OpenBalancer –Raspoređivač opterećenja za OpenStack platformu

Aleksandar Vukotić
Institut RT-RK
Banja Luka, RS, BiH
aleksandar.vukotic@rt-rk.com

Mihajlo Savić
Elektrotehnički fakultet
Univerzitet u Banjoj Luci
Banja Luka, RS, BiH
mihajlo.savic@etfbl.net

Sažetak - **Kao rezultat upotrebe *cloud computing* usluga nastala su brojna komercijalna i *open source* rješenja za upravljanje cloud sistemima. Jedno od tih rješenja je i OpenStack *open source* platforma namijenjena za sve vrste *cloud* sistema. Ova platforma iz dana u dan postaje popularnija i neki smatraju da predstavlja budućnost *cloud computing*-a. O ozbiljnosti ovog projekta govori broj svjetskih kompanija i broj članova *open source* zajednice koji je uključen u njen razvoj. OpenStack je modularna platforma i sastoji se od nekoliko projekata. Cijena fleksibilnosti ove platforme je kompleksna instalacija. Vremenom su razvijeni alati čime je proces instalacije ove platforme automatizovan. Interakcija korisnika sa sistemom je omogućena putem *web* interfejsa, klijentskih biblioteka i CLI. Kao rezultat analize i istraživanja rada ove platforme u ovom radu je predloženo jedno rješenje za raspoređivanje opterećenja u OpenStack sistemu. Sistem nosi naziv OpenBalancer i realizovan je u Python programskom jeziku.**

Ključne riječi- Cloud computing, OpenStack, load-balancing, Python, RabbitMQ

I. UVOD

OpenStack predstavlja *open source* platformu koja služi za upravljanje IaaS baziranim *cloud computing* sistemima. Svrha OpenStack-a jeste da odgovara potrebama javnih i privatnih cloud sistema bez obzira na njihovu veličinu, tako što će biti jednostavna za upotrebu i masivno skaliranje.

Prva verzija ove platforme je izašla u julu 2010. godine. OpenStack je projekat započet od strane NASE i RackSpace-a, trenutno se održava od strane neprofitne OpenStack fondacije. Sastoji se od nekoliko slojeva i projekata koji zajedno daju *cloud* platformu. Vrlo je prilagodljiva platforma upravo iz razloga velike modularnosti. Trenutno postaje najozbiljniji *open source* projekat zahvaljujući širokoj podršci *open source* zajednice. Svakih 6 mjeseci izlazi nova verzija OpenStack-a. Prvo slovo naziva svake naredne verzije prati abecedu. Aktuelna verzija u trenutku pisanja ovog rada je Liberty. Svaka naredna verzija OpenStack-a izlazi sa novim komponentama sistema koje proširuju funkcionalnost platforme ili zamjenjuju već postojeće komponente zbog uočenih mana tokom korištenja. Platforma je jako modularna i trenutno u njen sastav ulazi 12 projekata. [1] OpenStack je u potpunosti napisan u Python programskom jeziku, sa malim procentom JavaScript koda. Distribuira se pod Apache v2.0 licencom.

Cilj ovog rada je opis OpenStack platforme, njenog funkcionisanja i izrada primjera mogućnosti proširenja funkcionalnosti upotrebom Python programskog jezika.

II. CLOUD COMPUTING I OPENSTACK

Cloud computing predstavlja model usluga koji omogućava korisnicima mrežni pristup dijeljenim računarskim resursima (npr. serveri, *storage*, aplikacije i servisi) na zahtjev uz minimalan napor i interakciju provajdera. [2] Sa razvojem interneta i *cloud computing* je dobijao sve veći značaj. Napredak na polju virtualizacije doveo je do toga da se usluge *cloud computing*-a komercijalizuju i počinju naplaćivati. Iako su tehnologija virtualizacije i razvoj interneta ujedno doveli do razvoja *cloud computing*-a, usluge koje *cloud computing* nudi nisu u velikoj mjeri značajne za pojedince kao korisnike, koliko zapravo za velike organizacije i kompanije kao korisnike. Danas u svijetu tehnologije gotovo je nezamislivo da kompanije u svom poslovanju nemaju i ne koriste računarsku infrastrukturu. Ta infrastruktura povlači za sobom dodatne troškove nabavke i održavanja računarske opreme koji mogu da značajno utiču na smanjenje budžeta kompanija. Koristeći usluge *cloud computing*-a kompanije se rješavaju troškova nabavke i održavanja, jer koriste gotovu infrastrukturu koju održavaju *cloud* provajderi. Kompanije plaćaju samo korištenje računarskih resursa. Pokazalo se da korištenje *cloud computing* usluga smanjuje troškove za 23%. [3] S obzirom da je kvalitet usluga koje *cloud computing* nudi na jako visokom nivou, a i na značajno smanjenje troškova, kompanije su se vrlo lako odlučivale na korištenje ovih usluga. *Cloud computing* sistemi se mogu organizovati na više načina. Postoji više modela organizacije *cloud computing* sistema, svaki sa svojim karakteristikama odgovara različitim potrebama korisnika te se u zavisnosti od njih koristi ih.

Svaki *cloud* provajder opslužuje određene funkcije i potrebe korisnika, dajući tako korisnicima više ili manje kontrole u zavisnosti od tipa usluge. Prilikom biranja *cloud* provajdera, potrebno je uporediti potrebe koje je potrebno opslužiti sa uslugama koje *cloud* provajder nudi. Potrebe za različitim uslugama će varirati u zavisnosti od toga koliko resursa je potrebno. Ukoliko je riječ npr. o ličnim potrebama korisnika onda je potrebno izabrati drugi tip *cloud*-a i provajdera u odnosu na slučaj kada je riječ o poslovnim potrebama. Usluge kod cloud provajdera se naplaćuju po

principu *pay-per-use*, tako da u bilo kojem trenutku korisnik može da zatraži više resursa ili da ih vrati. Postoje generalno tri modela *cloud* provajder usluga poznatih pod nazivom *cloud computing stack*: IaaS (infrastruktura kao usluga), PaaS (platforma kao usluga) i SaaS (softver kao usluga). [4] Pomoću OpenStack platforme vrši se upravljanje infrastrukturom IaaS sistema, gdje se pod infrastrukturom misli na servere, mreže, skladišta za podatke ukratko računarski resursi.

OpenStack je vrlo modularna i skalabilna platforma koja se sastoji iz nekoliko komponenata, zasebnih projekata. Svaka od tih komponenti je zadužena za upravljanje određenim dijelom resursa u sistemu. Korisnik može u zavisnosti od potrebe da ukloni ili doda određene komponente u zavisnosti od potrebe i na taj način u sistem ukloni ili uključi određenu funkcionalnost. Logički posmatrano, komponente OpenStack platforme se mogu podijeliti u tri zasebne grupe: upravljačka, mrežna i grupa za rad sa virtualnim mašinama. Upravljačka grupa izvršava API servise, *web* interfejs, bazu podataka i servis za proslijeđivanje poruka (AMQP servis). Mrežna grupa izvršava servise zadužene za upravljanje mrežom, koji podrazumjevaju konfiguraciju mreža, dodjelu IP adresa i rad sa VLAN-ovima u okviru *cloud* sistema, dok grupa za rad sa virtualnim mašinama predstavlja sloj za rad sa hipervizorom.

Svi projekti OpenStack-a koriste bazu podataka i AMQP servis. U bazi podataka čuvaju se informacije o stanju sistema i njegovih pojedinih komponenti. Neke komponente imaju još dodatno svoje baze podataka za pohranjivanje podataka. AMQP predstavlja *messaging* protokol u OpenStack *cloud* sistemu. Interni procesi OpenStack komponenti kao što su Nova, Cinder, Neutron koriste ovaj protokol za međusobnu komunikaciju. Moguće je koristiti Qpid, ActiveMQ ili RabbitMQ brokerske servise koje implementiraju AMQP protokol. Na AMQP servis se može gledati kao na brokera za poruke koji jednostavno prima i proslijeđuje poruke. Poruke se proslijeđuju do procesa koji su registrovani za primanje poruka. Putem posebnih adapterskih klasa poruke se mapiraju u pozive odgovarajućih funkcija u ovim procesima. Rezultati poziva se takođe vraćaju preko AMQP putem poruka. Na ovaj način ostvaren je RPC u OpenStack-u. RPC (*eng. Remote Procedure Call*) se jednostavno implementira putem asinhronih poruka koje se šalju preko ovog servisa. Na ovaj način nema blokiranja između komponenti OpenStack sistema i njihovih dijelova. Preko ovog servisa komponente šalju svoj status koji se čuva u glavnoj OpenStack bazi. Za bazu podataka koriste se MySQL, MariaDB ili PostgreSQL. [5] U manjim Openstack sistemima, baza podataka i AMQP servisi se obično nalaze na kontrolnom hostu (*eng. controller*). Zbog horizontalne skalabilnosti sistema poželjno je da se svaka grupa nalazi na posebnom hostu ili čak na više njih, ukoliko je potreban *failover* servisa.

Za svaki OpenStack projekat postoji kodni naziv. Trenutno aktuelni projekti za posljednu verziju su:

- *Horizon (Dashboard)* - predstavlja *web* bazirani interfejs pomoću kojeg korisnik može na vrlo jednostavan način da koristi ostale komponente cloud sistema
- *Keystone (Identity)* - komponenta zadužena za autentifikaciju i autorizaciju korisnika za rad sa ostalim komponentama
- *Glance (Image)* - predstavlja katalog i repozitorijum slika pomoću kojih se pokreću virtualne mašine
- *Nova (Compute)* - komponenta zadužena za rad sa hipervizorom i virtualnim instancama
- *Cinder (Block Storage)* - komponenta zadužena za upravljanje perzistentnim prostorom za skladištenje podataka za virtualne mašine
- *Swift (Object Storage)* - komponenta zadužena za čuvanje i korištenje objekata
- *Neutron (Networking)* - komponenta zadužena za upravljenje i rad sa mrežama u *cloud* sistemu
- *Ceilometer (Telemetry)* - služi za monitoring korištenja *cloud* servisa i određivanje cijene za naplatu troškova. Ova komponenta se takođe koristi za prikupljanje statističkih podataka u zavisnosti od potrošnje.
- *Heat (Orchestration)* - služi za rad sa *cloud* aplikacijama kroz Openstack REST API i CloudFormation Query API
- *Trove (Database)* - predstavlja *Database as a Service*. Olakšava rad sa relacionim bazama u *cloud* sistemu.

Svaki projekat komunicira sa ostatkom *cloud* sistema pomoću RESTful API-ja. Ukoliko neki projekat zatreba resurse od drugog projekta, potrebno je da mu pošalje odgovarajući REST zahtjev putem API-ja. [6]

OpenStack API predstavlja RESTful API-i putem kojeg se šalju korisnički zahjevi i upravlja *cloud* sistemom. REST zahtjevi predstavljaju zahtjeve koji koriste metode HTTP v1.1 protokola, koji se šalju API serveru. OpenStack Python SDK (*eng. Software Development Kit*) je interfejs prema OpenStack API-ju putem kojeg se mogu koristiti funkcionalnosti API-ja u Python programskom jeziku. Predstavlja set klijentskih biblioteka za svaki OpenStack projekat. SDK omogućava pisanje skripti koje mogu da upravljaju resursima i obavljaju neke automatizovane zadatke u *cloud* sistemu pozivanjem metoda odgovarajućih objekata. Svaki poziv metode nad objektom rezultuje odgovarajućim REST zahtjevom prema API serveru, te je moguće direktno uputiti REST zahtjev koji ima isti efekat. OpenStack API se može koristiti i putem CLI pomoću OpenStack klijenata. Svaki projekat u OpenStack-u ima svog CLI klijenta. Svi klijenti su implementirani koristeći OpenStack Python SDK. [7]

III. PROŠIRENJE FUNKCIONALNOSTI OPENSTACK PLATFORME - OPENBALANCER

Jednostavnija organizacija *cloud* sistema se sastoji od jednog kontrolerskog hosta na kojem se izvršavaju ključni OpenStack servisi i više *compute* hostova na kojima se pokreću instance. *Compute* host predstavlja hosta na kojem se nalazi hipervizor i nova-compute proces. Nova-compute predstavlja interni proces Nova projekta koji služi za rad sa hipervizorom. Na svakom hostu na kojem se nalazi hipervizor mora da postoji i nova-compute proces. Prilikom pokretanja instance preko kontrolne ploče (Dashboard), korisnik nema mogućnost izbora *compute* hosta na kojem će se instanca pokrenuti. Odluku na kojem *compute* hostu će instanca biti pokrenuta određuje nova-scheduler servis Nova komponente OpenStack sistema. Ovaj servis određuje optimalnog host pomoću filtriranja i na taj

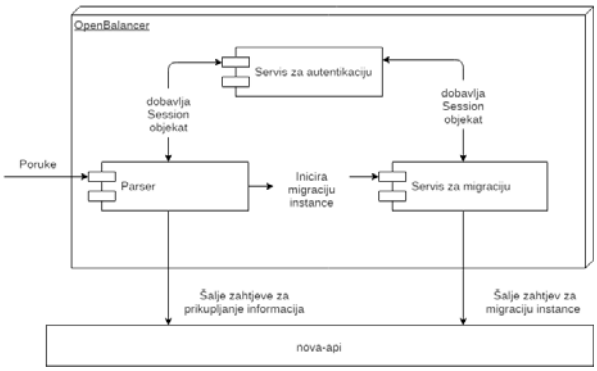
način raspoređuje opterećenje na hostovima u *cloud* sistemu prilikom pokretanja instanci. U slučaju da se instance ugase ili migriraju sa jednog *compute* hosta na drugi, vrlo je vjerovatno da će se balans opterećenja poremetiti. Ideja koja se nametnula za realizaciju praktičnog rada je sistem koji bi detektovao ovaj poremećaj i izvršio migraciju instanci sa opterećenih hostova na manje opterećene hostove čime bi se popravio balans u sistemu.

Kao praktičan rad realizovan je sistem OpenBalancer za balansiranje opterećenja u OpenStack sistemu. Sistem je napisan u Python programskom jeziku koristeći klijentske biblioteke. Sistem detektuje stanje *cloud* sistema u kojima su neki compute hostovi opterećenjiji u odnosu na ostale compute hostove i započinje migraciju instanci na neopterećene hostove. Da bi ovaj sistem detektovao ovakve situacije on mora da u svakom trenutku ima informacije o stanju u kojem se cloud nalazi. Te informacije predstavljaju broj compute hostova koji se nalazi u sistemu, resurse kojima oni raspolažu, broj instanci na njima, resurse koje svaka instanca zauzima. Ove informacije mogu se prikupiti na dva načina:

- slanjem API zahtjeva putem klijentskih biblioteka [7]
- obrađivanjem poruka koje komponente šalju putem AMQP-a [8]

U ovom sistemu, za prikupljanje podataka korištene su obe metode jer svaka metoda ima svoje mane ukoliko se pojedinačno koristi. Ukoliko se za prikupljanje podataka koriste samo klijenstke biblioteke onda bi se API servis morao često kontaktirati iz razloga što aplikacija mora da poznaje stanje *cloud* sistema u gotovo svakom trenutku. Ovaj način je jako sličan *polling*-u i u većini odgovora server bi vratio iste podatke jer se stanje *cloud* sistema ne mijenja tako često. Prikupljanje podataka se može obaviti i putem AMQP kojeg koriste komponente OpenStack-a. Osim za RPC, komponente koriste ovaj servis i za slanje poruka o svom statusu. Bilo kakva promjena u *cloud* sistemu od strane korisnika preko *web* interfejsa ili klijentskih biblioteka rezultovaće slanjem poruke preko AMQP-a. Mana ovog načina prikupljanja informacija o stanju *cloud*-a je što je potreban određen period da bi se prikupile informacije i većinu relevantnih podataka nije moguće dobiti na ovaj način. Iz tog razloga korištena je kombinacija oba načina za prikupljanje informacija. U početku se svi podaci prikupe slanjem zahtjeva API serveru, a zatim se svaka promjena u *cloud* sistemu detektuje obrađivanjem poruka koje stižu preko AMQP-a.

Glavne komponente OpenBalancer sistema su: parser, servis za migraciju i servis za autentikaciju. Parser komponenta je zadužena za parsiranje poruka i izračunavanje opterećenja sistema, te čuvanje podataka o stanju sistema. Ti podaci uključuju broj radnih hostova, resurse sa kojima oni raspolažu, broj instanci, itd. Servis za migraciju je komponenta zadužena za migraciju instanci i čuvanje informacija o trenutno aktivnim migracijama. Servis za autentikaciju je komponenta zadužena za dobavljanje *Session* objekta sa odgovarajućim korisničkim atributima. Ovaj objekat se koristi za kreiranje klijentskog *Nova* objekta putem kojih se šalju zahtjevi *nova-api* servisu. Šema sistema je prikazana na Sl. 1.



Slika 1. Šema OpenBalancer sistema

A. *Funkcionisanje OpenBalancer sistema*

Poruke relevantne za rad aplikacije se šalju na *nova topic exchange* preko AMQP servisa. Analizirane su sve poruke koje prolaze kroz nova topic i poruke koje nose potrebne informacije šalju se sa sledećim ključevima za rutiranje: "notifications.info" i "conductor". Poruke sa ključem "notifications.info" nose informacije o instancama, dok poruke sa ključem "conductor" nose informacije o radnim hostovima i servisima. Svaki radni host izvršava na sebi *nova-compute* servis. Poruke o servisu predstavljaju podatke o *nova-compute* servisu i sadrže samo informaciju o tome da li je servis aktivan ili ne. Ove poruke svaki *nova-compute* servis šalje periodično.

Na početku rada aplikacija se povezuje na RabbitMQ servis. Kreira se poseban *queue* koji se veže na *nova topic*. Obrada poruka se izvršava pomoću *callback* funkcije koja se poziva svaki put kada stigne nova poruka. Poruke koje stižu će se serijski obrađivati, *callback* funkcija se neće pozvati dok se obrada prethodne poruke ne završi. Unutar ove metode poziva se metoda parserskog objekta za parsiranje poruka.

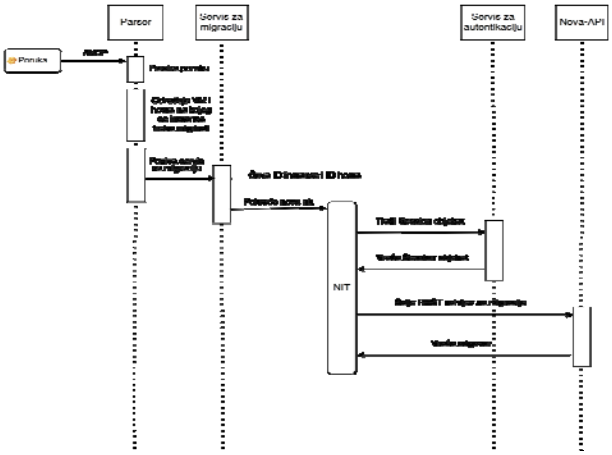
Parserska komponenta OpenBalancer sistema predstavlja objekat klase RabbitMQMessageService koja se nalazi u *rabbitmq_message_service.py* modulu. Ova klasa je zadužena za parsiranje poruka, čuvanje informacija o *cloud* sistemu i sadrži logiku za raspoređivanje opterećenja radnih hostova u sistemu. Prilikom pokretanja OpenBalancer sistema vrši se inicijalizacija parsera. Inicijalizacija podrazumijeva slanje zahtjeva *nova-api* servisu putem kojih se prikupljaju sve informacije o trenutnom stanju sistema. Naknadne promjene u sistemu se detektuju putem poruka. Radi osiguranja od nekonzistentnosti podataka kojima parser raspolaže, parser periodično pokreće inicijalizaciju za prikupljanje podataka. Pored klase RabbitMQMessageService u modulu *rabbitmq_message_service.py* se nalaze klase koje predstavljaju instancu (VMInstance), compute host (ComputeNode) i servis (Service). Informacije koje se prikupe čuvaju se kao atributi objekata ovih klasa.

Za računanje opterećenja hostova parserska komponenta koristi kolekciju filtara. Postoje filtri za radnu memoriju, broj virtuelnih procesora i lokalno skladište hosta. Filtri služe za određivanje opterećenosti hosta za svaki resurs, provjeru da li host ima dovoljno resursa za neku instancu, te određuju opterećenje koje instanca pravi na hostu. Ukupno opterećenje hosta se dobija kao suma opterećenja pojedinačnih resursa.

Opterećenost hosta za neki resurs se određuje kao procenat zauzetosti resursa pomnožen sa određenim faktorom težine. Faktori težina služe da se opterećenje određenih resursa prioritetizuje u sistemu. Vrijednosti ovih faktora se mogu postaviti u konfiguracionom fajlu aplikacije. Opterećenje za hostove se računa pri obradi podataka o hostu bilo da su ti podaci prikupljeni putem pristupa *nova-api* ili putem obrade poruka.

Servis za migraciju predstavlja objekat klase *MigrateService* koji se nalazi u modulu *migrate_service.py*. Ova klase služi za pokretanje migracija i čuvanje informacija o pokrenutim migracijama. Parserska komponenta koristi ovu klasu za migraciju instanci. Kada se pozove metoda za migraciju instance ova klasa čuva podatake o migraciji i pokreće nit zaduženu za slanje zahtjeva *nova-api* servisu. Nit kreira klijentski objekat *Nova* klijenta i šalje zahtjev za migraciju API serveru. Nova nit se kreira jer slanje REST zahtjeva i čekanje odgovora od API servera može da potraje i samim tim može da blokira OpenBalancer sistem.

U slučaju da je zahtjev odbijen od strane servera ili nije stigao do servera, neće doći do migracije i sačuvani podaci se brišu. Na slici Sl.2 je data dijagram sekvence za migraciju instance.



Slika 2. Sekvenca prilikom pokretanja migracije

B. *Algoritam za raspoređivanje opterećenja*

Algoritam koristi vrijednosti opterećenja hostova da bi odredio sa kojih hostova je potrebno migrirati instance. Prvi korak u algoritmu je određivanje vrijednosti aritmetičke sredine opterećenja u cijelom sistemu. Dobija se kao suma opterećenja na svim hostovima podijeljena sa brojem hostova. Na osnovu dobijene vrijednosti hostovi se organizuju u dvije liste. Host čija je vrijednost izračunatog opterećenja veća od vrijednosti aritmetičke sredine smatra se da je opterećen i dodaje se u listu opterećenih hostova, dok se u suprotnom slučaju dodaje u listu neopterećenih. Lista opterećenih hostova se sortira u opadajućem redoslijedu po težinama. Za hostove koji se nađu u listi neopterećenih se prilikom dodavanja u listu računa još jedna vrijednost opterećenja koja predstavlja opterećenje koje bi host imao da se instance koje se trenutno migriraju na njega, nalaze na njemu. Lista neopterećenih hostova se zatim sortira u rastućem redoslijedu u odnosu na vrijednost ove posebne težine.

Drugi korak algoritma koristi ove liste i prvo provjerava veličine ovih listi. U slučaju da je bilo koja od ovih lista prazna, algoritam se završava. U slučaju da liste nisu prazne, algoritam prelazi na određivanje hosta sa kojeg će se migrirati instance. Prolazi se kroz listu opterećenih hostova i uzima prvi host koji nema instance koje su u stanju migriranja. U slučaju da takvog hosta nema, algoritam se završava. Hostovi koji imaju migrirajuće instance se preskaču iz razloga što migracija instanci predstavlja proces koji traje određeni vremenski period. Migracija instance traje duže ukoliko instanca zauzima više resursa.

Nakon određivanja hosta sa kojeg će se migrirati instanca, lista instanci koje se nalaze na njemu se sortira u rastućem redoslijedu u odnosu na količinu radne memorije. Lista se sortira u rastućem redoslijedu iz razloga što migracija instanci sa manjim zauzećem radne memorije traje kraće. Za svaku instancu se prolazi kroz listu neopterećenih hostova. Instanca se uzima u obzir ukoliko je u aktivnom stanju i ukoliko je vrijeme posljednje migracije instance veće od definisane dozvoljene vrijednosti parametra koji je moguće odrediti kroz konfiguracioni fajl aplikacije. Pomoću filtara se provjerava da li host ima dovoljno resursa za instancu. Računa se opterećenje koje instanca ima na hostu na kojem se trenutno nalazi i računa se opterećenje koje će instanca da ima u slučaju da se migrira na neopterećeni host. Ukupno opterećenje se oduzima od hosta sa kojeg se instanca migrira, a dodaje na ukupno opterećenje hosta gdje instanca treba da se migrira. Ukoliko je neopterećenji host prošao provjeru resursa i ukoliko važi da je nova težina izvornog hosta veća od nove težine odredišnog hosta započinje migracija instance i izlazi se iz algoritma.

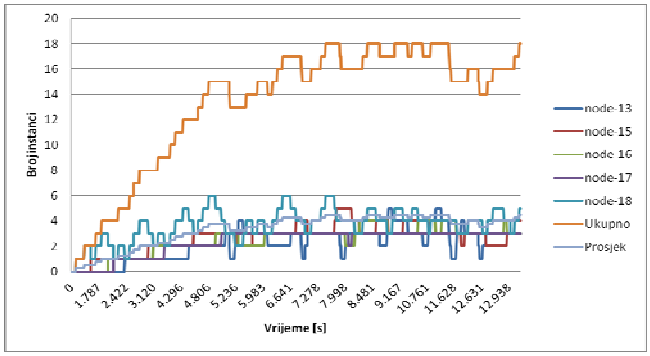
IV. REZULTATI TESTIRANJA

Sistem je testiran na infrastrukturi Elektrotehničkog fakulteta u Banjoj Luci koja se sastojala od šest hostova: jednog kontrolerskog i pet radnih hostova. OpenStack okruženje je instalirano pomoću verzije 6.0 Mirantis Fuel [9] u VirtualBox okruženju. Svi radni hostovi su bili identične konfiguracije: 4 procesora, 2 GB RAM i 3 diska po 65 GB.

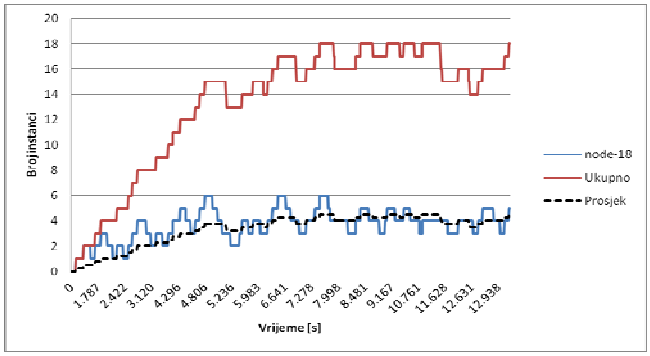
Da bi testirali funkcionisanje sistema na jednostavan i predvidiv način, kreiran je Python program za generisanje opterećenja sistema. Program funkcioniše na sljedeći način: na osnovu generisanih pseudoslučajnih vrijednosti se vrši pokretanje novih instanci na uvijek istom hostu i zaustavljanje instanci na uvijek istom hostu koji nije isti kao host na kom se instance pokreću. Ovo predstavlja najgori scenario koji bi bez upotrebe raspoređivača opterećenja vrlo brzo doveo do velikog nesrazmjera u sistemu. Korisniku je ostavljena mogućnost da definiše srednje vrijednosti intervala za pokretanje i zastavljanje instanci kao i varijansu neophodnu za generisanje pseudoslučajnih vrijednosti. Osim navedenog, moguće je definisati i detalje vezane za samu instancu koja se koristi za testiranje, kao i definisati hostove na kojima se instance pokreću i zaustavljaju.

U testu je korištena samo jedna slika za pokretanje instanci pod nazivom TestVM. Ova slika dolazi sa CirrOS [10] operativnim sistemom. Korištena je ova slika jer je pokretanje

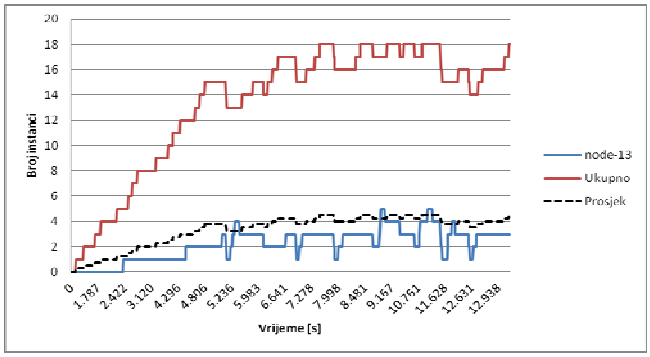
instance pomoću ove slike izuzetno efikasno, zahtijeva skromne resurse i samim tim migracije ovih instanci su brže. Flavor korišten za ovu instancu je m1.micro (1 VCPU, 64 MB RAM, 0 HDD, 0 GB Ephemeral Disk). Testiranje je trajalo 4 časa, a podaci o toku testiranja su pohranjeni u odgovarajućem *log* fajlu. U toku testiranja su pokrenute 33 instance a zaustavljeno je 15 instanci. Prikupljeni podaci su obrađeni i prikazani grafički. Na Sl. 2 je dat sumarni prikaz broja instanci u vremenu po svakom hostu, kao i ukupan broj instanci. Na slikama Sl. 3 i Sl. 4 je dat prikaz broja instanci u vremenu na hostu na kojem su instance pokretane (*node-18*) i zaustavljane (*node-13*).



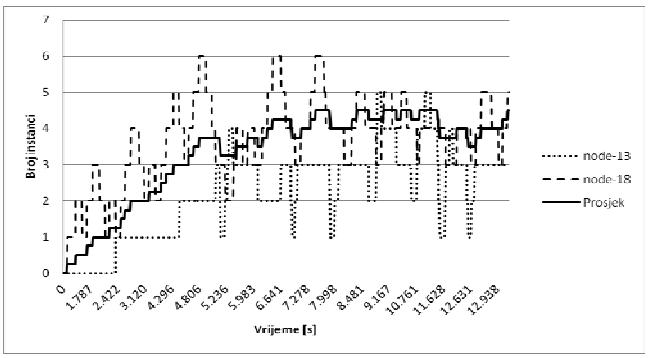
Slika 3. Prikaz promjene broja instanci na svim hostovima u vremenu



Slika 4. Prikaz promjene broja instanci na node-18 u vremenu



Slika 5. Prikaz promjene broja instanci na node-13 u vremenu



Slika 6. Hostovi node-13 i node-18

Rezultati ilustrovani slikama su zadovoljavajući jer se može primijetiti da je testirani sistem pravilno raspoređivao instance. Na slikama, a pogotovo na slici Sl. 6, se mogu identifikovati vrhovi koji pokazuju kada su instance migrirane sa opterećenog na slobodni host. Broj instanci na hostovima je približan vrijednosti aritmetičke sredine ukupnog broja instanci na svim hostovima. Na graficima za pojedinačni host pored ukupnog broja instanci u sistemu i trenutnog broja instanci na hostu prikazan je idealan broj instanci koji bi trebao da se nalazi na hostu. Može se vidjeti da se ta vrijednost većim dijelom poklapa sa vrijednošću trenutnog broja instanci. Očekivano se pojavljuju i manja odstupanja od idealnog broja instanci kojima su uzrok naglo pokretanje većeg broja instanci na hostu *node-18* i zaustavljanje većeg broja instanci na hostu *node-13*, jer je potrebno neko konačno vrijeme za migraciju instanci na slobodan host.

Rezultati koji su prikupljeni tokom testiranja pokazali su da ovaj sistem vrši adekvatno balansiranje instanci u sistemu. Raspoređivanjem opterećenja poboljšavaju se performanse i kvalitet usluga u *cloud* sistemu. Neke dalje analize ovog sistema bi mogle biti usmjerene u ovom pravcu, analizi poboljšanja performansi. Iako je sistem pokazao zadovoljavajuće rezultate, pravi test za ovaj realizovani sistem bi bio upotreba u okviru produkcione OpenStack infrastrukture koja obrađuje realne korisničke zahtjeve i koristi veliki broj različitih instanci

Postoji nekoliko planova u budućnosti za poboljšanje trenutnog rada. Jedan od tih planova jeste testiranje OpenBalancer sistema u okviru produkcione OpenStack infrastrukture. Analizom prikupljenih podataka tokom rada sa takvom OpenStack strukturom ukazao bi na eventualne mane sistema čime bi se odgovarajućom reorganizacijom strukture sistema i optimizacijom algoritama mogla izvršiti optimizacija performansi izvršavanja OpenBalancer sistema. S obzirom da je OpenBalancer trenutno konzolna aplikacija i tokom izvršavanja korisnik nema odgovarajuće povratne informacije od aplikacije, drugi plan za poboljšanje sistema je pravljenje korisničkog interfejsa u vidu web aplikacije. Kroz interfejs web aplikacije korisnik bi imao jasan pregled trenutnog stanja sistema. Korisnik bi mogao da vidi koji *nova-compute* hostovi postoje u sistemu, koje instance se izvršavaju na njima i koje instance se trenutno migriraju. S obzirom da se za hostove i instance prikuplja dosta podataka, korisnik bi imao detaljne

informacije o svakom hostu i instanci u sistemu. Takođe bi se korisniku ostavila mogućnost da kroz web interfejs dinamički mijenja parametre koje koristi algoritam za balansiranje i tako utiče na raspoređivanje instanci tokom izvršanja aplikacije.

LITERATURA

[1] "What is OpenStack", *opensource.com*, http://opensource.com/resources/what-is-openstack

[2] P. Mell, T.Grance, *The NIST Defintion of Cloud Computing*, studija. National Institute of Standards and Technology, Gaithersburg, 2011.

[3] L. Columbus, "Making Cloud Computing Pay", *Forbes Tech*, http://www.forbes.com/sites/louiscolumbus/2013/04/10/making-cloud-computing-pay-2/

[4] B. K. Douglas, "Cloud Computing Definition", *Service Architecture*, http://www.service-architecture.com/articles/cloud-computing/cloud_computing_definition.html

[5] D. Radez, "OpenStack architecture" iz *OpenStack Essentials,* Packt Publishing, Birmingham, 2015, glava 1, str. 2-2.

[6] "OpenStack API Documentation.", http://developer.openstack.org/api-ref.html.

[7] "The novaclient Python API — python-novaclient 3.2.0 documentation.", http://docs.openstack.org/developer/python-novaclient/api.html.

[8] "AMQP and Nova", http://docs.openstack.org/developer/nova/rpc.html.

[9] "OpenStack Deployment | Installer | Fuel | Mirantis.", https://www.mirantis.com/products/mirantis-openstack-software/openstack-deployment-fuel/.

[10] "CirrOS in Launchpad.", https://launchpad.net/cirros.

ABSTRACT

As a result of the use of cloud computing services, a number of commercial and open source platforms were created to manage cloud systems. One of these platforms is the OpenStack open source platform designed for all types of cloud systems. This platform is becoming, day by day, more popular and some believe that it will be the future of cloud computing. How important this project is is shown by the number of international companies and the number of members of the open source community that is involved in its development. OpenStack is a modular platform and consists of several projects. Price of modular design of this platform is a complex installation. Over time installation tools are developed for automated installation of OpenStack services. User interaction with the system is enabled via a web interface, client libraries and CLI. As a result of the analysis and research work of this platform in this paper we proposed one solution for load balancing in OpenStack system. The system bears the name OpenBalancer and it is implemented in the Python programming language.

OPENBALANCER – OPENSTACK LOAD BALANCER

Aleksandar Vukotić, Mihajlo Savić