavanjem korisničkih aplikacija podaci čitaju sa pravih senzora u realnom vremenu.

VI. ZAKLJUČAK

U radu je predložena arhitektura platforme u oblaku za automatizovano testiranje softvera korišćenih za razvoj autonomnih vozila. Automatsko testiranje značajno umanjuje napor potreban za testiranje, povećava broj izvršenih testova i daje mogućnost bržeg otkrivanja situacija u kojima se softver ne ponaša adekvatno. Ipak, ne može se očekivati da će automatski testovi u potpunosti zameniti manuelne. Zbog toga se završna faza testiranja autonomnih vozila sprovodi u stvarnom okruženju, samim tim automatsko testiranje ne treba gledati kao zamenu, već kao dopunu manuelnog testiranja.

Imajući u vidu da proizvođači koriste različite tehnologije za izradu softvera, platforma tehnikom virtuelizacije nudi nezavisno testiranje svake od njih. Pored različitih tehnologija za izradu softvera, rešenja proizvođača se razlikuju i po korišćenju senzora. U zavisnosti od upotrebe senzora platforma obezbeđuje različite pakete test podataka, oni koji se oslanjaju samo na kemere dobiće slike kao test podatke, dok će oni koji koriste više različitih senzora dobiti željenu fuziju podataka kao test primer.

ZAHVALNICA

Autori zahvaljuju Fondu za inovacionu delatnost Republike Srbije koji je delom omogućio izradu ovog rada u okviru projekta broj 50301 pod nazivom "Platform for Remote development of Autonomous Driving algorithms in realistic environment - READ".

LITERATURA

1. Francisca Rosique, Pedro J. Navarro, Carlos Fernandez, Antonio Padilla, "A systematic review of perception system and simulators for autonomous vehicle research", Februar 2019.
2. WuLing Huang, Kunfeng Wang, Yisheng Lv, FengHua Zhu, "Autonomus vehicle testing methods review", 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil
3. Antonios Makris, Konstantinos Tserpes, Theodora Varvarigou, "Transition from monolithic to microservice-based applications. Challenges from the developer perspective", Februar 2022.
4. Zenon Chaczko, Venkatesh Mahadevan, Shahrzad Aslanzadeh, Cristopher Mcdermid, "Availability and Load Balancing in Cloud Computing", Internationl Conference on Computer and Software Modeling IPCSIT vol.14 (2011).
5. Payal Beniwal, Atul Garg, " A comparative study of static and dynamic load balancing algorithms" , 2014.
6. Raj Patel, "Data + Education. Redis is a cache or more?", June 2021.
7. Jay Shah, Dushyant Dubaria, "Building modern clouds: using Docker, Kubernets & Google cloud platform", 2019.
8. Thomas Uphill, Arundel Khare, Saito, Lee, Carlos Hsu, "DevOps; Puppet, Docker and Kubernets- Learning path", Packt Publications, First Edition, 2017.
9. Matthew O'Kelly, Houssam Abbas, Jack Harkins, Cris Kao, Tash Vardhan Pant, Raul Mangharam, Varundev Suresh Babu, Dipshil Agarwal, Madhur Behl, Paolo Burgio, Marko Bertogna, "F1/10: An open-source autonomous cyber-physical platform", Jan 2019.
10. Ming-Yu Liu, Thomas Breuel, Jan Kautz, "Unsupervised Image-to-Image translation networks", Jul 2018.

ABSTRACT

Development and testing on remote computers do not require the possession of target hardware and save, on multiple levels, time and resources for collecting test cases. The platform that is the subject of the paper offers automatic testing of algorithms related to autonomous vehicles. The basic functionality of such algorithms is processing data from internal sensors or devices such as lidar, camera or radar. The user has the ability to upload his application, observe the test process and get a test report. Applications run on specialized hardware, designed for use in the aforementioned industry. The platform does not limit users in choosing the technology they will use.

AN AUTONOMOUS VEHICLE EMULATOR IN THE CLOUD

Mihailo Žarković, Stefan Stefanović, Goran Ferenc