Univerzitet u Novom Sadu, Prirodno – matematički fakultet

DEPARTMAN ZA MATEMATIKU I INFORMATIKU

Trg Dositeja Obradovića 4, 21000 Novi Sad

Komunikacija u distribuiranim sistemima orijentisana porukama

Master rad

dr Danijela Boberić Krstićev

Robert Sabo

Novi Sad, 2020

Predgovor

U ovom radu je detaljno i naširoko obrađen problem komunikacije u distribuiranim sistemima. Problem se fokusira na *publish-subscribe* paradigmu komunikacije. Nakon poređenja sa drugim radovima, odabir spada na 3 brokera koja su detaljno analizirana, a to su: MQTT, Redis i Kafka.

Detaljno je istraživan svaki tip brokera, te su predstavljeni osnovni i napredni koncepti, kao i prednosti i mane svakog brokera. Nakon temeljnog razumevanja funkcionisanja sva 3 brokera rad prelazi na razvoj aplikacije za testiranje performansi svakog od brokera, te analiza i poređenje rezultata.

Po uspešnom postavljanju i konfigurisanju okruženja, uspešnom razvoju i pokretanju aplikacije za testiranje performansi, izvršeni su testovi. U ovom radu predstavljeni su rezultati dobijeni ovim testovima, koji kreću od lakših i osnovnijih. Napretkom testiranja se povećavaju ciljane performanse. Testiranje ide do granica podnošljivosti brokera. Neki od testova su prikazani i u ovom radu.

1. U samom uvodnom delu objašnjeno je šta je distribuirani sistem, te kakva komunikacija se realizuje unutar distribuiranog sistema. Nakon toga se objašnjava paradigma komunikacije kojom se ovaj rad bavi.
2. Drugo poglavlje je posvećeno istraživanju drugih radova na sličnu temu. Istraživanje ima svoj tok i kreće od osnovne problematike skalabilnosti distribuiranog sistema, preko različitih rešenja komunikacije, do samog odabira brokera za testiranje ovog rada.
3. Poglavlje broj 3 je posvećeno istraživanju i opisu konkretnih brokera na osnovnom i naprednom nivou svakog brokera.
4. Poglavlje 4 predstavlja ideju, arhitekturu i implementaciju aplikacije za testiranje performansi brokera
5. Poglavlje prezentovanja rezultata gde su prikazani neki od rezultata osnovnih i naprednih testova
6. Potom ide poglavlje posvećeno diskusiji rezultata i tumačenju istih
7. Zaključak

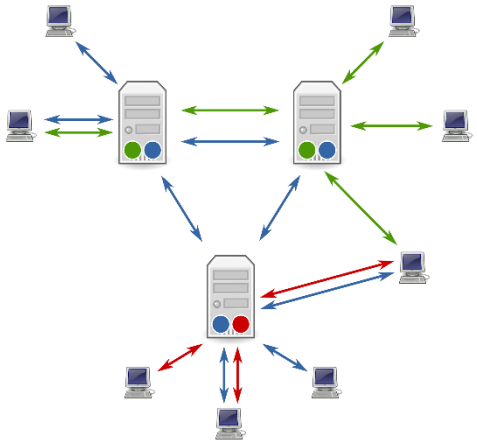
Sadržaj

Poglavlje 1

Uvod

Svedoci smo svakodnevnih inovacija. Iz dana u dan pojavljuju se nove ideje pa time i novi koraci u razvoju tehnologije. Upotreba pametnih uređaja eksponencijalno raste. Time, podataka je sve više i više. Rastom prikupljanja podataka raste i potreban potencijal da se podaci obrade, pretvore u korisne informacije te na kraju u usvojivo znanje.

Istorijski razvoj procesa obrade podataka je dugačak i obiman za jedan uvod u rad. Značajno je nepomenuti da već godinama obrada podataka najčešće biva decentralizovana. Ako ne i fizička, odvojenost u arhitekturi je prisutna u različitim celinama obrade. Međusobna povezanost celina se ogleda u komunikaciji. Primer podele celina u distribuiranom sistemu je prikazan na slici 1.1



Slika 1.1 Ilustracija komunikacije

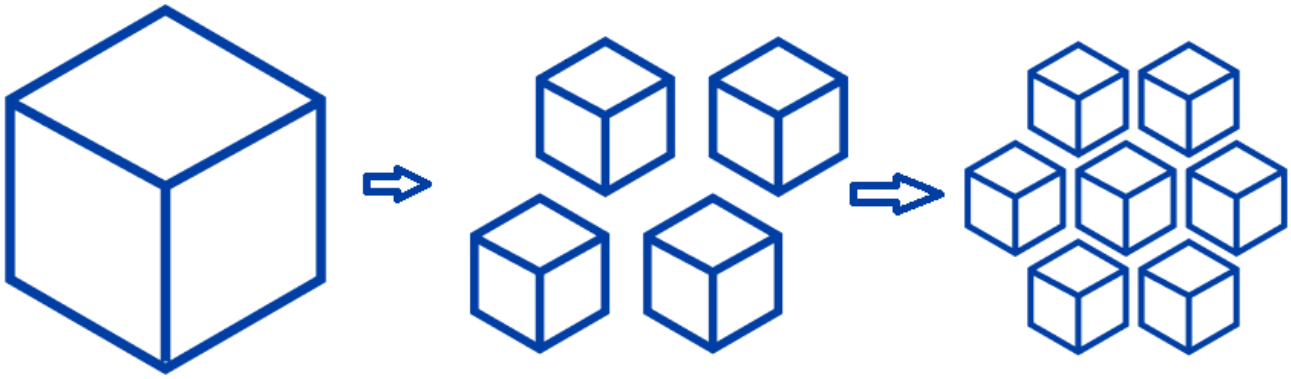
Konfuzijom raznih pitanja vezanih za komunikaciju inspirisan je ovaj rad sa temom: *Komunikacija u distribuiranim sistemima orijentisana porukama.* Iako sam naslov ne odgovara na sva pitanja, u radu se detaljno obrađuje i analizira jedno po jedno.

1.1 Komunikacija u distribuiranim sistemima orijentisana porukama

Radi razjašnjenja čime se to ovaj rad bavi najbolje je početi analizom samog naslova. Stoga će ovaj deo uvoda objasniti jedan po jedan pojam naslova u cilju razjašnjavanja šta je šta.

1.1.1 Distribuirani sistemi

Porastom neophodnog potencijala za obradom podataka menjala se arhitektura softvera koji ga obrađuje. U početku je količina podataka potrebna za obradu bila značajno manjih razmera te je jedan računar samostalno uspeo da obradi sve podatke. Rast količine podataka za obradu se prati pojačavanjem hardverskih komponenti. Prekretnica se dešava kada se ustanovi da je jedan jak računaj mnogo skuplji nego dva osrednja. Došlo se na ideju da se posao obrade rasporedi na 2 celine i da se svaka celina obrađuje podatke, ali na zasebnoj hardverskoj komponenti.



Slika 1.2 Razvoj distribuiranog sistema

Istim putem, koji je od jednog doveo do 2 hardvera, se došlo i do raspodele na *n* računara. Broj n se vremenom sve više povećava (kao što je i ilustrovano na slici 1.2). Razlog je jednostavan: distribucija se pokazala kao dobra praksa.

Svakako se prvi cilj i prednost distribucija ogledala u iskorišćenosti i ceni hardvera. Međutim, to se pokazalo kao značajan napredak i u softveru. Naime, podeliti problem na prostije celine je u prirodi funkcionisanja ljudskog mozga tokom razmišljanja. Čovek kada dobije kompleksan problem, obradi ga tako što podeli na manje i trivijalne probleme. Rešavanjem manjih, trivijalnih problema biva rešen i inicijalni kompleksan problem. Činiti isto i u razvoju softvera predstavlja potpuno prirodan i očekivan korak. Lakše je da se problem raščlani na više celina te da se rešavaju pojedinačno, pa se i sama softverska arhitektura sve više razvijala u manjim softverskim rešenjima koji zajedno daju sistem.

Distribuiran sistem predstavlja rešenje podeljeno na manje celine koji u korelaciji rešavaju zadati problem. Delovi distribuiranog sistema mogu biti fizički i geografski odvojeni, dok je moguće, s druge strane, i da je podela samo logička. Kako god se sistem delio, manje celine predstavljaju logički udeo u rešenju problema. Jedna celina je zasebna i nezavisna i bavi se rešavanjem trivijalnog problema.

Osim uprošćavanja rešenja problema, podela na celine donosi i niz drugih prednosti. Međutim, ovaj rad se ne bavi time i distribuirani sistemi su samo oblast u kojoj se ogleda srž ovog rada. Istraživanje o mikroservis orijentisanoj arhitekturi, što je jedan vid distribucije sistema, se može pronaći u radu [1]

1.1.2 Komunikacija u distribuiranim sistemima

Postavlja se pitanje odnosa celina distribuiranog sistema. Bilo koja arhitektura i odnos delova distribucije da se primenjuje, jedno je sigurno: delovi sistema moraju da komuniciraju. Komunikacija unutar sistema se može definisati kao razmena podataka. Dakle, svaki distribuirani sistem sadrži neki vid komunikacije u smislu razmene poruka između svojih sastavnih delova.

Vrste komunikacije mogu biti različite. Podelu je moguća po raznim kriterijumima i dobijaju se sasvim jasne definicije i uočljive razlike. U ovom radu neće biti obrađena svaka podela, štaviše samo će se neke od njih pomenuti a jedna detaljno objasniti.

1.1.3 Komunikacija orijentisan porukama

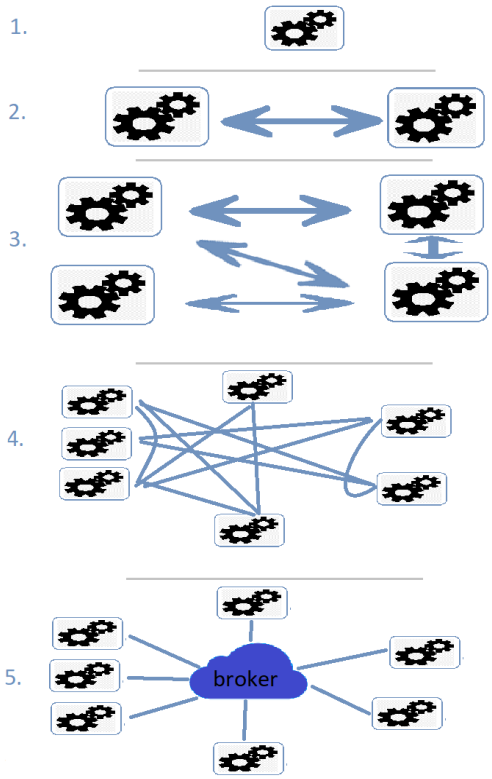
Prilikom razmene podataka postoje različite svrhe i upotrebljivosti razmenjenih podataka. Ovaj rad se fokusira na komunikaciju orijentisanu ka porukama. To znači da je tok podataka ključni pokretač sistema. Samim tim poruke predstavljaju prioritet.

U ovakvim sistemima svaka poruka je ključna i gubitak podataka se ne toleriše. Svaka poruka ima svoju težinu i obavezno mora biti ispostavljena svim delovima sistema koji to zahtevaju.

Do ogromnog značaja dolazi pitanje komunikacije u trenucima ekspanzije upotrebe sistema. Prethodno postavljena težina poruka predstavlja problem tek kada sistem postaje zasićen i pokazuje prve simptome zagušenja. Zbog toga je neophodno da se u ranim fazama sklapanja arhitekture komponenti sistema ima na umu mogućnost opterećenosti sistema, te se na odgovarajući način pripremiti za to. U tim fazama odabir načina i alata komunikacije predstavlja ključnu i tešku odluku.

1.2 Brokeri poruka

Broker poruka je alat, odnosno softver, koji ima ulogu razmene poruka. Radi lakšeg razumevanja, pre objašnjenja čemu služi i za šta se koristi, biće objašnjeno kako jedan sistem dođe do potrebe za brokerom poruka. Ilustracija na slici 1.3 je slikoviti prikaz toka razvoja jednog sistema i sastoji se od, grubo rečeno, sledećih koraka:



Slika 1.3 Put do brokera poruka

1. U prvoj fazi se ne vidi potreba za distribuiranim sistemom, te se razvije jedno softversko rešenje.
2. Nakon nekog vremena rada, dolazi do potrebe za pomoćnim alatom koji dopunjava rad prvobitnog. Ova 2 alata međusobno komuniciraju direktom vezom, jedan od njih poziva drugog po nekom protokolu.
3. Narednim zahtevima za razvoj dolazi do potrebe za uvođenjem još 2 alata koji takođe direktno komuniciraju sa delovima sa kojima ima potrebe. U ovom koraku se ova struktura može nazvati distribuiran sistem.
4. Komplikovanjem zahteva biznis logike i daljim razvojem uvode se sledeće potrebne celine. Dodavanjem još celina i zadržavanjem direktne komunikaciji dolazi do haosa. U 4. koraku distribuirani sistem ima 7 celina i svaka celina direktno poziva nekoliko drugih. Postaje previše komplikovano ispratiti tok komunikacije, povezanost celina i ko s kim komunicira.
5. Kada sistem poraste do koraka 4 i postane nejasno definisati komunikaciju, uvodi se pomoćni alat koji vodi računa o komunikaciji. To je broker poruka. Svi delovi sistema komuniciraju sa brokerom i

jedina direktna komunikacija svakog dela sistema je sa brokerom. Veze između delova sistema i dalje postoje, neophodna komunikacija je i dalje prisutna, samo što ona nije direktna, već se odvija preko brokera.

Broker ima zadatak da prati tok podataka. Može se zamisliti kao raskrsnica, saobraćajnica poruka. Svaka poruka stigne do njega, te on usmerava kuda sve ta poruka treba da ode. Površno se izriče problem što veza nije direktna, jer veza između celina zaista nije direktna. Međutim direktna veza nije neophodna. Delovi sistema postaju nezavisniji, odrade svoj zahtev i odgovor šalju na broker. Ni jedna celina ne mora da vodi računa kome sve odgovor treba da ispostavi. Štaviše, u dobro dizajniranim sistemima celine ne bi trebale da imaju dodirnih tačaka s drugim celinama. Dakle, distribuirani sistem je ostvarim indirektnom vezom, preko brokera.

Radi lakšeg pojašnjenja prednosti upotrebe brokera poruka uzeće se u analizu poređenje 4. i 5. koraka sa ilustracija prikazane na slici 1.3. Prednosti upotrebe brokera su sledeće:

- Nezavisnost delova sistema – svaka celina je priča za sebe. Radi svoj posao i ne komplikuje logiku brigom o tome kome treba dostaviti podatke. Na 4. koraku deo sistema ilustrovan s desne strane gore šalje podatke na 3 druga procesa. Neophodno je voditi računa o sva 3 procesa, dostupnosti istih i slično. Kad bi se dodao još jedan proces kojem pomenuti proces šalje podatke, bilo bi potrebno raditi izmene na istom kako bi se podaci usmerili i ka novom procesu. U slučaju korišćenja brokera poruka, pominjani proces poruke šalje na broker i ne interesuje ga ni dostupnost drugih procesa ni dodavanje novih.
- Nadzor toka podataka – na 4. koraku bi bilo izuzetno teško, gotovo nemoguće, napraviti kompletno pokriven nadzor toka poruka u smislu koliko podataka vezanih za određeni deo biva razmenjeno i kada. Na koraku 5 bi se prostim dodavanjem još jednog procesa za nadzor moglo napraviti rešenje analize svih poruka.
- Ponovno iskorišćenost – ova prednost se nadovezuje na nezavisnost. Naime, kada se rešenja pišu univerzalno, nezavisno od sistema mnogo ih je lakše iskoristiti kasnije i kao deo nekog drugog sistema. Dok bi, s druge strane, jedan proces iz koraka 4. zbog direktne komunikacije s drugim delovima te usko zavisnim od njih, bio mnogo teže primenljiv u nekim drugim sistemima.
- Sigurnost isporuke – kod brokera poruka sam broker je taj koji vodi računa da poruka bude dostavljena na sve neophodne delove. Dok u slučaju prikazanom na koraku 4 svaki proces bi morao da ima svoju logiku provere dostave poruka

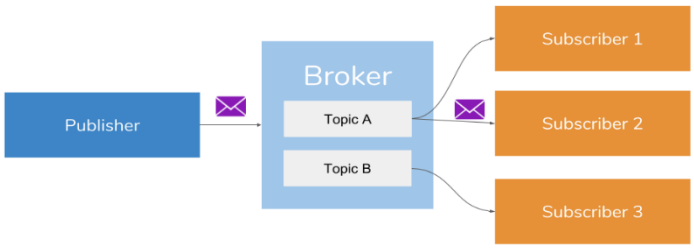
Pored prednosti, naravno, postoje i mane korišćenja brokera. Glavna mana ogleda se u potrebi za postavljanjem te i održavanjem još jednog softvera u sistemu. Ova mana povlači niz omanjih potencijalnih problema kao što su: potrebe resursa brokera, mogući problemi dostupnosti, poteškoće konfigurisanja i slično. Međutim, naveden problem je konačan i zanemariv u odnosu na prednosti upotrebe brokera.

1.3 Publish-subscribe patern

Zajednička tačka svih brokera poruka je prisutnost šablona raspodele uloga u komunikaciji. Naime, svi brokeri poruka rade po tzv. *publish-subscribe* šablonu (od engleskog *publish-subscrib* čiji prevod bi bio objava-pretplata. Zbog prirodnijeg izgovora u ovom radu će se koristiti prisvojene engleske reči). Ovaj šablon predstavlja raspored uloga u komunikaciji. Na slici 1.4 su slikovito predstavljene sve 3 uloge ovog šablona, a to su:

1. Publisher – subjekat koji šalje poruke
2. Broker – sam posrednik. Subjekat koji je u konekciji i sa *publisher* i sa *subscriber* komponentom, te razmenjuje poruke.
3. Subscriber – subjekat koji se pretplati, te prima poruke.

Pored uloga, još jedan termin je neophodno definisati kod ove paradigme: tema (eng. topic). Tema može da se predstavi kao kanal toka podataka. Podaci u različitim temama se ne mešaju. *Subscriber* prilikom pretplate navodi temu na koji se pretplaćuje. To znači da će on dobijati sve poruke koje se šalju na navedenu temu. Takođe, *publisher* u trenutku slanja poruke navodi na koju temu šalje poruku. U trenutku kada *publisher* pošalje poruku na određenu temu, broker prima poruku, te je prosleđuju svim *subscriber* komponentama koji su se pretplatili na tu temu.



Slika 1.4 Uloge u publish-subscribe komunikaciji

Brojnost *publisher* i *subscriber* komponenti nije ograničena. Broker poruke raspoređuje i prosleđuje poruke po temama. Broj *publisher* komponenti povezanih sa brokerom ne igra ulogu. Sve *subscriber* komponente koje su pretplaćene na određenu temu će dobiti sve poslate poruku od strane *publisher* komponente. Dakle, ni broj *subscriber* komponenti nije bitan za broker.

Evo jednog hronološki ispraćenog primera toka razmene podataka preko brokera sa dve *subscriber* i jednim *publisher* komponentom:

- *Subscriber* 1 uspostavlja konekciju sa broker
- *Subscriber* 1 se pretplaćuje na temu *Temperatura*
- *Subscriber* 2 uspostavlja konekciju sa broker
- *Subscriber* 2 se pretplaćuje na temu *Temperatura*
- *Publisher* šalje poruku „23.44“ na temu *Temperatura*
- Broker prima poruku i prosleđuje na sve *subscriber* komponente pretplaćene na temu
- *Subscriber* 1 i *Subscriber* 2 dobijaju oba po jednu poruku „23.44“

Navedena pravila i ograničenja su uopštena. Kod različitih implementacija brokera se dešava da variraju neka od navedenih pravila. Primeri varijacija će biti obrađeni u nastavku ovog rada.

Poglavlje 2

Istraživanje na temu

Ovo poglavlje opisuje radove i publikacije sa sličnom tematikom istraženih u cilju poređenja i izučavanja problematike ovog rada.

Mnogo radova postoji koji se bave skalabilnošću te ograničenjima i proširenjem ograničenja nekih sistema. Ali, većina njih se bave izučavanjem i istraživanjem konkretnih tehnologija.

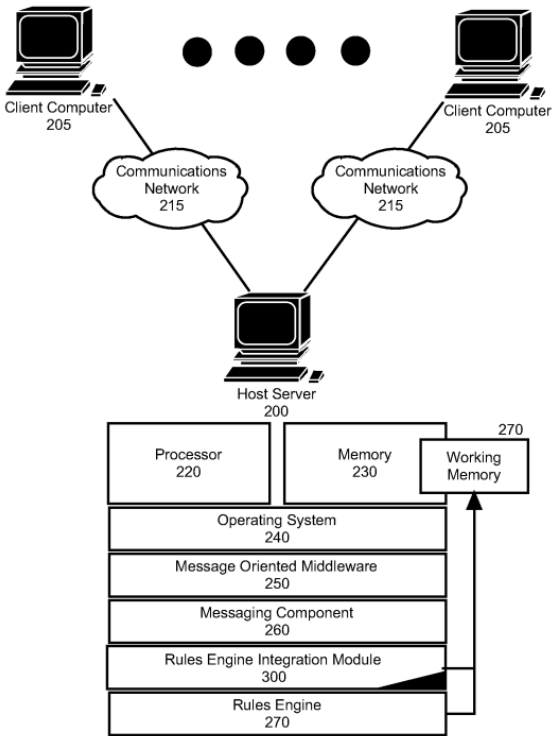
Rad na temu *Skalabilan mikroservis orijentisan sistem namenjen berzi kriptovaluta* [1] se bavi predstavljanjem primera distribuiranog sistema sa konkretnim namenama u konkretnoj tehnologiji. Tema je vezana za distribuirane sisteme i bavi se razvojem i pojašnjenjem primera jednog skalabilnog i visoko dostupnog sistema. Arhitektura komponenti sistema je napravljena tako da bude pogodna za horizontalno skaliranje [4] samih komponenti. Komunikacija između komponenti je direktna putem REST API protokola.

U navedenom radu je korektno postavljena arhitektura u cilju skalabilnosti, ali svojom arhitekturom nalazi se na 2. ili 3. koraku sa ilustracije prikazane na slici 1.3. To znači ,da bi vremenom i daljim razvojem tog sistema, te dodavanjem komponenti i zahteva došlo do potrebe za korišćenjem brokera. Možda ne strogo neophodno, ali svakako bi poželjno bilo prebaciti komunikaciju na korišćenje brokera.

Patent sa naslovom *Message oriented middleware with integrated rules engine* [2] predstavlja rad na temu usko vezanu za ovaj rad. Inicijalna problematika oba rada se podudara, a to je kako uprostiti komunikaciju u distribuiranim sistemima. U navedenom radu se iznosi potreba za izdvajanjem alata koji će se baviti samo posredovanjem poruka. Time se donose značajne prednosti i olakšanja u distribuiranom sistemu.

Navedeni patent i ovaj rad imaju usku povezanost teme i inicijalne problematike. Međutim, bitna razlika je prisutna: pomenuti rad predstavlja patent, dok je ovo naučno istraživački rad. U patentu se predstavlja arhitektura implementacije posrednika poruka. Pored arhitekture, rad se bavi i problemima integracije do nivoa hardvera. Osim toga, može se pronaći i detaljno analiziran slučaj obrađivanja poruka u smislu dijagrama toka podataka unutar samog posrednika.

Ovaj master rad je fokusiran na upotrebu posrednika na višem nivou apstrakcije, ne obrađuje detalje vezane za implementaciju posrednika kao ni na moguće mehanizme pravila posrednika. Posrednik sa integrisanim mehanizmom pravila predstavlja otvorenu mogućnost za bilo koji način implementacije pravila raspoređivanja poruka. On može da bude implementiran s bilo kojim mehanizmom pravila. Dok se, s druge strane, ovaj rad fokusira na *publish-subscribe* paradigmu u već postojećim rešenjima, te poređenje istih. U ovom radu neće biti implementiran posrednik, već se samo koriste te porede postojeći posrednici u konkretnom šablonu.

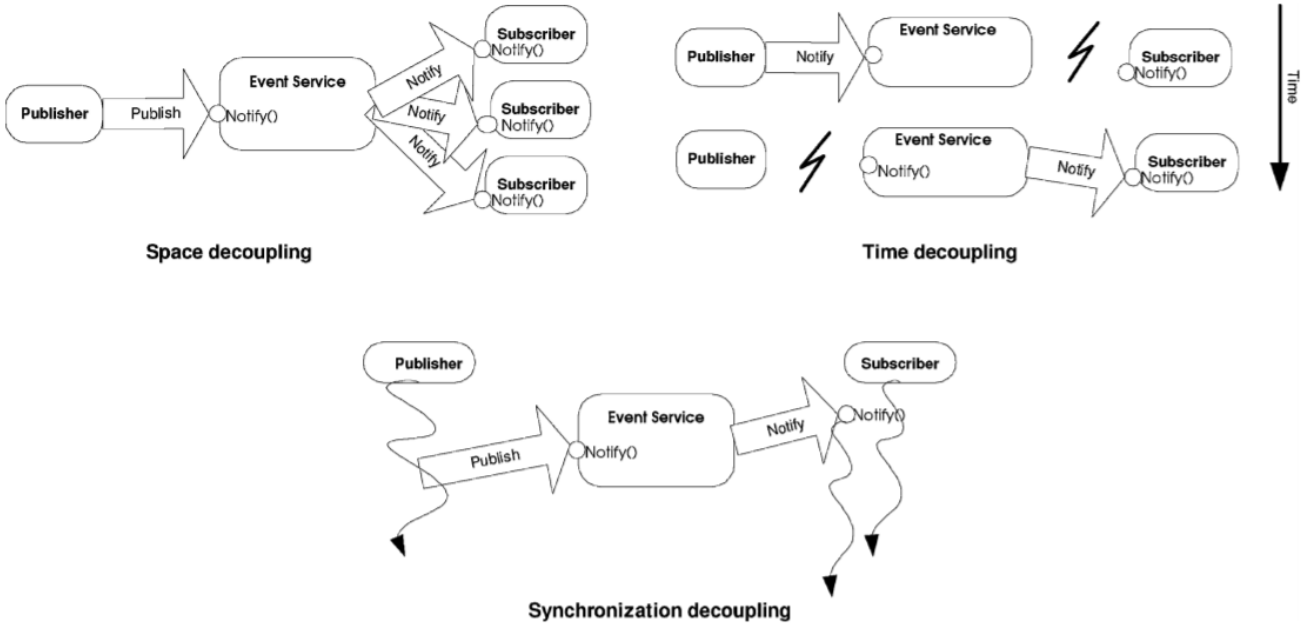


Slika 2.1 Ilustracija posrednika sa integrisanim mahanizmom pravila

Rad koji značajno detaljnije razrađuje *publish-subscribe* paradigmu je *The many faces of Publish/Subscribe* [3]. Naveden rad se bavi razjašnjavanjem i objašnjavanjem ove komunikacione paradigme. Vrši poređenje sa mnogim drugim načinima komunikacije između celina distribuiranog sistema, kao što su:

- *message queuing* (red opsluživanja poruka) - razmena poruka kroz redove opsluživanja,
- *RPC/RMI* - RPC - Remote Procedure Call; RMI – Remote Method Invocation; oba označavaju pozivanje funkcija odnosno metoda udaljenog softvera,
- *Shared space* (deljena memorija) – ramena podataka deljenjem iste memorijske jedinice (može biti fajl sistem, baza podataka i sl.),
- *Notifications* (obaveštenja) – prisutna upotreba *obećanja* (*promise* poznatiji u web razvoju); *observable* patern;

Kako se u samom radu navodi, ovi načini komunikacije su preteča *publish-subscribe* paterna. Svaki od njih može da se koristi i ostvariva je komunikacija, ali su prisutne značajne prednosti upotrebe *publish-subcribe* paterna. U radu se prednosti iskazuju u razdvojenosti sistema po 3 dimenzije: prostor, vreme i sinhronizovanost (slika 2.2).



Slika 2.2 Odvojenost u 3 dimenzije

Svaka dimenzija odvojenosti je prethodno pominjana u ovom radu, u delu gde se naznačene prednosti upotrebe brokera zaokruženo kao nezavisnost delova sistema, dok pomenuti rad detaljnije objašnjava. Odvojenost komponenti u prostoru se ogleda u ne korišćenju istog memorijskog prostora. Delovi sistema komuniciraju sa brokerom i dodirna tačka deljenja memorije im je memorija brokera, ali tim prostorom sam broker upravlja. Vremenska odvojenost se ogleda u nezavisnosti delova sistema od slanja do isporuke poruke. Postoji mogućnost da u momentu slanja poruke *subscriber* nije prisutan, te da poruku primi naknadno (ilustrovano u gornjem desnom uglu slike 2.2). Desinhronizovanost, odnosno odvojenost sinhronizacije, je jasna jer nijedan subjekat ne čeka odgovora, dakle nakon slanja poruke *publisher* nastavlja s radom. S druge strane, *subsciber* takođe asinhrono i nezavisno prima poruke.

Interesantan deo rada ogleda se u varijacijama *publish-subscribe*-a. Uobičajeno, te i ovaj rad obrađuje *publish-subscribe* baziran na temama. Svaka poruka se šalje na temu, *subscriber* komponente naznačavaju koja tema ih zanima. Međutim, to je samo jedan od tipova *publish-subscribe* komunikacije i nazivaju ga baziran na temama. Postoje i drugi kao što su: baziran na tipovima i baziran na sadržaju. Ovaj rad se zadržava na najzastupljenijem i najprirodnijem, odnosno baziranom na temama.

2.1 Odabir brokera

Daljim istraživanjem nailazi se na rad slične teme, ali novijeg datuma. Rad pod nazivom *Ensuring Low-Latency and Scalable Data Dissemination for Smart-City Applications* [5] se bavi problemom komunikaciju u konkretnom distribuiranom sistemu. Domen primene je pametan grad. S obzirom na domen, očekuje se ogromna količina podataka te je neophodno odabrati odgovarajući alat za komunikaciju. Povezanost sa ovim radom se ogleda bas u odabiru paradigme jer i taj rad bira *publish-subscribe* kao najpogodniji i odgovarajući zahtevima.

Pomenuti rad predstavlja paralelu ovog master rada. Pre svega, u tom radu se pominju implementacije gotovih brokera koji su predmet i ovog rada, što je direktna posledica datuma kada je rađen rad. Noviji rad direktno podstiče analizu danas realno upotrebljivanih brokera, što nije slučaj kod prethodno pominjanih radova. U tom radu se pominju konkretno: MQTT, Kafka i Redis uz navođenje da se sve više koriste u realnim IoT (*Internet of Things*) sistemima. Konfigurisanje parametara i samo testiranje ponašanja je ekvivalent eksperimentalnog dela ovog rada. Razlikuje se što se je u tom radu testira njihova interna implementacija *publish-subscribe* brokera koja je najsličnija Kafki. Ipak ovaj rad se bavi i pokriva eksperiment sa 3 postojeća i već razvijena brokera.

Poglavlje 3

Brokeri

Ovo poglavlje predstavlja teorijski uvod u eksperimentalan deo ovog rada. Potrebno je navesti odabrane implementacije brokera, te razjasniti šta su osnovne karakteristike svakog od njih. Potreba za detaljnom teorijskom bazom je neophodna radi lakšeg razumevanja eksperimentalnog dela.

3.1 MQTT - *mosquitto*

Prvi ujedno i najprostiji broker je MQTT. MQTT je skraćenica od MQ telemetrijski transport (eng. *MQ Telemetry Transport*). *MQ* je preuzeto od IBM MQ, gde MQ predstavlja skraćenicu od red posluživanja poruka (eng. *message queue*). Cela skraćenica je, dakle, *message queue telemetry transport*.

MQTT predstavlja lako-kategorijski protokol razmena poruka [7]. Realizuje, prethodno objašnjen, *publish-subscribe* šablon komunikacije. Kao broker koristi direktnu vezu sa komunikacionim uređajima. Direktna veza sa klijentima se realizuje putem mrežnog TCP/IP protokola ili, takođe moguća, *websocket* komunikacija. MQTT podržava i bezbednu komunikaciju na bazi SSL-a (*Secure Sockets Layer*). MQTT se najčešće koristi, te i najznačajniju ulogu ima u polju IoT-a.

Uz MQTT se uvek usko veže termin *lightweight*, što se tumači kao kategorijski lagan i prost. To znači da je MQTT nužno jednostavan i nisko zahtevan protokol komunikacije, te nisko zahtevan po hardverske resurse. Lako kategorisanje predstavlja pogodnu upotrebu i za bežične konekcije sa mogućim kašnjenjem i gubitkom podataka. Klijentske biblioteke MQTT protokola su široko dostupne u većini programskih jezika i neopterećujuće su po softver zbog prirode samog protokola.

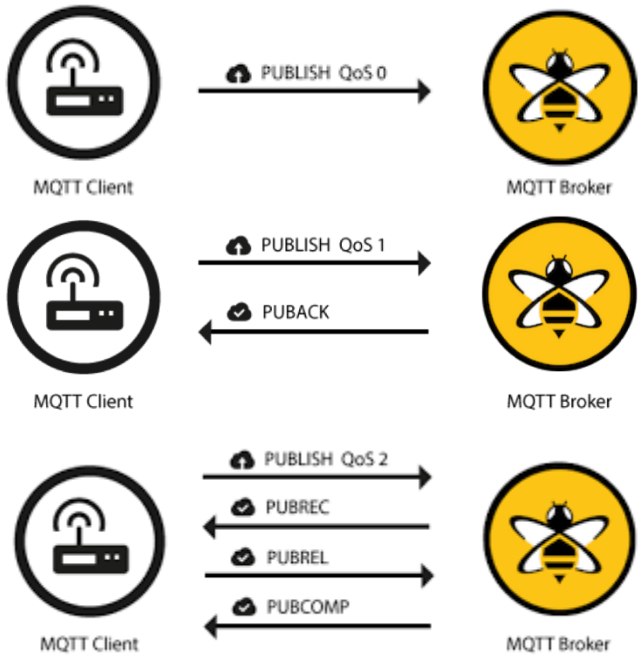
MQTT broker, odnosno server, mora biti dostupan svim klijentima u komunikaciji. Najčešće se MQTT postavlja na mrežu u *cloud* kako bi bio dostupan svima. Za klijent, u MQTT komunikaciji, je bitno da bude na mreži (internet ili lokalnoj) te da mu server bude vidljiv.

MQTT kao broker *publish-subscribe* paradigme sadrži 2 uloge klijenata: *publisher* i *subscriber*. Bez obzira koja uloga klijenta je u pitanju, prvo i neophodno je uspostavljanje komunikacije sa serverom, odnosno MQTT brokerom. Zavisno od podešavanja, postoji mogućnost potpuno različitih načina bezbedne veze a to su: otvorene konekcije, obezbeđene autentifikacijom i bezbedne (*secure*) komunikacije bazirane na SSL-u. Po uspostavljanju komunikacije klijent može da se pretplati na temu (*subscriber*) ili da pošalje neke poruke (*publisher*). Moguće je da jedan klijent bude istovremeno i *subscriber* i *publisher*, jer nakon pretplate na temu klijent može i sam da šalje poruke.

3.1.1 Quality of Service

Karakteristično za MQTT je *Quality of Service*, skraćeno QoS, odnosno kvalitet usluge. Klijenti prilikom obe akcije, i slanja poruke i pretplate na temu, definišu QoS. Numerička vrednost 0, 1 ili 2 koja označava stepen proverenosti razmene poruke između klijenta i servera. Svaka oznaka ima svoj naziv i objašnjenje:

- 0 – *najviše jednom (*eng. *at most once)*– nazivaju ga još i *pošalji i zaboravi* (eng. *fire and forget*); klijent pošalje poruku i ne očekuje potvrdu. Zbog toga postoji mogućnost da usred gubitka podataka poruka nikad ne bude isporučena, a najviše je moguće jednom da se isporuči jer se ne vrši ponovno pokušavanje slanja.
- 1 – *najmanje jednom (at least once)* – klijent pošalje i očekuje potvrdu o prijemu od servera. U slučaju da nema potvrde, poruka će se pokušati ponovo slati. Iz tog razloga, poruka će biti dostavljena bar jednom, ali postoji mogućnost i više puta u slučaju da se izgubi paket potvrde.
- 2 – *tačno jednom (exactly once)* – za razliku od prethodnog, ovde se nakon potvrde od strane servera ponovo šalje potvrda da je slanje završeno. Ovakva definicija garantuje da će poruka biti isporučena tačno jednom.



Slika 3.1 MQTT Quality of Service

Objašnjenje razmene poruka je predstavljeno i na slici 3.1. Shodno razmenjenim porukama postoje prednosti i mane različitog QoS-a. 0 je najjednostavniji te i najbrži. 2 je najsigurniji i najpreciznji, ali zahteva mnogo dodatne komunikacije za svaku poruku, te je značajno sporiji. U okviru praktičnog dela ovog rada će se obraditi i poređenje brzine komunikacije za različite QoS-e.

3.1.2 Zamenski karakteri pretplata na temu

Konvencija MQTT protokola nalaže da se ime teme gradi po nivoima te da se nivoi razdvajaju kosom crtom: „/“ (nije pravilo, niti je neophodno, već samo preporučeno). Tako bi, na primer, tma temperature u dnevnoj sobi bio: *kuca/dnevnasoba/temperatura*; za svetlo u istoj sobi bi bilo: *kuca/dnevnasoba/svetlo.*

Osnovni način pretplate jeste da se navede konkretan i ceo naziv te se primaju poruke poslate na navedenu temu. Međutim, MQTT sadrži još jednu prednost, takozvani *wildcards*, odnosno zamenski karakteri.

- # - predstavlja zamenski karakter za više nivoa. To znači da će klijent koji se pretplate sa nekim prefiksom i # na kraju dobijati sve poruke sa koje imaju isti prefiks. Isto važi i za sufiks, samo se # navodi pre sufiksa. Na primer, preplatiti se na *kuca/#* označava da će stizati sve poruke sa prefiksom *kuca/* , dakle i poruke sa teme *kuca/dnevnasoba/svetlo, kuca/kuhinja/temperatura* kao i sa *kuca/vlaznost.* Primer za sufiks bi bio *#/temperatura;* kada bi stizale sve poruke sa sufiksom na *temperatura,* dakle i *kuca/kuhinja/temperatura, kuca/dnevnasoba/temperatura* ali i *vikendica/soba/temperatura.* Nije ograničeno koristiti samo prefiks i sufiks, već je najbolja praksa. Postoji mogućnosti korišćenja i u vidu infiksa, kao na primer: *kuca/#/svetlo*¸ što je pretplata na sve teme koji počinju sa nivoom *kuca* a na kraju imaju nivo *svetlo*
- + - predstavlja zamanski karakter za tačno jedan nivo. Dakle, klijenti koji se pretplate na temu sa zamenskim karakterom + umesto konkretnog nivoa, dobijaće poruke bilo koja naznaka da stoji na tom nivou, ukoliko se ostali nivoi poklapaju. Na primer, pretplata na *kuca/+/temperatura* označava prijem svih poruka kojima je prvi nivo *kuca,* i treći nivo *temperatura*.

Korišćenjem nivoa u okviru team i zamenskih karaktera prilikom pretplate MQTT protokol dobija značajno opširniju i moćniju mogućnost kombinovanja razmene poruka.

3.1.3 Zadržane poruke

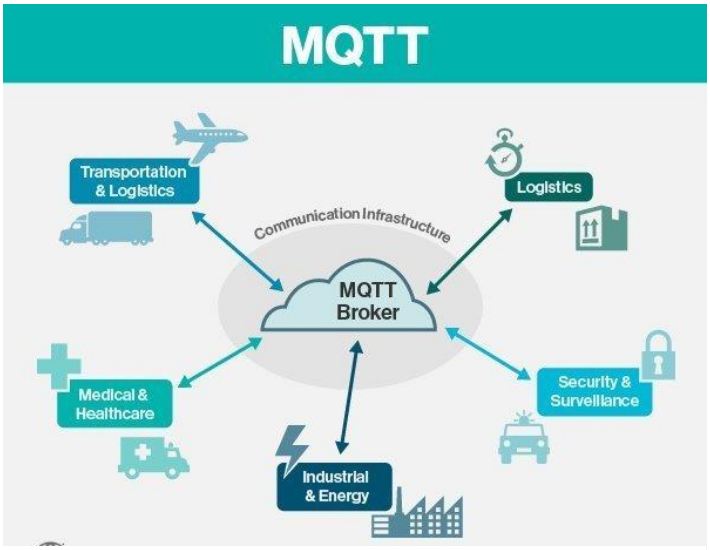
MQTT protokol podržava zadržavanje poruke, takozvani *retain*. Publisher prilikom slanja poruke može da pošalje sa *retain flag* naznakom odnosno naznakom zadržavanja. Poruke naznačene za zadržavanje broker čuva vezano za temu. Prilikom pretplate novog klijenta na temu, odmah će dobiti poslednju poruku poslatu na tu temu sa oznakom zadržavanja, bez obzira kada je poruka zaista poslata.

Pokazalo se kao dobra praksa da kada se šalje nekakvo stanje uvek poruke budu sa oznakom zadržavanja od strane *publisher* komponente. Tada će svaki *subscriber* uvek po pretplati dobiti uvid u poslednje prijavljeno stanje.

3.1.4 Upotreba

Zbog svoje lakoće MQTT je izuzetno pogodan za upotrebu u IoT-u. Najčešće se koristi za slanje podataka sa pametnih uređaja u *cloud* u cilju prikupljanja podataka radi prikazivanje trenutnog stanja, analize i održavanja. Konkretni primeri korišćenja su ilustrovani na slici 3.2 a to su: transport i logistika, pametne kuće, industrija, medicina i zdravstvena briga.

Najveća prednost MQTT-a se ogleda u rasprostranjenosti. Dugi niz godina je prisutan i u upotrebi pa gotovo sve platforme za IoT i većina *cloud* provajdera imaju podršku za



Slika 3.2 Upotreba MQTT-a

MQTT, kao na primer: Amazon Web Services (AWS), Google Cloud, Microsoft Azure, IBM Cloud i InView Web SCADA.

Osim od strane raznih provajdera prostranost MQTT-a se ogleda i u prisutnosti biblioteka za klijentske aplikacije. Na primer, *Paho* [8] je projekat razvijen od strane *Eclipse foundation* koji je razvio MQTT biblioteke za većinu danas upotrebljivanih programskih jezika.

Konkretan primer upotrebe MQTT protokola je prisutan od strane gigantske i opšte poznate korporacije *Facebook*. Naime, *Facebook* u svom proizvodu *Facebook Messenger* koristi MQTT protokol. Lakoća protokola obezbedila je ekonomičnost i nisku potrošnju baterije prilikom upotrebe na telefonu.

3.1.5 Mosquitto

MQTT predstavlja protokol komunikacije, dakle skup pravila komunikacije. Da bi se obezbedila komunikacija putem MQTT-a, pored klijenata, potrebno je obezbediti MQTT server.

Mnogo *cloud* provajdera nude svoj MQTT server kao gotovo rešenje. Problem ovakvih rešenja je što se najčešće plaćaju ili je besplatna upotreba, ako postoji u ponudi, prisutna sa raznim ograničenjima.

Pored gotovih rešenja već dostupnih na mreži postoje i implementacije servera. U pitanju je softver koji predstavlja MQTT server. Moguće ga je instalirati na razvojnom računaru u svrhu testiranja. Osim testiranja, isti softver je moguće koristiti i u produkciji ako se postavi na javno dostupnu adresu. Evo i primera ovakvih MQTT brokera: RabbitMQ, HiveMQ i Mosquitto. Ovo su samo neki primeri. Postoji još mnogo sličnih. Najčešće, svaki od njih po instalaciji sadrži i test klijent za *publish* i *subscribe*.

U ovom radu, u okviru eksperimenta, će se koristiti Mosquitto koji je proizvod već pominjane *Eclipse Foundation.* Broker je potpuno besplatno, dostupan na zvaničnom sajtu [9] i slobodan za svaku vrstu upotrebe. Upotreba je otvorena za većinu najčešće korišćenih operativnih sistemima. Moquitto implementira MQTT protokol u potpunosti i bez ograničenja.

3.2 Redis

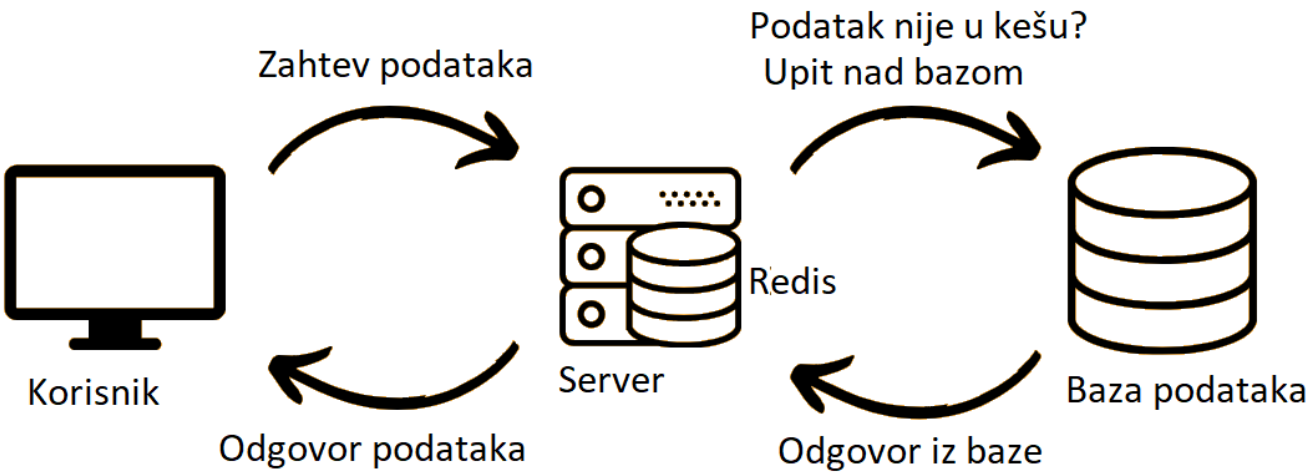
Redis je izvedenica od *Remote Dictionary Server* , što bi u bukvalnom prevodu značilo udaljen serverski rečnik (rečnik u smislu ključ-vrednost kolekcije). Redis je brzi serverski alat koji služi kao keš memorija, baza podataka i broker poruka [10].

Specifičnost ovog alata je što sve podatke čuva u memoriji i zbog toga je izuzetno brz. Takav rad softvera nosi i svoje mane u smislu ograničenja memorije, pa ga shodno tome treba i koristiti. Hard disk, odnosno SSD, ne koristi prilikom opsluživanja klijenata, te operacije čitanja i pisanja može mnogo brzo da izvršava.

Ipak, hard disk nije neupotrebljen kod ovog alata. Naime, Redis povremeno čuva sve podatke iz radne memorije na hard disk. Jedini razlog tome je sprečavanje gubitka podataka u slučaju restarta. Zbog ovakvog ponašanja Redis je spreman u okviru milisekundi da obrađuje zahteve klijenata.

3.2.1 Keš memorija

Redis se primarno i najčešće koristi kao keš memorija. Razni alati u raznim programskim jezicima nude vrlo jednostavno rešenje za keširanje podataka iz relacionih baza u Redis. Tada, umesto čitanja iz baze, podaci se mogu dobaviti iz Redis servera čime će odgovor biti mnogo brži. Osim baze podataka, keš može da se koristi i prilikom upotrebe nekih drugih resursa koji mogu da kasne, na primer, korišćenje drugih API-a.



Slika 3.3 Ilustracija komunikacije koristeći keš

Na slici 3.3 je prikazan primer komunikacije korišćenjem Redis keša. Dakle, na zahtev klijenta postoje 2 mogućnosti:

1. Traženi podatak je prisutan u keš memoriji. Tada server može pročitati podatak iz keš-a i odgovoriti. Ovaj slučaj će biti ekstremno brzo izvršen.
2. Traženi podatak nije prisutan u keš memoriji. Tada server zahteva podatak upitom nad bazom podataka. Po dobijanju odgovora iz baze, server će sačuvati traženi rezultat u keš, kako bi sledeći zahtev išao po postupku navedenom pod 1, a potom i odgovoriti klijentu na zahtev.

Čuvanje podataka u formatu ključ-vrednost je osnovna upotreba Redis servera kao keš memorije. Kao što i skraćenica govori: serverski rečnik, savršena terminologija za ključ-vrednost arhitekturu. Redis svaku vrednosti koju drži u memoriji, čuva direktno kao ključ vrednost par. Ovakvo pravilo asocira na to da je vrednost prostog tekstualnog tipa, međutim, prednost Redisa se ogleda baš u mogućnosti čuvanja različitih tipova podataka u kešu. Zbog toga se Redis predstavlja i kao baza podataka, a ne samo keš memorija. Vrednosti mogu biti neki od tipova:

- String – tekstualni ili binarni zapisi do 512MB
- List – kolekcija stringova

- Skup – skup podataka sa podržanim operacijama nad skupovima (presek, unija, i razlika u odnosu na drugi skup)
- Sortiran skup
- Heš mape – mogu se čuvati osobine i vrednosti. Ovim tipom vrednosti je moguće čuvati cele objekte (entitete)
- Bitmape

3.2.2 Redis pub/sub

Pub/sub, skraćenica od *publish-subscribe*, predstavlja implementaciju *publish-subscribe* paradigme kod Redis servera. Kao što je prethodno opisano, uloge klijenata su *publisher* i *subscriber*. Takođe je moguće da isti klijent bude oba od navedenog. Komunikacija je po prethodno opisanom pravilima ove komunikacije: *publisher* šalje poruke na temu a broker, u ovom slučaju Redis, raspoređuje i dostavlja svim *subscriber* komponentama koje su se prethodno pretplatile na tu temu.

Zamenski znakovi postoje i ovde, kao i kod MQTT-a. Međutim, ovde ne postoje nivoi u okviru definisanja tema, već je sve kao jedan string. Zamenski karakter je * i predstavlja zamenu za više karaktera. Pa tako, ukoliko se klijent pretplati na temu *novosti** dobijaće poruke i sa tema *novostisrbije, novosti.zabava-dana* a i *novosti-danas-sutra*.

Iako je Redis jako brz, postoji jedna mana koja mu je neoprostiva. Naime, za razliku od MQTT-a, Redis se koristi unutar zatvorene mrežne arhitekture, dakle lokalno. Redis nije preporučljivo i ne koristi se u distribuciji fizički udaljenih sistema. Razlog je prost: ovaj alat nije namenjen za to, ne podražava ni bezbednu razmenu podataka, niti je optimizovan za komunikaciju putem interneta, niti podržava autentifikaciju. Postoji mogućnost zaštite lozinkom, ali to je sve, ne podržava ni višestruke korisnike. Redis ne podržava sigurnu komunikaciju čime se mnogo ograničava oblast upotrebe.

3.2.3 Klaster

Redis podržava klastere. Dakle, moguće je podići više Redis servera koji rade zajedno, kao jedan. Po dokumentaciji [11] ukoliko je povezano više servera u klaster, oni će obezbediti da svaka poruka poslata na bilo koji od servera, bude poslata svim pretplatnicima na svim serverima. Trenutna implementacija samo šalje svima dalje podatke, bez optimizacija i usavršavanja komunikacije. Na tome se tek planira raditi. Ovakav stav naveden u dokumentaciju ne uliva mnogo nade za upotrebu Redis klastera u svrhu brokera.

Prednosti upotrebe klastera kod Redis servera su uočljive kada se koristi kao keš memorija. Naime, tada će svaki server moći da koristi svu radnu memoriju za keš, te se sistem keširanja ne ograničava na količinu memorije jednog hardvera, već je moguće povezati ih u klaster te zajedno deljeno koristiti radnu memoriju više hardverskih jedinica. Međutim, tema ovog rada se fokusira na *pub/sub* deo Redis servera, te neće biti detaljnija obrada ove teme.

3.3 Kafka

Kafka predstavlja distribuiranu striming arhitekturu za obradu podataka u realnom vremenu [12]. Razvijena je od strane LinkedIn kompanije 2011 godine za interne potrebe. Deo je *Apache Software Foundation* [13].

U srži, Kafka predstavlja *publish-subscribe* broker poruka. Međutim, razvijan je namenski da reši sledeće probleme: brzina, skalabilnost, trajnost i netoleranciju greške. Karakteriše ga visok protok podataka, pouzdanost, skalabilnosti i replikabilnost. Spajanjem navedenih karakteristika i prednosti koje nosi, Kafka je postao alat za striming događaja.

Za razliku od Redis i MQTT servera, Kafka koristi hard disk i mnogo radi sa istom kako bi obezbedila sve zahteve.

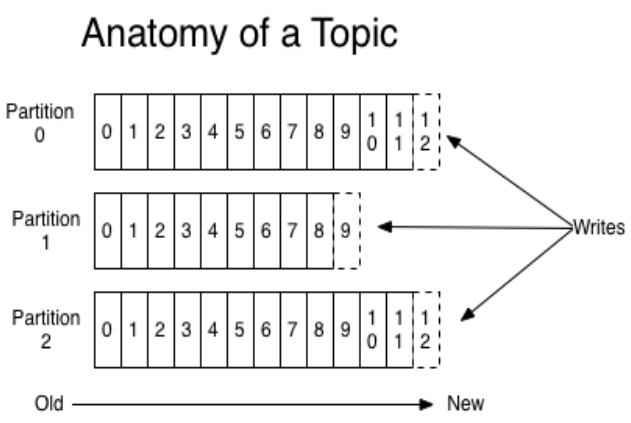
Terminologija kod Kafke se pomalo razlikuje: umesto *publish* i *subscribe*, koriste se termini *produce* i *consume* (proizvodi i upotrebljava). Pa tako, uloga u komunikaciji nije *publisher* već *producer*, i s druge strane, nije *subscriber* već *consumer*. Radi manje konfuzije u okviru ovog rada će biti korišćeni isti termini kao do sad, dakle *publish-subscribe*.

3.3.1 Particije, ofseti i komit

Ključna razlika između drugih brokera i Kafke se ogleda u tretiranju poruka. Dok ostali brokeri imaju kratko memorisanje ili nemaju uopšte, Kafka dugotrajno vodi računa o svim porukama i obezbeđuje da svaka poruka sigurno stigne na svaku lokaciju, bez mogućih gubitka podataka. Mogućnost čuvanja podataka je dugotrajna, ograničava se u kapacitetima hard diska računara (podrazumevano čuvanje poruka je 7 dana, ali postoji mogućnost čuvanja podatak večno).

Da bi se objasnilo tačno kako funkcioniše, neophodno je prethodno uvesti termine specifične za ovaj alat. Prilikom kreiranja teme, neophodno je navesti koliko particija ima. Kada *publisher* poruke šalje na temu ona se rasporedi u jednu od particija. *Subscriber*, prilikom pretplate, osim teme navodi i particiju koju prati. Dakle, Kafka radi po automatici raspoređivanje poruka u cilju rasterećenja celina distribuiranog sistema. Za korišćenje osnovnog *publish-subscribe* modela dovoljno je napraviti temu sa 1 particijom i sve *subscriber* komponente će dobijati sve poruke.

U okviru svake particije teme, za svakog *subscriber* klijenta se čuva ofset. Ofset predstavlja pokazivač na poslednju poruku koju je *subscriber* uspešno preuzeo. *Subscriber* po inicijalnoj pretplati prima redom sve poruke (osim ako navede drugačije); po prijemu poruke šalje komit signal na Kafka server kao naznaku da je poruka obrađena te da Kafka pomeri ofset. Jako bitno je za uočiti da se unutar svake particije čuvaju ofseti za svakog *subscriber* klijenta.



Slika 3.4 Particije, ofseti i komit

Na slici 3.4 je pokazan primer jedne teme sa 3 particije. Na tu temu je poslato ukupno 33 poruke (12 poruka na particiji 0, 9 poruka na particiji 1 I 12 poruka na particiji 2). Nove poruke se raspoređuju bez specijalnog pravila. *Subscriber* se pretplati na temu i određenu particiju, te ima 3 načina preuzimanja prethodnih poruka:

1. *Earliest* – da čita sve poruke od početak particije
2. *Latest* – da čita samo poruke koje stignu nakon njegove pretplate
3. *Offset specific* – da prilikom pretplate navede i tačno ofset odakle želi da dobija poruke.

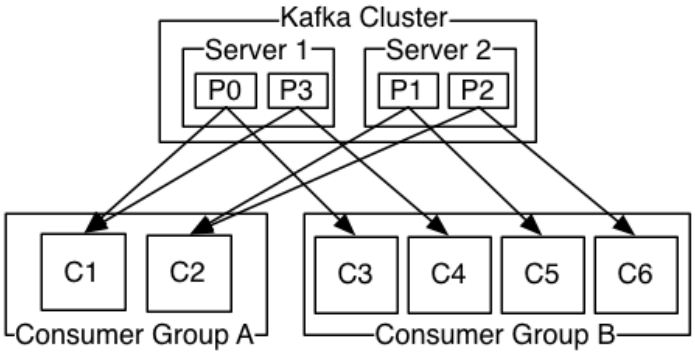
Ovakvim mehanizmom se dobija nešto što ni kod MQTT ni kod Redis servera nema u mogućnostima a to je čitanje istorijskih poruka kao i garancija da *subscriber* klijenti dobiju sve poruke, čak i one koje su poslate za vreme kada *subscriber* nije bio uopšte u komunikaciji.

Kafka je dizajnirana da radi raspodelu posla, u smislu obrađivanja poruka. Do ovakve upotrebe dolazi kod horizontalnog skaliranja. Kada jedan deo sistema postaje preopterećen, podigne se još jedan isti na novoj hardverskoj jedinici. Ukoliko bi svaki od njih dobijao sve poruke za obrađivanje, izgubio bi se značaj, jer bi tada postojala 2 opterećena sistem. Naravno, postoji mogućnost aplikativnog rešavanja raspodele obrađivanja poruka, ali Kafka nudi gotovo rešenje za to. Raspodelom tema po particijama obezbeđuje ravnomernu raspoređenost poruka, te ravnomerno opterećenje istih servisa. Ukoliko se 2 ista servisa pretplate na istu temu a različite particije, ravnomeran raspored poruka će biti usmeren na svakog od njih.

Kako bi se podigao nivo apstrakcije o particijama i time uprostila logika *subscriber* komponente postoji *subscriber* grupisanje: takozvanim *consumers group*. Prilikom pretplaćivanja na Kafka temu nije neophodno navesti particiju, ali jeste obavezno navesti *subscriber* grupu. To predstavlja običan tekst koji služi za identifikaciju paralelnih delova sistema koji rade isti posao. Pa tako, u primeru iz prethodnog pasusa, oba servisa će se pretplatiti na temu kao ista *subscrdiber* grupa te će poruke biti ravnomerno raspoređene.

Koristeći grupe, klijenti ne vode računa o broju particija već to radi sam Kafka server. Na primer, ukoliko postoje 4 particije, *subscriber* klijenti bi se trebali pretplatiti na po 2 particije (mada se preporučuje da broj particija pri inicijalizaciji bude jednak broju *subscriber* klijenataa baš iz ovog razloga). Koristeći istu grupu i pretplatom 2 servisa sa istom grupom, Kafka će automatski da ih prijavi na po 2 particije.

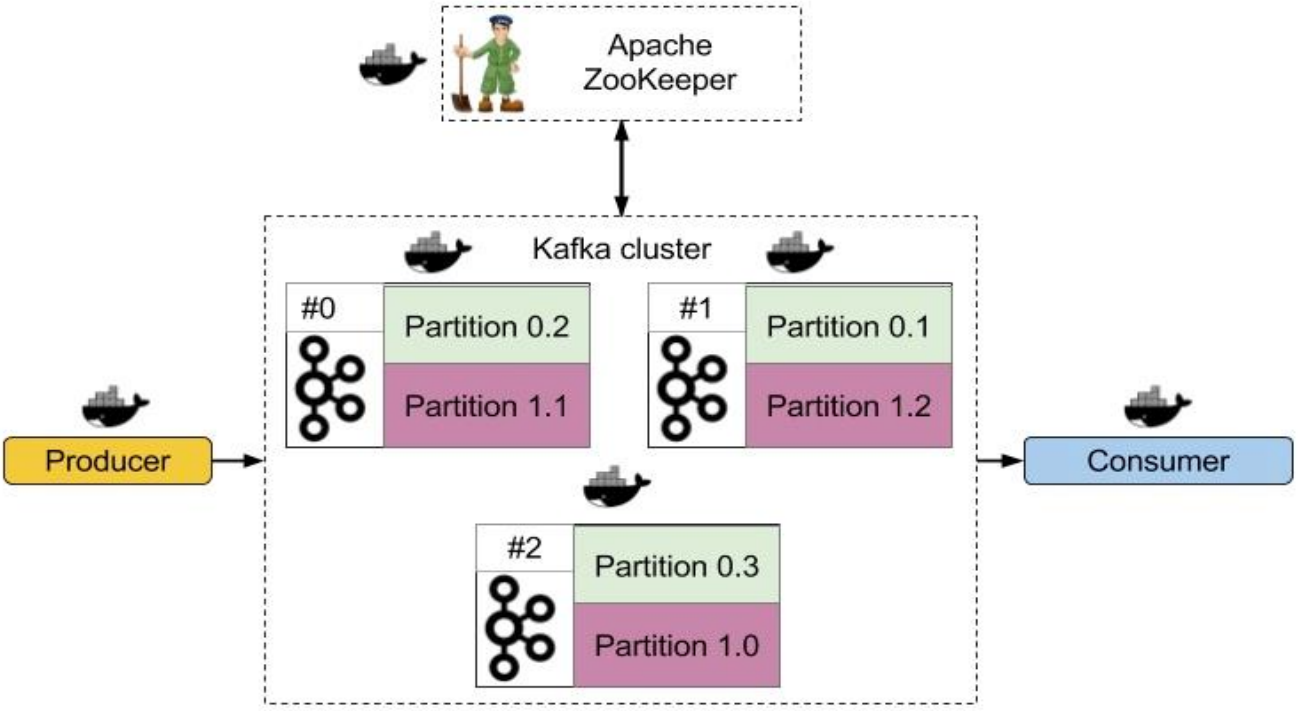
Na slici 3.5 je dat primer teme sa 4 particije i 2 *subscriber* grupe (gornji deo slike je vezan za klaster, o tome u nastavku teksta. Sada je dovoljno da se gornji deo slike posmatra kao jedan Kafka broker). Na brokeru je prikazana tema sa 4 particije. *Subscriber* grupa A sadrži 2 *subscriber* klijentaa, pa će po pretplati svaki *subscriber* primati poruke sa po 2 particije. U grupi B je predstavljeno 4 *subscriber* klijenta, pa će tako svaki *subscriber* biti zadužen za po jednu particiju. U slučaju nedeljivih broja particija i *subscriber* klijenata unutar grupe neki od *subscriber* klijenata će dobiti više particija od drugih (razlika u broju zaduženih particija ne sme da bude veća od 1)



Slika 3.5 Ilustracija raposdele poruka po subscriber grupama

3.3.2 Klaster

Kafka podržava klastere. Ne samo da podržava već se i preporučuje korišćenje klastera (skoro sve jedan primer i vodič je dat bar sa 2 Kafka brokera) jer su prednosti ovog alati izraženi prilikom upotrebe klastera. To znači da podizanje Kafka brokera podrazumeva podizanje više od jednog servera koji su povezani i rade zajedno. Kafka serveri unutar klastera ne komuniciraju direktno. Spona između severa je pomoćni alat *Apache Zookeeper*.



Slika 3.6 Kafka klaster ilustracija

Konfigurisanje klastera je intuitivno i jednostavno. U konfiguraciji samog Kafka servera se navodi adresa do *Zookeeper* servera. *Zookeeper* je taj koji održava konzistentnost između klastera. Klijenti prilikom konekcije sa klasterom navode jednu ili više adresa Kafka servera. To znači da je moguće povezati se uvek na jedan server. Po konvenciji implementacije Kafka klijenata uvek je moguće navesti adrese svih Kafka servera. S druge strane, Kafka klasterima se može pristupati i putem *Zookeeper* servera. Moguće je prosto umesto adresa Kafka servera navesti adresu Zookeeper servera. Na slici 3.6 je predstavljena ilustracija arhitekture Kafka klastera.

Unutar Kafka klastera politika raspodele particija tema je ujednačena raspodela po različitim serverima. Ukoliko je tema napravljen sa više od jedne particije, particije će biti ravnomerno raspodeljene unutar klastera na različite servere.

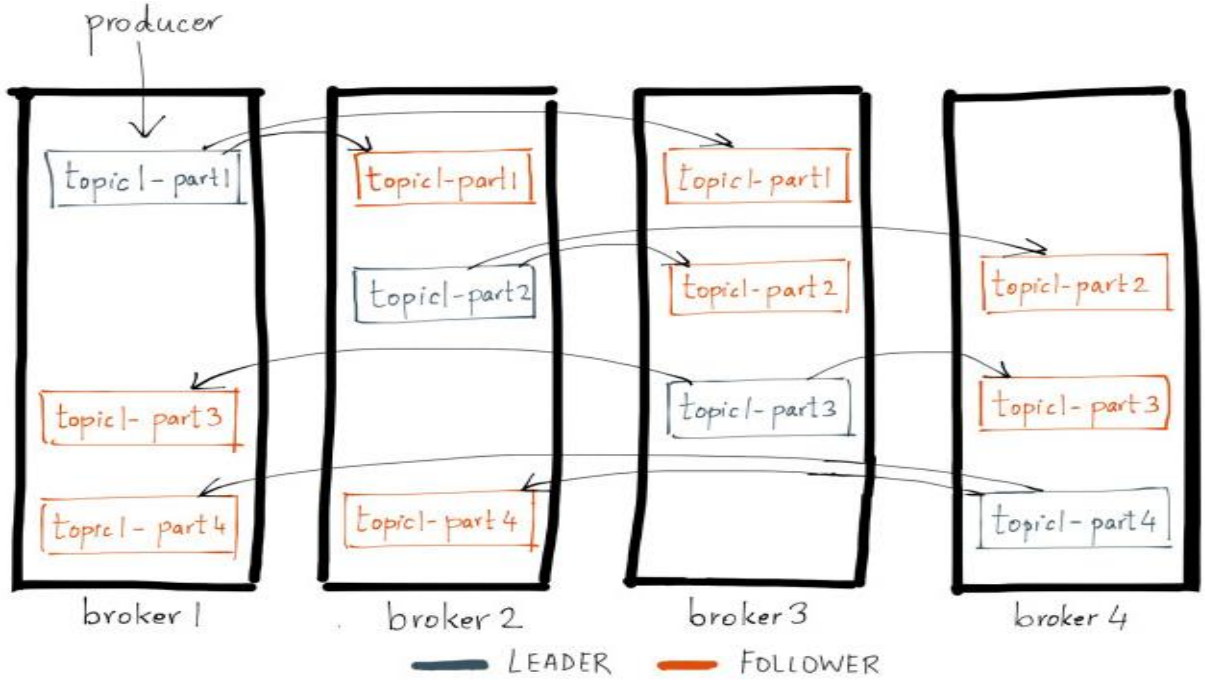
Klaster upotreba predstavlja horizontalno skaliranje Kafka servera. Time se značajno podiže nivo dostupnost i pouzdanost, kao i kapacitet obrade podataka. Iz ovih razloga Kafka predstavlja visoko pouzdan broker sa jako velikim kapacitetom protoka poruka.

3.3.3 Replikacija

Prilikom kreiranja teme, pored broja particija, neophodno je navesti i faktor replikacije. Taj broj označava na koliko Kafka servera unutar klastera će se čuvati kopije svakog od particija. Preciznije, koliko će kopija podataka postojati unutar klastera.

U cilju sigurnije dostupnosti podataka, svaka poruka se kopira na više servera. Problem se javlja kod mnogo podataka, jer višestruko čuvanje istih predstavlja problem po količinu zauzete memorije. Međutim, Kafka koristi hard disk, tako da su ograničenja mnogo manja. Pored toga, konfigurabilna je i količina čuvanja podataka kao i broj čuvanih kopija: *faktor replikacije*.

Definisanjem prioriteta, ko čuva originalan podatak a gde se on kopira, serveri dobijaju različite uloge: *leader* i *follower*. *Leader* jedne particije je server unutar klastera koji čuva originalni podatak teme i primarno vodi računa o njoj u smislu ažuriranja ofseta. *Follower* serveri su serveri na kojima su kopirani podaci od *leader* servera. Svaku promenu na particiji ažurira *leader* pa potom podatak biva potvrđen i na *follower* servere. S obzirom da *leader* i *follower* servera može biti ukupno koliko unutar klastera ima brokera postoji ograničenje gornje granice faktora replikacije. Nema smisla da se neki podaci 2 puta čuvaju na istom brokeru. Tako je maksimalan faktor replikacije jednak broju brokera unutra klastera.



Slika 3.7 Leader i follower serveri replikacije unutar Kafka klastera

Na slici 3.7. je ilustrovan sledeći primeri: klaster sa 4 Kafka brokera; tema sa 4 particije i faktor replikacije 3. Pošto je tema sa 4 particije, a klaster je od 4 brokera, raspodela je jasna, svaki server po jednu particiju, kao što i jeste predstavljeno na ilustraciji: crni pravougaonici. Faktor replikacije je 3, što znači da se svi podaci čuvaju na 3 servera, odnosno na *leader* i 2 *follower* servera. *Follower* serveri svake particije teme su na ilustraciji prikazani narandžastom bojom. Na particiji gde je *leader* broker 1, *follower* serveri su brokeri 2 i 3; kada je *leader* broker 3,*follower* serveri su brokeri 4 i 1, itd., kao što je ilustrovano na slici.

Organizovanje particija i raspored uloga server Kafka radi automatski. Na korisniku je da po kreiranju teme navede tražene parametre. Primer kreiranja teme je naveden na listingu 3.1. U konkretnom primeru kreiranja su navedeni parametri sa slike 3.7.

```
bin/kafka-topics.sh --create \
   --zookeeper localhost:2181 \
   --replication-factor 3 \
   --partitions 4 \
   --topic topic1
```

Listing 3.1 Kerianje teme

Kada je tema kreirana postoji komanda za tzv. opis teme, odnosno prikaz rasporeda particija i replikacija unutar klastera. Na listinu 3.2 je prikazan primer opisa kreirane teme iz prethodnog primera. Ispis Kafka opisa je ekvivalent vizualizovanom primeru na slici 3.7.[1]

```
Topic: topic1   PartitionCount:4      ReplicationFactor:3   Configs:
     Topic: topic1 Partition: 1 Leader: 1 Replicas: 1,2,3 Isr: 1,2,3
     Topic: topic1 Partition: 2 Leader: 2 Replicas: 2,3,4 Isr: 2,3,4
     Topic: topic1 Partition: 3 Leader: 3 Replicas: 3,4,1 Isr: 3,4,1
     Topic: topic1 Partition: 4 Leader: 4 Replicas: 4,1,2 Isr: 4,1,2
```

Listing 3.2 Opis Kafka teme

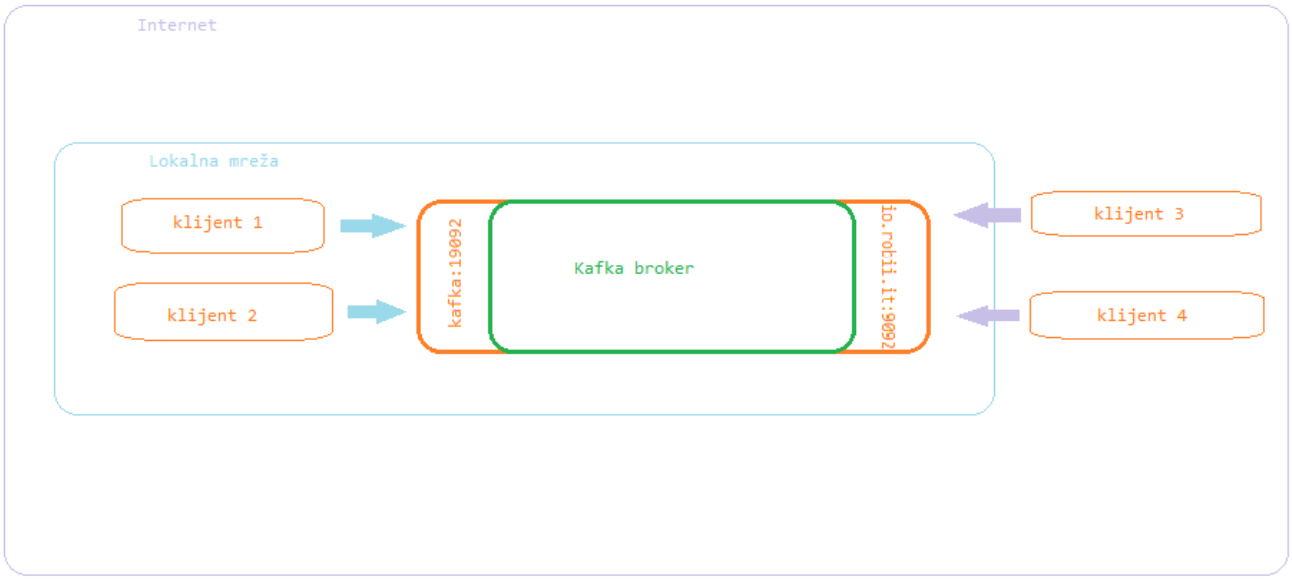
3.3.4 Interfejs ka klijentima

Kafka pruža bezbednu komunikaciju. Razmena kriptovanih podataka na bazi SSL-a je moguće konfigurisati. Podrazumevano ponašanje ne uključuje SSL. Takođe, autentifikacija klijenata je konfigurabilna bez ograničenja broja kombinacija korisničkih imena i lozinki (za razliku od Redis servera). Kafka podržava i autorizaciju, odnosno otvorena je mogućnost konfigurisanja koji korisnici koje topike smeju da pišu a sa kojih smeju da čitaju.

Gore navedeno podešavanje se vrši kroz konfiguraciju brokera i naziva se protokol osluškivača. Osluškivač predstavlja interfejs ka Kafka klijentu. Osim različitih protokola Kafka nudi mogućnost i definisanja višestrukih osluškivača. Na taj način moguće je definisati različite osluškivače klijenata po različitim protokolima.

Različiti osluškivači su jako korisna stvar kada se Kafka broker koristi dvostruko: i za komunikaciju unutar mreže, lokalnu, i putem interneta, eksternu. Lokalna komunikacija može da bude prosta bez bilo kakve bezbednosti jer su podaci u lokalnoj mreži i ne preti opasnost po bezbednost. S druge strane, eksterna komunikacija je nužno obezbeđena odgovarajućom sigurnošću kako bi se zaštitilo od svih vidova napada i krađe podataka. Osluškivače je dozvoljeno imenovati prilikom konfigurisanja.

[1] Listing 3.2 je korigovan da bude identičan sa slikom 3.7. Realan ispis je nula indeksiran. Umesto brokera 1,2,3 i 4 realno piše 0,1,2 i 3. Ova korekcija je urađena radi izbegavanje konfuzije prilagođavanjem slici 3.7.



Slika 3.8 Ilustracija višesrukih osluškivača

Na slici 3.8 je dat primer Kafka brokera sa 2 osluškivača. Jedan je za lokalnu mrežu i dostupan je na adresi *kafka* i portu 19092. Drugi osluškivač je konfigurisan za spoljašnju upotrebu i dostupan je na adresu *io.robii.it* i portu 9092. Klijenti 1 i 2 su nužno u istoj mreži kao i sam broker, dok su klijenti 3 i 4 bilo gde samo je uslovljeno prisustvo interneta.

3.4 Poređenje

Nakon upoznavanja sa sva 3 ciljana brokera ovog rada, a pre eksperimentalnog dela, moguće je uraditi pregled razlika brokera.

3.4.1 Skladište manipulacije podataka

Redis podacima manipulišu u radnoj memoriji i povremeno vrši čuvanje na hard disk radi sigurnosti u slučaju restarta. Kafka sve podatke čuva na hard disk. Zbog toga je za očekivati da će Kafka imati donekle sporiji odziv u odnosu na Redis. Očekivana veća brzina daje prednost za Redis. Kod MQTT-a nije strogo definisano gde se manipuliše podacima jer je MQTT samo protokol. *Mosquitto,* kao implementacija MQTT brokera, podacima manipuliše u radnoj memoriji.

S druge strane, način čuvanja podataka Kafka brokeru pruža prednosti kao mogućnost istorijskog konzumiranja podataka, višestruko konzumiranje istih podataka kao i sigurniji mehanizam protiv gubitka podataka.

Mosquitto na hard disku čuva poruke naznačene sa zadržavanjem (*retain*) da u slučaju restart može da prosledi poslednje vrednosti. Osim ovog ograničenja MQTT broker nema potrebu za bilo kakvim čuvanjem podataka na tešku memoriju.

3.4.2 Šabloni tema

MQTT podržava najnapredniji i najintuitivniji način šablonskih tema. Nivoima i mogućim zamenskim karakterima za nivoe MQTT pruža široke mogućnosti upotrebe. Redis, slično, podržava pretplatu na teme po šablonu samo bez mogućnosti uvođenja nivoa, već se prepušta mašti korisnika da organizuje teme po volji.

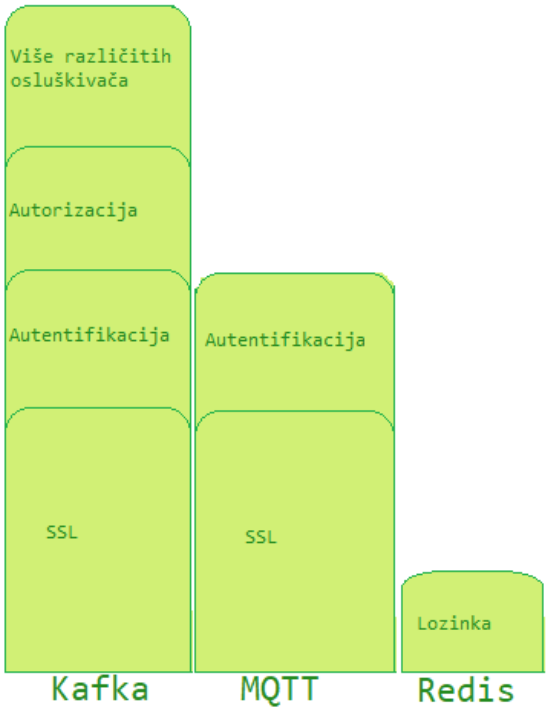
Kafka, s druge strane, ne podržava pretplatu na teme po šablonima, ali omogućava značajne prednosti uravnoteženog raspoređivanja poruka po particijama tema.

Različite mogućnosti upotrebe su smernica korisnicima da odaberu odgovarajući alat shodno potrebama distribuiranog sistema. Svaki sistem ima svoje karakteristike i specifičnosti te će različite mogućnosti upotrebe više ili manje odgovarati različitim distribuiranim sistemima.

3.4.3 Bezbednost

Po pitanju bezbednosti kod ova 3 brokera postoji jednoznačno linearno poređenje:

- Kafka nudi najširi spektar mogućnosti konfigurisanja bezbednosti i visoko napredne mogućnosti
  - SSL
  - Autentifikacija
  - Autorizacija
  - Mogućnost višestrukih osluškivača s različitom definicijom
- MQTT nudi manju širinu ali zadovoljavajući nivo bezbednosti:
  - SSL
  - Autentifikacija
- Redis ne nosi atribut mogućnosti bezbedne komunikacije. Jedino što nudi je:
  - Lozinka



Slika 3.9 Nivoi bezbednosti

Na slici 3.9 je prikazana ilustracija prisutnosti različitih načina bezbednosti u cilju slikovitog poretka

3.4.4 Skalabinost

Na polju horizontalne skalabilnosti Kafka nudi najjaču podršku. Razlog je što je Kafka pravljena sa ciljem da predstavi skalabilno rešenje brokera poruka.

Redis podržava klaster upotrebu, ali nije mu to primarna upotreba i nije optimizovan za to.

MQTT ponovo zavisi od implementacije samog servera. *Mosquitto* ne podržava skalabilnost. Neke druge implementacije pak nude mogućnost, ali nije u opisu MQTT protokola i ne poklapa se sa glavnim atributom MQTT: *lightweight*.

Odabir se ponovo svodi na pitanje potreba distribuiranog sistema. Za distribuirane sisteme sa visokim prometom i potrebama za obradom velike količine podataka se preporučuje upotreba Kafke, dok za neko lakše i brže rešenje alata za komunikaciju će potpuno korektno poslužiti i Redis ili MQTT.

3.4.5 Mehanizam pretplate podataka

Bitno za napomenu je mehanizam pretplate kojim *subscriber* komponenta dobija poruke. Naime, za Kafka server može da se kaže da je maskiran *publish-subscribe* šablon. Za razliku MQTT i Redis servera gde *subscriber* dobija obaveštenje od servera o novoj poruci, kod Kafke se radi takozvani *pulling* mehanizam, odnosno mehanizam povlačenja poruka.

Ključna razlika se ogleda u tome što Redis i MQTT uspostave konekciju sa brokerom, pretplate se na temu i čekaju. Server je taj koji pokreće akciju slanjem poruke. S druge strane, Kafka klijent se pretplati na temu i nužno ciklično proziva server sa pitanjem: „da li ima novih poruka?“

U zavisnosti od biblioteke koja se koristi, moguće i vrlo je verovatno da se i kod MQTT i Redis klijenata u okviru implementacije klijentske biblioteke takođe vrši *pulling* mehanizam, ali se to nužno sakriva od korisnika. Kod Kafka brokera je drugačija politika i očekuje se da korisnik prilagodi implementaciju tako da uzimanje poruka s Kafka brokera bude po ovom mehanizmu.

3.4.6 Gubitak podataka

MQTT pomoću različitih *QoS*-a pruža mogućnost da sam korisnik da sam korisnik odabere nivo potrebe za osiguravanjem poruka, što je najbolje pokrivanje problema gubitka podataka.

Redis je dizajniran za upotrebu u lokalnoj mreži. Kada je u pitanju lokalna komunikacija očekivana je stabilnost veze i minimalne mogućnosti gubitka podataka. Iz tih razloga, Redis nema specifične mehanizme zaštite od gubitka podataka.

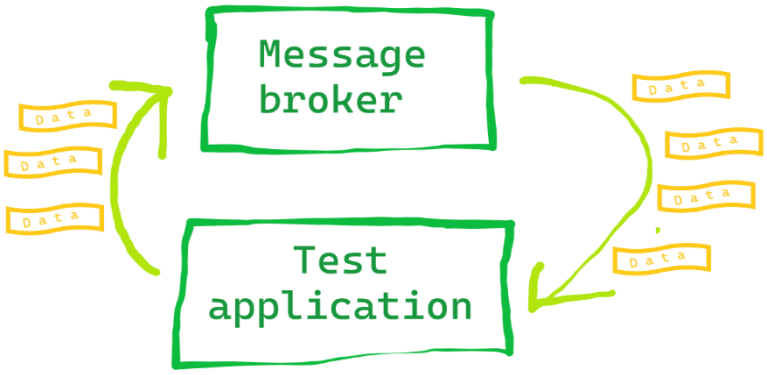
Kafka radi po principu da svaka preuzeta poruka mora da se potvrdi komitom. Ofset se pomera tek po komitu od strane klijenta. Na taj način, gotovo ne postoji mogućnost gubitka podataka.

Poglavlje 4

Arhitektura i alati eksperimenta

Poređenje brokera urađeno u poglavlju 3 je na teorijskoj osnovi i predstavlja razlike u brokerima po dokumentaciji. Srž ovog rada je eksperimentalno poređenje brokera, odnosno odmeravanje performansi u što je realnije mogućoj simulaciji realnog okruženja. Ovo poglavlje opisuje sam eksperimentalni deo, kako je zamišljen, kako struktuiran, kako implementiran i koji u su pomoćni alati korišćeni.

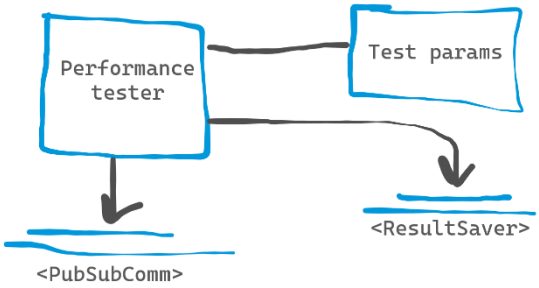
Praktični deo ovog rada obuhvata pisanje koda alata za testiranje performansi. Kako bi se razmenjivali podaci kroz broker potrebno je implementirati 2 uloge: *publisher* i *subscriber*. Pošto je u pitanju simulacija i testiranje, nema prepreke da se obe uloge implementiraju u okviru iste aplikacije. Naime, posmatraju se karakteristike brokera, a ni na jedan od njih neće uticati ukoliko *publisher* i *subscriber* bude ista aplikacija. Odnosno, bez obzira što je polazna i odredišna tačka svakog podataka ista, podaci moraju da prođu putanju do brokera, procesiranje od strane brokera i tek onda do odredišta, kao što je ilustrovano na slici 4.1.



Slika 4.1 Tok podataka testiranja brokera

Kako bi se otklonile mogućnosti razlika u kodu, te i upotrebljiviji i produktivniji kod napisao, aplikacija se razdvaja na različite nivoe apstrakcije. Viši nivo test aplikacije se bavi parametrisanjem okruženja, izvođenjem samog testa u smislu slanja poruka i osluškivanja i vršenje potrebnih merenja. Niži nivo aplikacije će se baviti samom implementacijom komunikacije sa svakim od brokera. Takođe, u niži nivo spada i implementacija čuvanja rezultata.

Na slici 4.2 je prikazana skica arhitekture višeg nova aplikacije. Ključ ovog dela se nalazi u modulu *PerformanceTester*. On dobija parametre testa, koristi *PubSubComm* interfejs, kao i *ResultSaver*. Na ovaj način je softver za testiranje podeljen u 3 dela sa minimalnom zavisnošću jednom od druge. Time je i odgovornost po celinama jasno razdvojena.



Slika 4.2 Viši nivo test okruženja

PubSubComm predstavlja interfejs komunikacije po *publish-subscribe* šablonu. Definiše obe uloge šablona čime je zahtevana implementacija samo komunikacije takva da ista aplikacija bude i *publisher* i *subscriber*. Na listingu 4.1 je prikazan kod *PubSubComm* interfejsa.

```
public interface PubSubComm extends AutoCloseable {
    public boolean isConnected();
    public boolean connect();
    public void publish(String topic, String message);
    public void subscribe(String topic, Consumer<String> onMessage);
}
```

Listing 4.1 PubSubComm intefejst

U listi metoda nije navedena metoda *disconnect* iz razloga što se prekid komunikacije radi u *close* metodi definisanom u okviru *AutoCloseable* interfejsa iz paketa *java.lang*. Nasleđivanjem *AutoCloseable* interfejsa je omogućeno da se koristi *try-with-resources* funkcionalnost [14].

4.1 Implementacija komunikacije

Sva 3 tipa komunikacije imaju svoju klasu koja implementira *PubSubComm* interfejs. Kreiranje instanci vrši putem *Factory method* dizajn paterna.

Sve 3 implementacije konekcije se oslanjaju na neku biblioteku. Aplikacija koristi *maven*[15] te su sve korišćene biblioteke uvezane pomoću njega.

4.1.1 MQTT

MQTTPubSubComm je klasa koja implementira *PubSubComm* interfejs kao MQTT klijent. Koristi se biblioteka iz projekta *Paho* o kojoj je pisano u ovom radu u okviru opisivanje MQTT-a (sekcija 3.1.4).

Upotreba biblioteke je vrlo jednostavna i intuitivna. Koristi se *MqttClient* klasa.

Karakteristično za MQTT jeste potreba za definisanjem QoS-a (sekcija 3.1.1). S obzirom da je to specifično za MQTT, ne i za ostale brokere, taj parametar je vezan samo za ovu klasu.

4.1.2 Redis

RedisPubSubComm klasa predstavlja *PubSubComm* kao Redis klijent. Korišćena biblioteka je takozvana *Jedis.* Ova biblioteka je preporučena sa zvaničnog sajta Redis projekta [16].

Interesantno za Redis je što pretplata na temu zahteva izvršavanje u odvojenoj niti (*thread*) jer će u suprotnom trenutna nit ostati zaglavljena i osluškivati poruke.

4.1.3 Kafka

KafkaPubSubComm je klasa Kafka klijenta koja takođe implementira *PubSubComm*. Biblioteka koja se koristi je izdata od strane fundacije *Apache* [13] koja je ujedno i potpisnik autor samog Kafka brokera.

Implementacija Kafka klijenta je složenija od prethodna 2 opisana. Naime, Kafka klijent zahteva precizno definisanje raznih konfiguracionih parametara kao što su npr. serijalajzer, deserijalajzer, identifikacija klijenta, identifikacija *subscriber* grupe i sl. Osim više parametara, Kafka je karakteristična i po mehanizmu pretplate. Neophodno je implementirati prethodno opisan *pulling* mehanizam.

4.2 *ResultSaver* interfejs

ResultSaver predstavlja deo aplikacije koji je zadužen da pamti rezultat tokom samog testa te da ga kasnije sačuva radi analize. On vodi računa kada je test pokrenut kako bi evidentirao hronološki tok testa. On pamti svaki rezultat prijavljen od strane testera performansi.

Interfejs definiše *done* metod, koji označava da je test završen i rezultat može da se sačuva po pravilu implementacije. Tokom vršenja testa se rezultati dodaju samo u memoriju. Ovakva organizacija je postavljena u cilju minimalnog opterećenja resursa za vreme trajanja testa.

4.2.1 JSONResultSaver

JSONResultSaver je klasa koja implementira *ResultSaver.* Na kraju testa sve podatke čuva u JSON file.

4.2.2 DbResultSaver

DbResultSaver je takođe klasa koja implementira *ResultSaver*. Ona je zadužen da rezultat sačuva u relacionu bazu podataka. Ovaj rad koristi MySql bazu podataka. Za konekciju sa bazom i mapiranje entiteta korišćen je *Hibernate*.

Rezultati se čuvaju u okviru 3 entiteta. Dijagram klasa koji se čuvaju u bazu su prikazani na dijagramu 4.1.

DbTestParams predstavlja parametre testa. Za svaki *DbTestParam* se vežu tačno 3 *DbTestRun*-a jer *DbTestRun* predstavlja izvršenje teste na jednom tipu brokera, a rad proverava 3 tipa. Za *DbTestRun*, odnosno za primenu parametara testa na tip brokera se veže lista *DbTestResulta.*

Dijagram 4.1 Dijagram klasa entite za čuvanje rezulata

4.3 Tester performansi

Tester perfomasni predstavlja samu srž ove aplikacije. Razvijen je algoritam za ovaj rad koji na osnovu parametara vrši simulaciju poruka realnog okruženja. Ovaj tester koristi sledeće ulazne parametre:

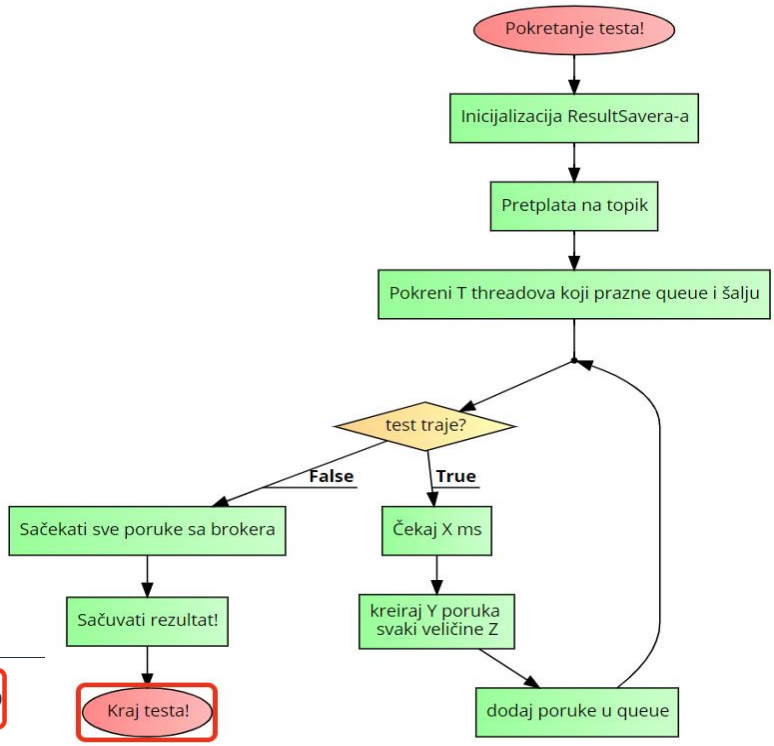
- S - dužina trajanja testa (u sekundama) -koliko dugo će se slati poruke na broker
- X - Interval između 2 slanja (milisekunde) – dužina pauze između 2 slanja
- Y – broj poruka koji će se poslati odjednom
- Z – veličina svake poruke (u bajtima)
- T – broj niti sa koliko će se istovremeno slati

Parametri su imenovani kako bi se jedan test mogao sažeti u jedno rečenicu:

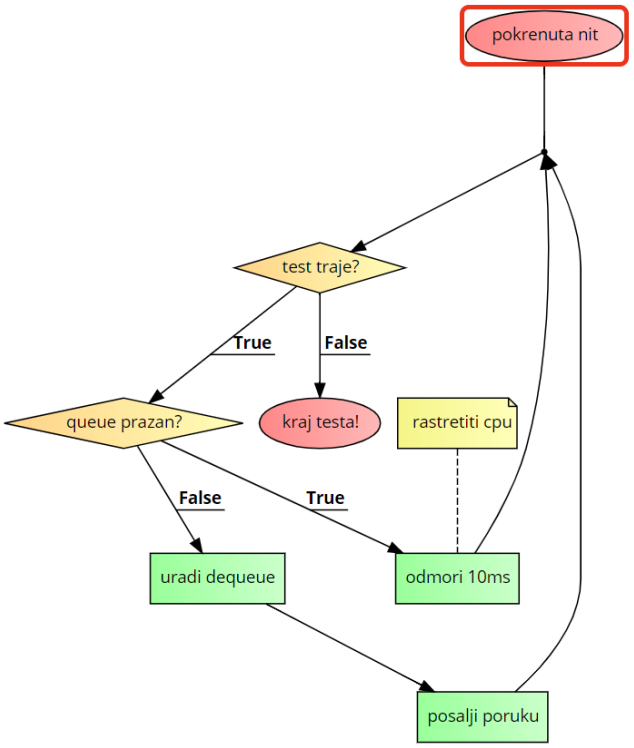
*Narednih **s** sekundi šalji svakih **x** milisekundi **y** poruka veličine **z** bajtova paralelno na **t** niti.*

Radi realizacije paralelnog slanja algoritam se izvršava na više od 1 niti, i to tačno **t+1** niti. Glavna nit testa služi za kreiranje poruka, ostale niti rade slanje na broker. Niti imaju deljeni resurs u vidu reda opsluživanja, *queue*-a. Glavna nit testa dodaje poruke shodno ostalim parametrima u *queue*. Pomoćne niti rade konstantno i prazne *queue* slanjem poruka na broker.

Algoritam glavne niti testa je prikazan na dijagramu 4.2. Dakle po pokretanju poziva se inicijalizacija nad *ResultSaver*. Zatim se uradi pretplata na temu kako bi se obrađivale dobijene poruke sa brokera. Potom se pokreću **t** komada pomoćnih niti. Sledećim korakom započinje test. Dok ne istekne **s** sekundi u petlji se rade sledeća 3 koraka: nit odmara **x** milisekundi; zatim kreira **y** poruka svaki veličine poruka svaki veličine **z** bajtova, te se te poruke dodaju u *queue*. Svaka iteracija uključuje ova 3 koraka do isteka testa. Po isteku test se sačeka da stignu sve poruke poslate na broker. Potom se sačuva rezultat.



Dijagram 4.2 Algoritam glavne niti testa



Dijagram 4.3 Tok koda pomoćnih niti

U prethodnom algoritmu je naznačeno kako glavna nit testa radi. Na dijagramu 4.3 je predstavljen tok koda na pomoćnim nitima koje čitaju podatke iz reda i šalju ih ka brokeru.

Nit iterira kroz petlju dok god traje test. Svaka iteracija proverava da li se nešto nalazi u *queue*-u. Kada se desi da je *queue* prazan, nit će napraviti pauzu od 10 milisekundi. Ovo je stavljeno samo radi rasterećenja procesora, da se ne bi oduzimali procesorski resursi na neprestano proveravanje da li je *queue* prazan. Međutim, dokle god ima poruka u *queue*-u, algoritam neprestano radi na pražnjenju i slanju na broker.

Kombinacijom opisane 2 paralelne radnje postignuta je maksimalna efikasnost iskorišćenosti

procesora po zadatim parametrima. Prema procenama, svaki drugačiji pristup bi rezultirao nepotrebnim čekanjem ili neproduktivnim korišćenjem procesorskih resursa.

4.4 Parametri i merenja

U fazi planiranja i definisanja ovog alata ciljano je maksimalna širina mogućnosti parametrisanja, odnosno postaviti maksimalno mnogo parametara smislenih za test. Razlog tome je ideja za alatom koji će simulirati što je više moguće različitih primena u sistemu. Naime, neki realni sistemi će imati jako puno manjih poruka. Neki sistemi će imati puno poruka iz različitih delova distribuiranog sistema. Treća krajnost upotrebe se ugleda u veličini poruka, neki sistem će zahtevati razmenu povećih poruka. Cilj aplikacije je da omogući pokrivenost što je većeg broja slučaja podešavanjem različitih parametara.

Nakon postavljenih parametara potrebno je definisati šta se tačno meri. Problem je u tome što je većina podataka koje se u praksi pokazuju kao merenja protoka, u ovoj aplikaciji je postavljeno kao ulazni parametar. Na primer: količina obrađenih poruka u sekundi (izražena u bajtovima) ili broj obrađenih poruka u sekundi. Ova aplikacija takve postavke konfiguriše na ulazu, a test proverava da li će broker obraditi zadato. Dakle, jedan od izuzetno bitnih rezultata je da li će broker uopšte stići da obradi podatke za zadate parametre. Rezultat ovog merenja se svodi na nekoliko mogućnosti: broker obradi sve podatke, broker napravi propust podataka ili uopšte ne izdrži da radi pod navedenim opterećenjem (*app crash, fatal error*).

Osim količine protoka podataka, ključni parametar je brzina obrade poruke od strane brokera. Ovaj parametar je jednoznačno meriv. Naime, moguće je za svaku poruku utvrditi dužinu vremenskog intervala od trenutka slanja ka brokeru do trenutka prijema. Za ovako merenje potrebno je identifikovati poruke te je iz tog razloga na svaku poruku dodat GUID formata 8-4-4-4-12. S obzirom da je neophodno više od jednog polja, poruka je stavljena u jednu struktura sa 2 polja: *guid* i *data*, gde je *guid* opisana identifikacija i data neki nasumičan tekst dužine parametra **z**. Na listingu 4.2 je prikazan primer jedne test poruke.

```
{
      “guid”: “2767753e-eaf0-4f25-a947-dafff7ada8ba”,
      “data”:
            “qmcqixfpceybopmzobllhmkfcvmcyzpzovzfvpanosxfqrmpxnspwbgq
            vjcuohksznpyjizajfvkqdqlstvkxbentvmijuwjbfhsanszbbxnoxabxvlc
            hqbptfomdqd”
}
```

Listing 4.2 Primer test poruke dužine 128B

Merenje brzine protoka poruka je trivijalno zahvaljujući postavljenoj arhitekturi. Pošto su obe uloge u komunikaciji (*publisher* i *subscriber*) jedna te ista aplikacija, one dele memoriju. Dovoljno je u jednu

strukturu podataka u memoriji zabeležiti po *guid*-u vremenski trenutak slanja poruke. Po prijemu poruke, proverava se u istoj toj strukturi kada je poruka poslata. Vremenska razlika između momenta prijema poruke (trenutak kada poruka pristigne) i momenta zabeleženog u strukturi daje brzinu protoka poruke od *publisher* do *subscriber* komponente.

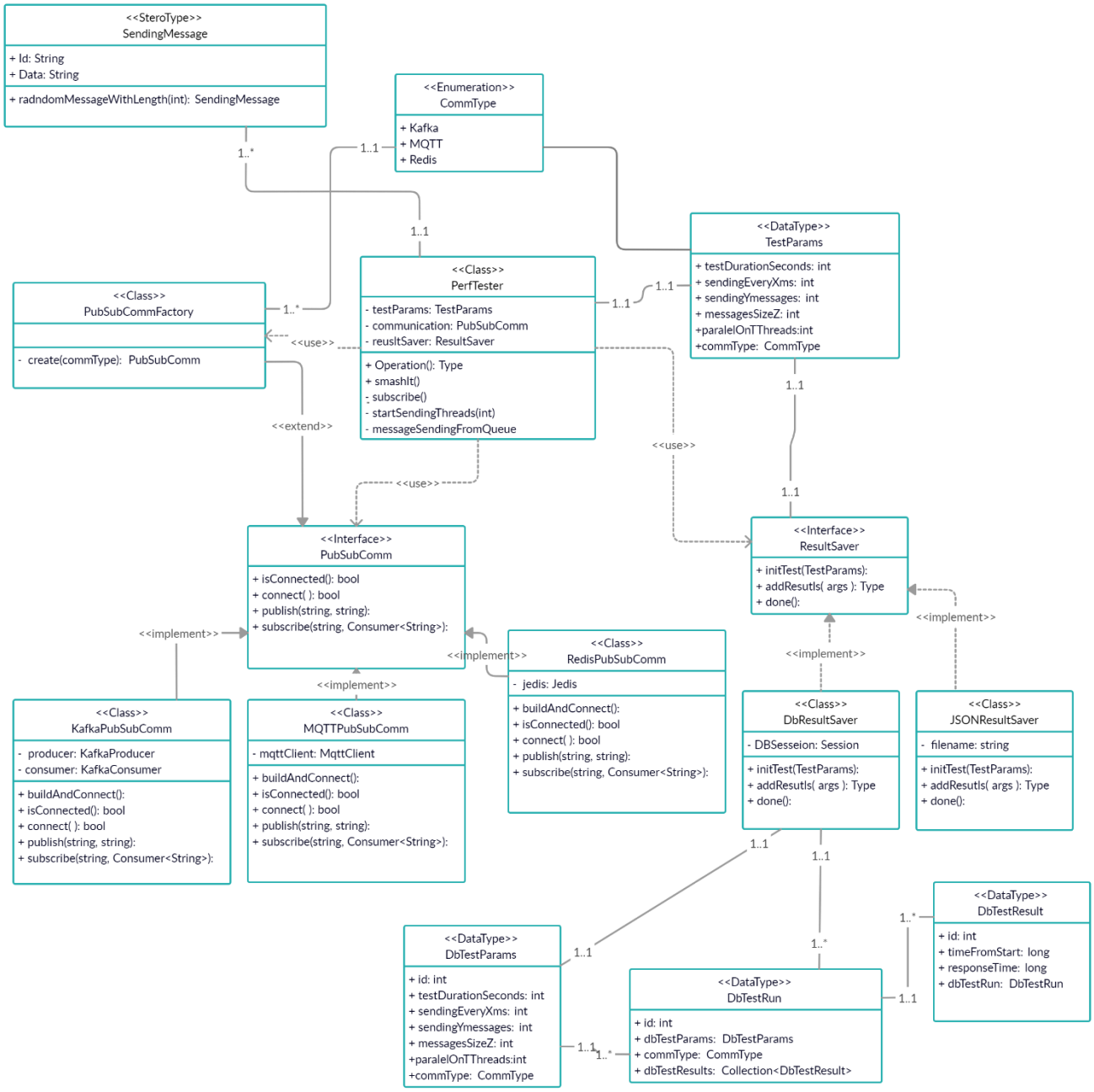
Prethodni pasus opisuje implementaciju *subscriber* uloge ovog testa. Posao *subscriber* dela se pokreće po prijemu poruke i radi sledeće

- Deserijalizuj poruku
- Proveri po *guid* kada je poruka poslata
- Oduzmi trenutno vreme od vremena slanja
- Evidentiraj izračunatu razliku na *ResultSaver*

Očigledno je da u vreme potrebno za tok poruke od *publisher* do *subscriber* komponente ulaze i neke dodatne radnje. Vreme se povećava za, na primer, deserijalizaciju (kod prijema poruke), vreme potrebno za dodavanje u *queue* i uzimanje iz *queue*-a, i slično. Ova dodatna vremena su prisutna kod sve 3 vrste brokera, pa se mogu posmatrati kao konstante (nisu fiksne, ali se mogu razlikovati minimalno) koje ne utiču značajno na merenje u krajnjem poretku.

4.5 Dijagram klasa cele aplikacije

Na dijagramu 4.4 je predstavljen dijagram klasa cele aplikacije. Gornja polovina dijagrama predstavlja, prethodno opisan viši nivo apstrakcije testiranja. On je koncentrisan na klasu koja izvršava sam test: *PerfTester*. On koristi 2 glavne pomoćne klase *SendingMessage*, reprezent poruke koja se šalje na broker, a primer serijalizovane u JSON format je dat na listingu 4.2, i *TestParams* koji predstavlja objekat grupisanja ulaznih parametara testa. Osim navedenih sadrži i zavisnosti po interfejsima *PubSubComm* i *ResultSaver-a*, na koje se oslanja prilikom samog vršenja testa.



Dijagram 4.4 Dijagram klasa test aplikacije za testiranje brokera

Na levoj strani dijagrama je uočljiv dizajn patern *Factory method* gde se jedna metoda koristi kreiranje instance. U konkretnom slučaju, ovaj dizajn patern je iskorišćen za kreiranje komunikacije po *PubSubComm* interfejsu u zavisnosti od odabranog tipa brokera, *enum* parametar tipa *CommType*. U nižem centralnom delu se nalaze implementacije interfejsa *PubSubComm,* za svaki tip komunikacije jedna implementacija, kao i *ResultSaver* implementacije čuvanja rezultata u JSON fajl i u relacionu bazu podataka.

Skroz dole (desna polovina), 3 klase predstavljaju entitete koje *DbResultSaver* mapira na tabele u bazi podataka, te ih pod istima čuva. Isečak sa klas dijagram posvećen analizi konkretno tog dela se nalazi na dijagramu 4.1 u okviru ovog poglavlja.

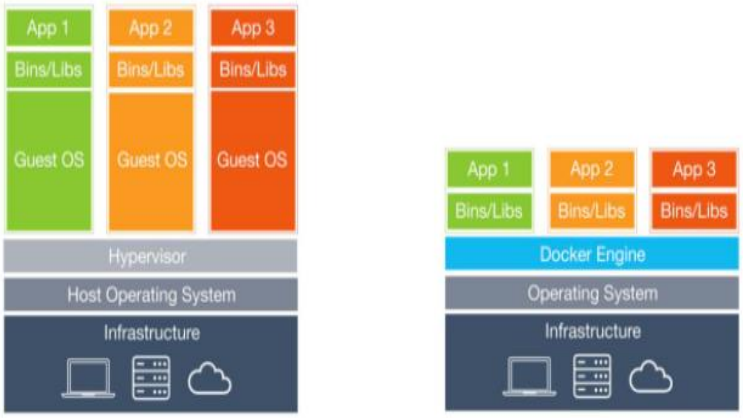
4.6 Pomoćni alat

Osim brokera i aplikacije za testiranje korišćeni su još neki pomoćni alati raznih svrha u cilju olakšavanja testiranja. Koristeći pomoćne alate, u smislu gotovih rešenja, prepušta se više vremena za fokusiranje na samu srž problematike zadatka dok se za sporedne delove koriste već gotovi alati.

4.6.1 Docker

Docker je softverska platforma za građenje aplikacija baziran na takozvanim kontejnerima. Kontejner je mala i minimalizovana slika operativnog sistema sa zasebnim okruženjem no ipak izolovanim od ostatka sistema.

Ideja kontejnerizacije je izgradnja softverske jedinice sa svojim okruženjem a maksimalno nezavisnim od ostatka sistema. U okviru okruženja, odnosno unutar jednog kontejnera, se nalazi aplikacija kao i sve potrebne zavisnosti. Na taj način se pokretanje aplikacije ograđuje od problema različitih okruženja i različitog hardvera. Preteča kontejnera je bila upotreba *Virtualnih mašina*. Na taj način se uspeo zadržati izolovan jedan deo sistema od drugog. Međutim, *virutalne mašine* su memorijski teške, svaka sadrži potpun operativni sistem te pokretanje zauzima mnogo nepotrebnih resursa. *Docker* takođe radi na principu *virtualne mašine* samo je operativni sistem iste sveden na minimalan da zadovoljava potrebe. Time se rešio problem teškoće mašine. Na slici 4.3 je ilustrovana razlika arhitekture između *virualne mašine* i *docker*-a.



Slika 4.3 Virtualne mašine i dokeri

Kontejneri se pokreću definisanjem i izgradnjom. Jedan *docker* kontejner stvara se pokretanjem *docker slike,* odnosno *docker image*. Gradnja *docker* slike se definiše unutar *dockerfile* fajla, što prestavlja tekstualni zapis u skladu sa propisanom sintaksom. *Docker* slike se nadograđuju. Definicija unutar *dockerfile* fajla uvek počinje navođenjem neke već postojeće *docker* slike. Na postojeću sliku se doda sloj neophodan konkretnom zahtevu i dobija se nova *docker* slika. Tako izgrađena *docke slika* po pokretanju predstavlja *docker* kontejner.

U ovom radu docker slike nisu građene. Iskorišćena je glavna prednost *docker* sistema, a to je jednostavno pokretanje gotovih alata. Na primer, svaki od navedenih brokera može da se instalira na radnu mašinu, konfiguriše i tako koristi. Ali, većina alata novijeg datuma, sadrži i mogućnost korišćenja samo pokretanjem već definisane *docker slike*. Pa tako, sva 3 brokera koja se testiraju ovim radom imaju već gotove i pripremljene *docker* slike. To znači, da je dovoljno pokrenuti jednu komandu i broker je dostupan i spreman za upotrebu, bez instalacije, bez konfigurisanja bilo kakvog. Na primer, za pokretanje *mosquitto* servera kao MQTT servera bilo je potrebno izvršiti komandu navedenu u listingu 4.3.

```
docker run -it
  -p 1883:1883
  -p 9001:9001
  -v %cd%\ dockers\mosquittoConfig:/mosquitto/config
  eclipse-mosquitto
```

Listing 4.3 Pokretanje docker slike mosquitto servera

Ključno iz komande sa listinga 4.3. su **docker run** što označava da se pokreće *docker* kontejner kao i **eclipse-mosqutto** u smislu identifikacije *docker* slike koja se pokreće. Po pokretanju komande *docker* sistem proveri da li ima već skinutu navedenu sliku, po prvom pokretanju nema i potrebno je da se skine. *Docker hub* predstavlja internet biblioteku *docker* slika. *Docker* sistem sa *docker hub*-a preuzme traženu sliku te je pokreće.

4.6.2 MySql

MySql je dobro poznat servis relacione baze podataka. U ovoj aplikaciji se koristi za skladištenje rezultata testova. Pogodan je za skladištenje podataka jer nudi naprednu pretragu koristeći SQL sintaksu.

Ovaj rad koristi MySql zbog mogućnosti skladištenja podataka, naprednog pretraživanja kao i mogućnost integracije sa Grafanom.

4.6.3 Grafana

Grafana predstavlja *web* servis za vizualizaciju podataka. Sadrži napredne mogućnosti konfigurisanja i crtanja grafika baziranih na podacima.

Sadrži integraciju sa mnogim tipovima skladištenja podataka. Između ostalog, sadrži integraciju sa MySql serverom. To znači, da Grafana ume da prikaže grafički podatke koji se nalaze u MySql bazama podataka. Pa tako, rezultati sačuvani tokom izvršavanja testova mogu biti prikazani ovim alatom.

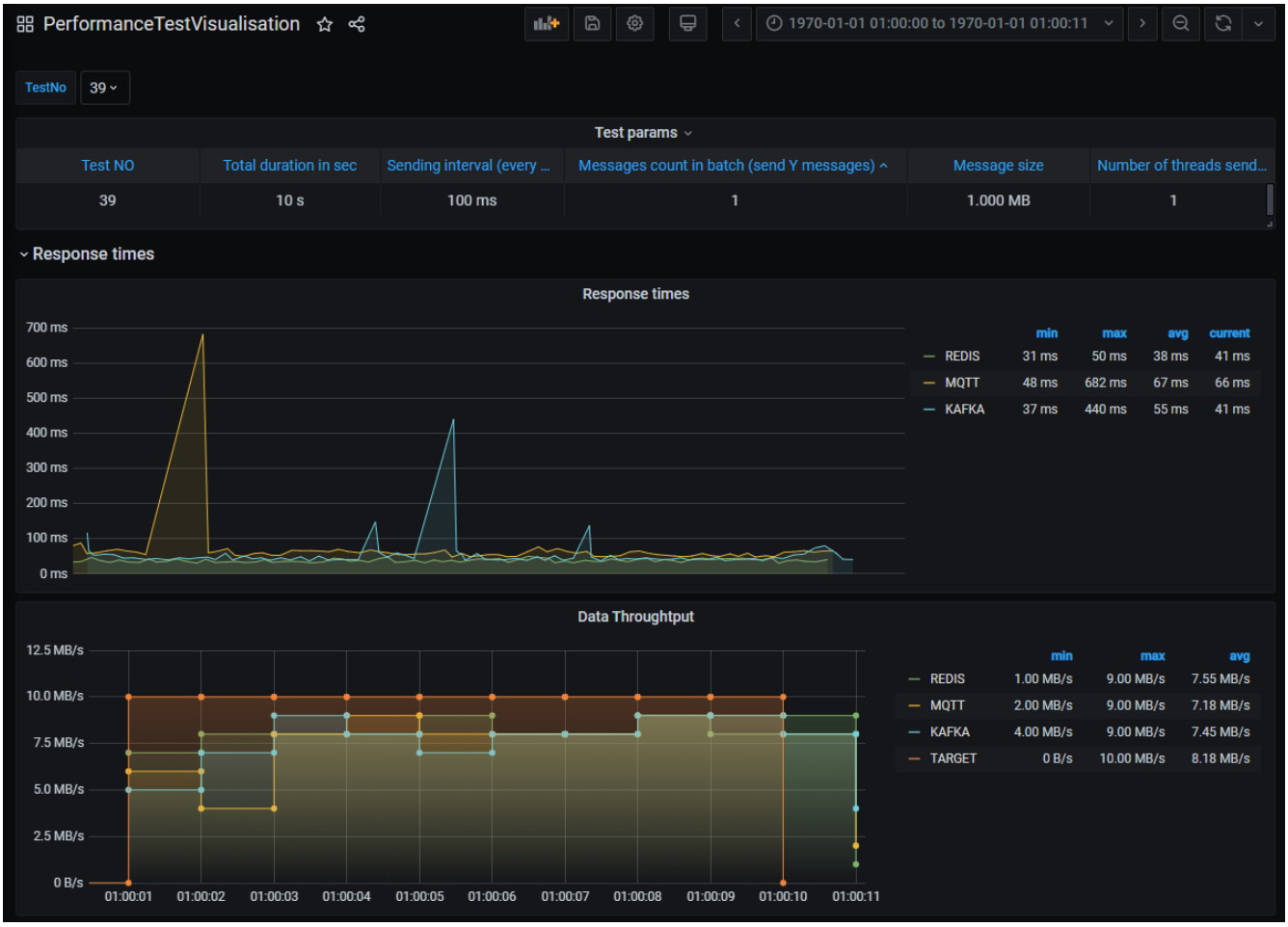
Grafana najčešće prikazuje podatke u formi vremenskih serija, odnosno kretanje vrednosti kroz vreme. Podaci se iscrtavaju u koordinatnom sistemu gde je x osa najčešće vreme, a y osa predstavlja vrednost u zadatom trenutku. Pored prikaza vremenskih serija, Grafana podržava i mnoge druge načine vizualizacije uz mnogo naprednije funkcije.

Kontrolna tabla, odnosno *dashboard,* predstavlja vid stranice koja prikazuje skup grafikona. Na kontrolnoj tabli je moguće nacrtati i konfigurisati proizvoljno grafikona različitih vrsta. Cilj kontrolne table je da smisleno vizualno predstavi podatke. Na slici 4.4 je dat proizvoljan primer jedne kontrolne table.

Slika 4.4 Grafana dashboard - primer

Grafana kontrolna tabla je korišćena u ovom radu kao alat za prikaz rezultata testova. Naime, aplikacija za testiranje brokera se bavi samim merenjem ponašanja broker i kao rezultata daje rezultate testova. Rečeno je da se ti rezultati, između ostalog, čuvaju u MySql bazi podataka. Grafana se podešava da gađa bazu podataka gde su rezultati testova, te se iscrtan jedna kontrolna tabla namenjena za vizualizaciju eksperimentalnog dela ovog rada.

Za potrebe ovog rada napravljen je Grafana kontrolna tabla koji prikazuje parametre i rezultate merenja testa. Na slici 4.5 je prikazan primer rezultata jednog testa. Na kontrolnoj tabli je data mogućnost odabira testa. Vidljiv je broj u gornjem levom ćošku nazvan *TestNo* gde su ponuđeni svi izvršeni testovi po identifikaciji. Iznad grafikona prikazana je tabla parametara nazvana *Test params,* gde su navedene vrednosti parametara odabranog testa. Zatim idu 2 dijagrama. Prvi dijagram je grafikon brzine protoka, nazvan *Response times*, prikazuje kretanje brzine protoka poruka kroz vreme, gde je brzina protoka poruke prethodno objašnjena razlika između trenutka slanja i prijema poruke. Drugi dijagram je grafikon količine protoka, nazvan *Data Throughput,* predstavlja usrednjen pokazatelj protoka podatak u sekundi. U okviru svakog od grafika s desne strane je prikazana legenda sa tabelom celokupne analize podataka, odnosno prikazom minimuma, maksimuma i proseka.



Slika 4.5 Primer vizualizacije podataka ovog rada

Nestandardno za korišćenje Grafane je x osa dijagrama ove kontrolne table. Najčešće se Grafana koristi za prikaz vrednosti u realnom vremenu, te x osa bude stvarno vreme događaja i vrednosti. U ovom primer, kao vreme je uzeta vrednost vremenske udaljenosti trenutka razmene poruke od početka testa u milisekundama. Zbog toga se na kontrolnoj tabli prikazuju podaci za 01.01.1970, po većini standarda prikaza vremena u softveru.

Podatak o vremenu razmene poruke, u vidu udaljenosti od početka testa, je namenski postavljen podataka za lepše i lakšu vizualizaciju. Ovaj podatak nije neophodan. Dodat je radi lakše i lepše integracije sa Grafanom.

Poglavlje 5

Rezultati testiranja

Srž praktičnog dela ovog rada je iskazivanje rezultata testiranja. Oni su opisani u ovom poglavlju. Predstavljen je proces izvršavanja testa kao i najinteresantniji dobijeni rezultati.

5.1 Podešavanje okruženja

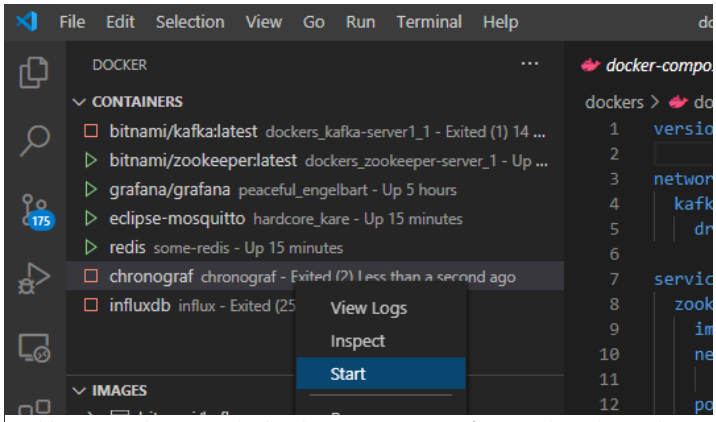
Prvi korak u pokretanju testa jeste podešavanje okruženja. Naime, da bi se izvršio test neophodno je obezbediti sva 3 brokera. Zahvaljujući upotrebi *docker* sistema, ovaj proces je izuzetno jednostavan i bezbolan. Pokretanje je moguće kroz komandnu liniju i to: izlistati sve *docker* kontejnere komandom `docker ps -a`, zatim pokretanjem pomoću komande `docker start idcontainer`. Na listing 5.1 je dat primer izvršavanja komandi i njihovog ispisa.

```
C:\Users\rober>docker ps -a
CONTAINER ID  IMAGE                     COMMAND                  STATUS                      NAMES
0e72de1f2b0b  bitnami/kafka:latest      "/opt/bitnami/script…"   Exited (1) 2 minutes ago    kafka-server1_1
9398908dc55e  bitnami/zookeeper:latest  "/opt/bitnami/script…"   Up 2 minutes                zookeeper_1
ab25b3b38d08  grafana/grafana           "/run.sh"                Up 5 hours                  peaceful_engelbar
881d58d37bb8  eclipse-mosquitto         "/docker-entrypoint.…"   Up 3 minutes                hardcore_kare
bdf1a4b9cf1c  redis                     "docker-entrypoint.s…"   Up 3 minutes                some-redis
218d70d46eae  chronograf                "/entrypoint.sh chro…"   Exited (255) 7 weeks ago    chronograf
3318ec0c883a  influxdb                  "/entrypoint.sh -con…"   Exited (255) 7 weeks ago    influx

C:\Users\rober>docker start 218d70d46eae
218d70d46eae
```

Listing 5.1 Izlistavanje doker kontejnera i pokrene

Intuitivnija upotreba *docker* sistema koristeći razvojno okruženje je moguće u mnogo alata. Na slici 5.1 je prikazan primer pregleda i pokretanja *docker* kontejnera koristeći Visual Studio Code. Upotrebom ovog okruženja moguće je upravljati *docker* kontejnerima i slikama pomoću grafičkog korisničkog interfejsa.



Slika 5.1 Pokretanje docker kontejnera pomoću Visual Studio Code-a

Kada su sva 3 brokera podignuta, može se pristupiti izvršavanju samog testa. Nakon preuzimanja koda aplikacije i izgradnje fajla sa ekstenzijom *.*jar*, aplikacija se pokreće kroz komandnu liniju i to po sledećem upustvu:

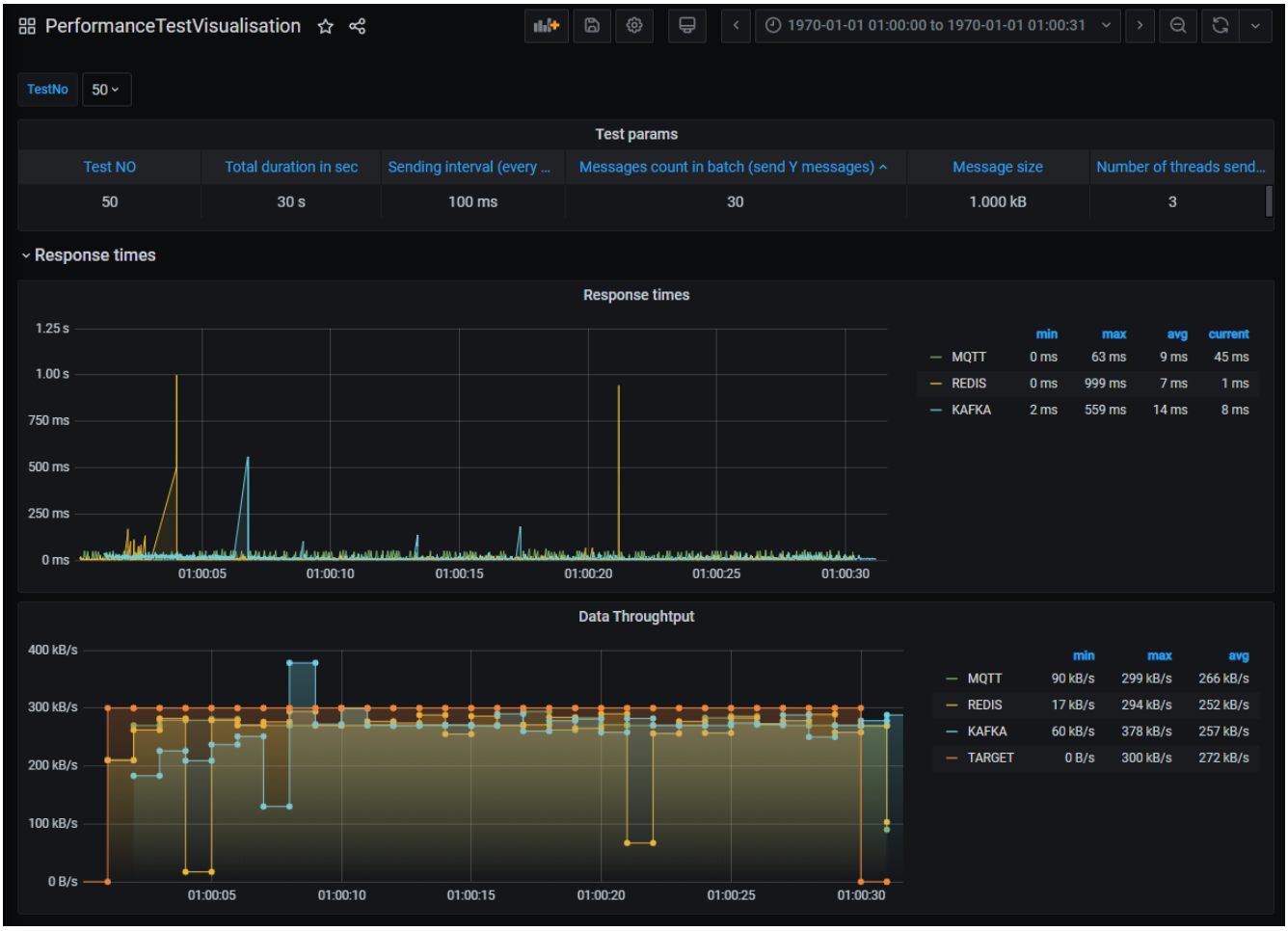
commands on run are: 1 2 3 4 5
Where:
1: S - Test duration in seconds
2: X - Cycle interval of sending messages
3: Y - Number of messages to send
4: Z - Message size
5: T - Number of threads sending from

Ovo upustvo će i sama aplikacija prikazati ukoliko se pokrene za argumetnom *-h*. Ukoliko se ne proslede navedeni argumenti, test će biti izvršen pomoću podrazumevanih parametara testiranja koji će se po pokretanju ištampati na konzoli. Na listingu 5.2 je prikazan primer pokretanja testa za sledeće parametre: *30 sekundi se šalje na svakih 100ms po 30 poruka dužine 1000B paralelno na 3 niti.*

```
java -jar MessageOrientedCommunicationTest-1.0-jar-with-dependencies.jar 30 100 30 1000 3
2020-09-24 22:15:05,745 main ERROR appender File has no parameter that matches element
Policies
22:15:05.968 [main] DEBUG it.robii.messageorientedcommunication.config.ConfigManager - All
config initializes successfully!
22:15:05.991 [main] DEBUG it.robii.messageorientedcommunication.Main - Hello, worlda!
Not parsed paralelOnThreads from index 4. Set default 3
22:15:05.992 [main] DEBUG it.robii.messageorientedcommunication.Main - testDuration=30
22:15:05.992 [main] DEBUG it.robii.messageorientedcommunication.Main - everyXms=100
22:15:05.992 [main] DEBUG it.robii.messageorientedcommunication.Main - sendYmessages=30
22:15:05.993 [main] DEBUG it.robii.messageorientedcommunication.Main - msgSize=1000
22:15:05.993 [main] DEBUG it.robii.messageorientedcommunication.Main - paralelOnThreads=3
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
22:15:06.131 [main] DEBUG it.robii.messageorientedcommunication.Main - let's subscribe!
22:15:06.133 [main] DEBUG it.robii.messageorientedcommunication.Main - Subscribed!
22:15:06.133 [main] DEBUG it.robii.messageorientedcommunication.Main - Let's publish...
22:15:06.355 [main] DEBUG it.robii.messageorientedcommunication.Main - Published...
22:15:06.355 [main] DEBUG it.robii.messageorientedcommunication.Main - Waiting response...
22:15:06.399 [pool-2-thread-1] DEBUG it.robii.messageorientedcommunication.Main - Hey,
man, i got a message:EvetiKur be: 2020-09-24T20:15:05.991105700Z
22:15:06.400 [main] DEBUG it.robii.messageorientedcommunication.Main - Got it. all good!
```

Listing 5.2 Primer pokretanja testa

Nakon pokretanja testa ispis na konzoli će se veoma brzo smenjivati i nije moguće ispratiti ispise. Znak da je test završen je ispis poslednje linije pred zatvaranje aplikacije u smislu obaveštenja o tome. Po izvršenom testu, na Grafani, u polju *TestNo* je ponuđen novi test. Po odabiru, na kontrolnoj tabli se prikazuju rezltati testa, kao što je prikazano na slici 5.2.

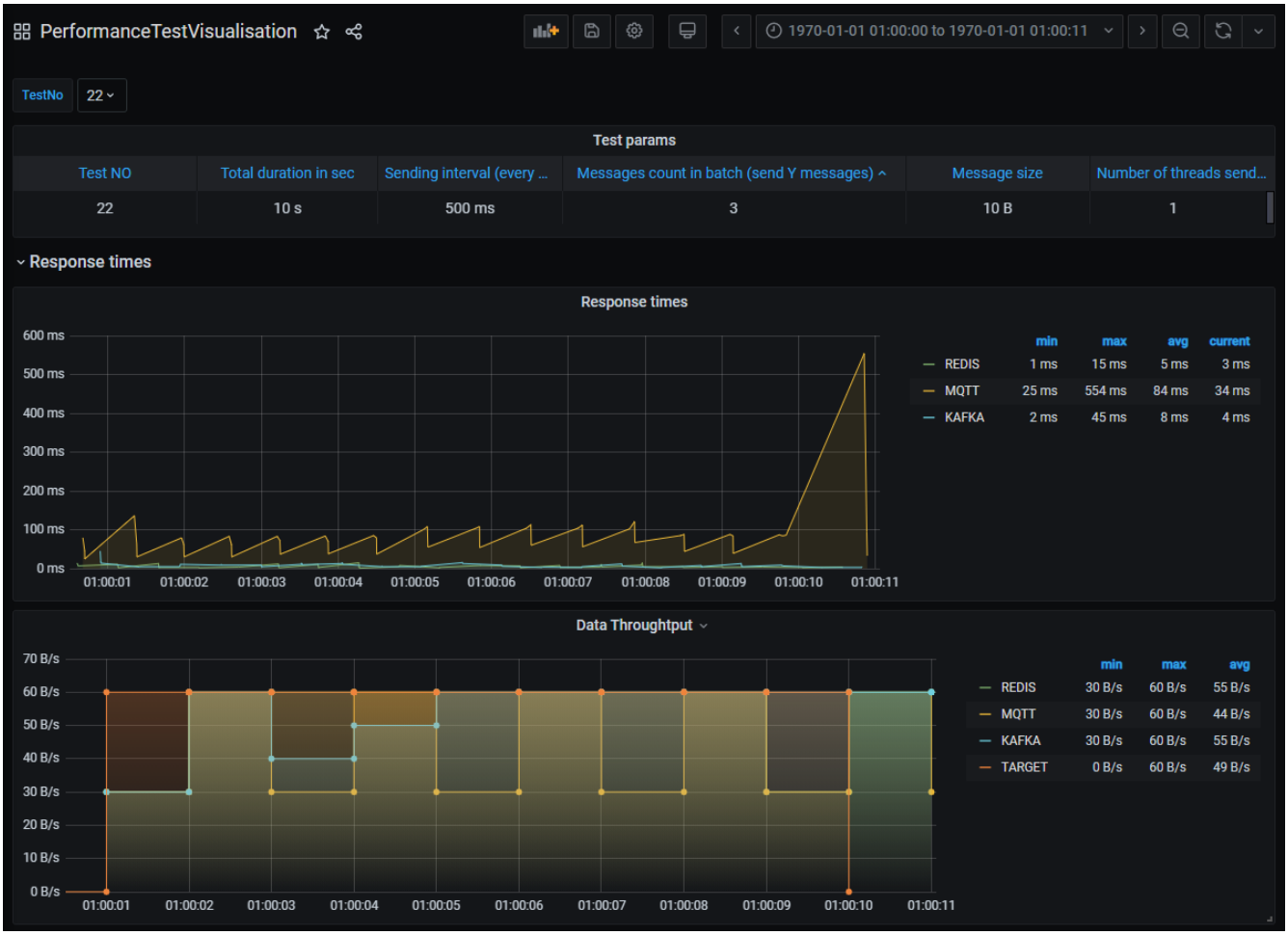


Slika 5.2 prikaz rezultata reprezentacionog testa

U cilju maksimalne ravnopravnosti između brokera poželjno je obezbediti jednake uslove iskorišćenosti resursa hardvera tokom trajanja celog testa. Iz tog razloga je preporuka da se računar na kome se vrši testiranje ne upotrebljava za vreme vršenja testa. Takođe, minimalno prisustvo aktivnih nepotrebnih procesa se preporučuje u cilju smanjenja mogućnosti uticaja na rezultate testa.

5.2 Osnovni rezultati

Prvi izvršeni rezultati nisu potpuno merodavni. Izvršavanjem testova su ustanovljene neke pravilnosti. Prvi korektan test je prikazan na slici 5.3. i evidentiran je kao *TestNo* 22. U pitanju je vrlo lagan test u *trajanju od 10 sekundi i slanju svega 60B po sekundi u vidu 500ms pauze po 3 poruke veličine 10B*.



Slika 5.3 Prvi korektan test. Test broj 22

Na gornjem grafikonu su interesantne 2 stvari:

- žuta linija koja predstavlja brzinu odgovora MQTT-a je značajno višlje pozicionirana tokom celog testa u odnosu na zelenu i plavu što znači da je MQTT tokom celog testa sporije sprovodio poruke. Osim grafikona i table pored legende pokazuje isto. Prosečno vreme protoka MQTTa je 84ms, dok su Kafka i Redis na 8ms i 5ms
- žuta linija pri kraju testa ima vrh koji doseže čak dok 500ms. Takođe i tabelarni prikaz dokazuje isto, naime maksimalno vreme odgovora je 554ms

Ovaj test spada u osnovna početna testiranja te ne treba pre vremena donositi zaključke.

Test broj 24 prikazan je na slici 5.4 i prvi je pokušaj povećanja protoka. *Naime u trajanju od 10s se šalje po* 10 *poruka na razmaku od 200ms veličine 10B paralelno sa 3 niti.* Ovaj test ima sličnu problematiku kao prethodni: MQTT značajno kasni. Na dijagramu količine protoka podataka je jasno vidljivo koliko je više vremena bilo potrebno da se završi test za MQTT. Naime, poruke se kreiraju tačno onoliko sekundi koliko je konfigurisano, ali niti za samo slanje rade dokle god se ne pošalju sve poruke, a *subscriber* je implementiran tako da sačeka još određeni vremenski period i obrađuje poruke.

Slika 5.4 Prvi pokušaj povećanja protoka. Test 24

Na grafiku količine protoka testa 24 (slika 5.4) jasno dolazi do izražaja crvena linija imenovana sa *TARGET*. Ova linije se docrtava i označava aproksimaciju količine poslatih podataka. Podaci ove linije u idealizovani, matematičkom formulom su izračunati sledećom formulom:

```
TARGET = sendingYMessages * ofZsize * (1000/everyXms)
```

Ova formula predstavlja teorijski reprezent količine poslatih poruka u jednoj sekundi izraženo u bajtovima. Množenjem broja poruka za slanje i veličine dobija se količina poslatih podataka u jednom slanju. U jednoj sekundi je moguće slati više od jednog slanja, ako je parametar **x** kao interval slanja, odnosno *everyXms,* manji od 1000. Zbog toga je potrebno pomnožiti i sa brojem slanja unutar jedne sekunde, a to je 1000/*everyXms*.

Ova vrednost je samo aproksimacija, odnosno idealizovana je i praktično nije realno da se ovoliko poruka pošalje u vremenskom intervalu trajanja testa. Razlog je što računaru treba neko vreme i da izvrši posao slanja. Iako su to milisekunde ili čak nanosekunde, sabiranjem rezultata je moguće dokazati da se izračunat rezultat može postići samo kod veoma nisko-zahtevno parametrizovanih testova. Ipak, ovaj podatak se iscrtava radi mogućnosti poređenja vršenja testa u odnosu na idealno.

Grafikon protoka testa 24 (slika 5.4) jednoznačno izražava problem kašnjenja poruka. Nakon spuštanja *Target* linije na 0 *MQTT* ostaje veći od 0 još 6 sekundi. To znači, da je MQTT-u bilo potrebno još 6 sekundi nakon poslate poslednje poruke da ih obradi sve. Ovakvi rezultati ukazuju na problematiku.

U testu 27 se prvi put pojavljuje greška sa nemogućnošću obrade poruka. Kao što su i prethodno 2 opisana testa ukazivala, MQTT nailazi na problem. *Mosquitto* ima konfiguracioni parametar koji ukazuje na maksimalan broj poruka u memoriji u jednom trenutku. Nakon povećanja pomenutog parametra test je prošao sa svim podacima, ali je MQTT i dalje bio značajno sporiji.

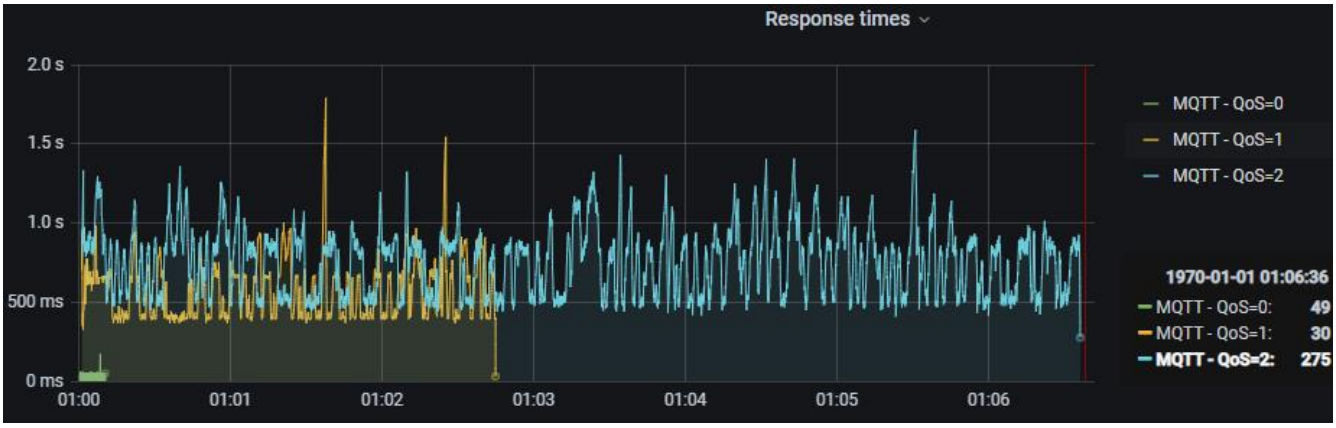
MQTT server ima jednu karakteristiku koja utiče na brzinu. U pitanju je kvalitet usluge, odnosno *Quality of Service*, iliti QoS. Ova karakteristika je detaljno opisana u poglavlju 3, sekcija 3.1.1. U tom poglavlju je navedeno da je QoS parametar kojim se direktno balansira prioritet između brzine i sigurnosti dostave poruke. Pošto je postavljeno okruženje za testiranje, izvršen je test koji je testirao samo MQTT broker ali sa različitim QoS parametrima.

Test broj 32 (slika 5.5) prikazuje rezultat ponašanja MQTT brokera sa različitim QoS vrednostima. Test cilja protok podataka od 5KB/s u *trajanju od 10 sekundi slanjem na svakih 50ms po 25 poruka veličine 10B paralelno sa 5 niti.*

Slika 5.5 MQTT broker s različitim QoS. Test broj 32

Na slici 5.5 na grafikonu brzine protoka podataka prikazan vidi se da je jedino QoS 0 usko blizu cilja. QoS 1 i 2 su mnogo niže i dugo nakon isteka trajanja testa od 10 sekundi su se još uvek dobijali podaci od brokera

Na slici 5.6 je rezultat istog testa odzumiran tako da se vidi kraj obrade podataka. Svi podaci su stigli i obrađeni, nema gubitka. Međutim, poruke iz *queue-a* dodate u okviru 10 sekundi su se obrađivali narednih 2 minute i 30 sekundi po QoS 1 odnosno 6 minuta i 15 sekundi po QoS 2.

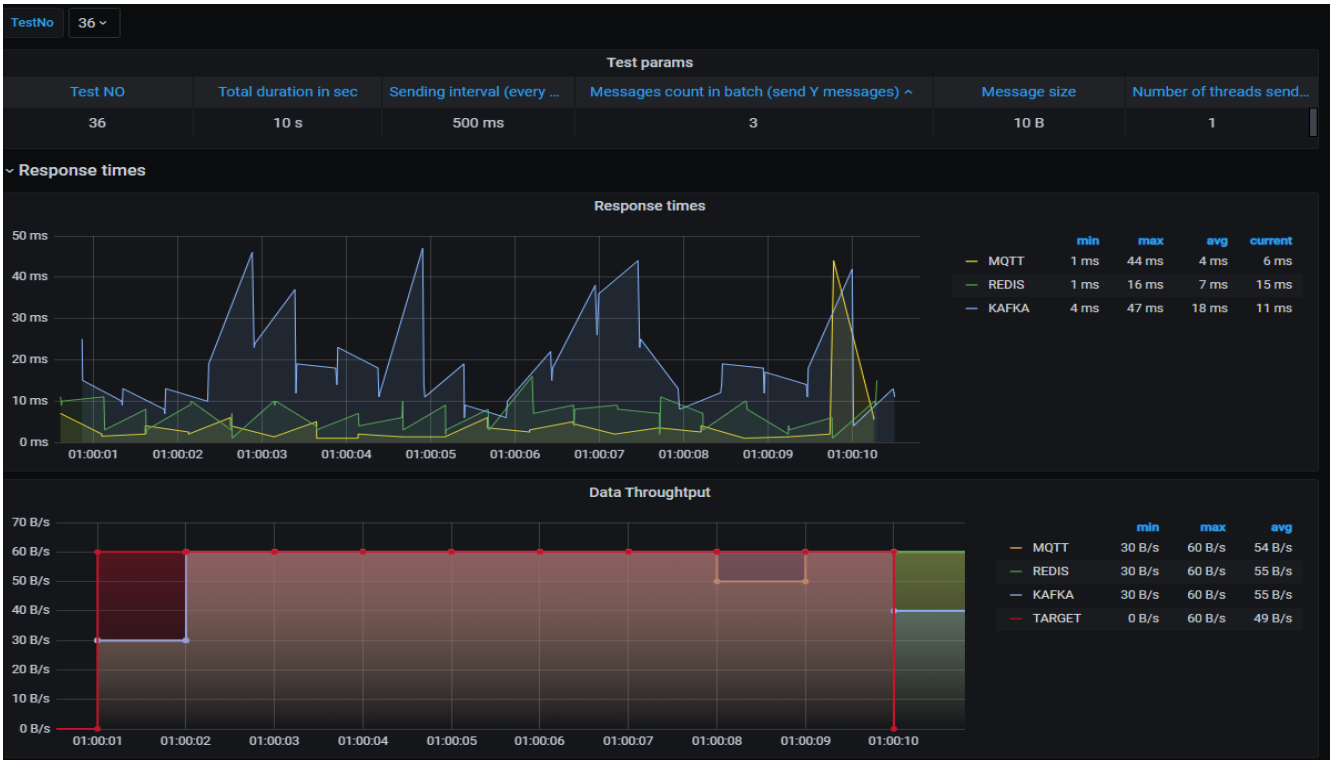


Slika 5.6 Ukupno vreme za obradu poruka po QoS

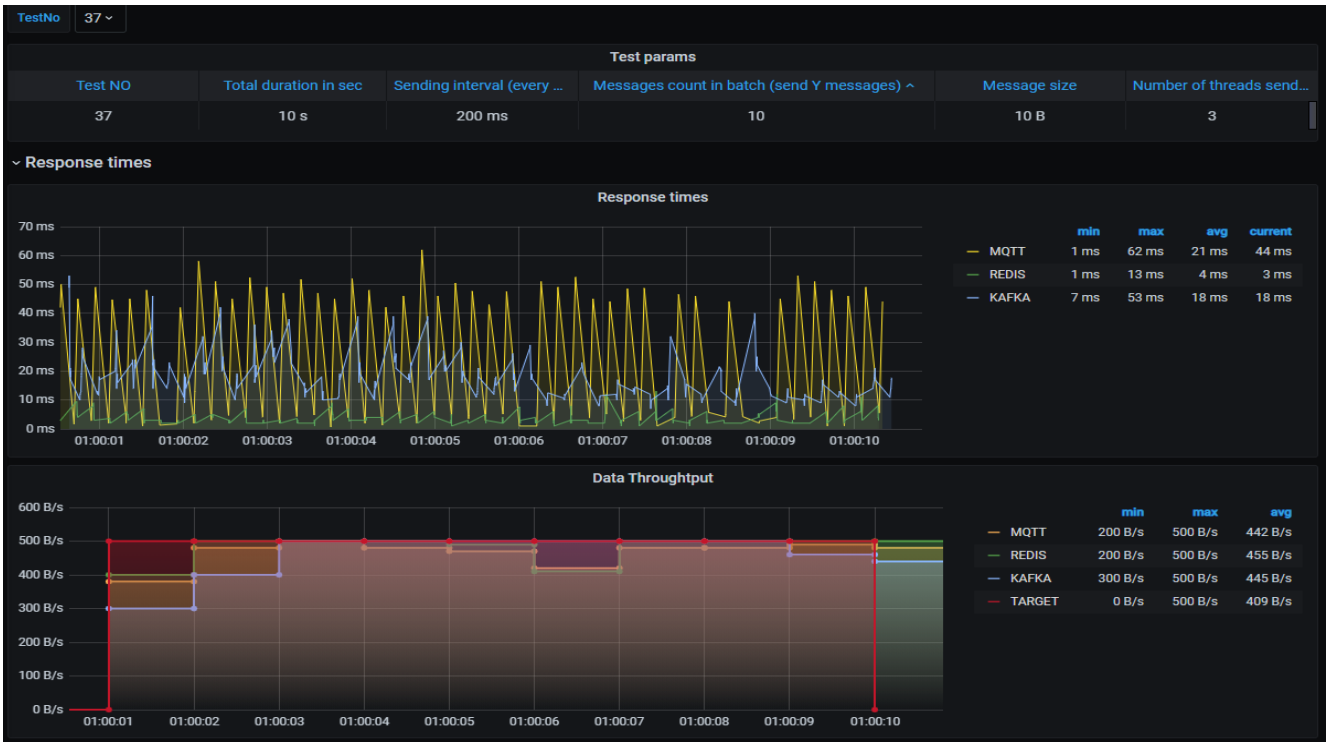
Ovakav test je poželjno detaljnije obraditi i analizirati za MQTT, ali cilj ovog rada nije poređenje MQTT servera sa različitim QoS vrednostima već poređenje brokera. Tako, s obzirom na naveden problem i dokaz istog, kako bi MQTT imao smislenu ulogu u ovom testiranju prebačeno je da QoS bude 0. Jedino QoS 0 može da se poredi po brzini sa Redisom i Kafkom.

Na slici 5.7 je prikazan test 36 koji je ekvivalent testa 22 uz izmenu QoS-a sa podrazumevanih 1 na 0. Prosečno vreme protoka poruke preko MQTT-a je smanjeno sa 84ms na 6ms što je u konkurenciji sa druga dva brokera.

Slika 5.8 je test 37 kao ekvivalent testa 24 nakon izmene QoS-a. Takođe je zabeležen drastičan pad prosečnog protoka poruke preko MQTT-a i to sa 204ms na 21ms. Iako vizualno podsećaju skokovi koji su prisutni kod oba testa oni se značajno razlikuju. Na testu 37 skokovi variraju između 1ms i 62ms dok na testu 24 isto variranje je u opsegu 53ms i 650ms.

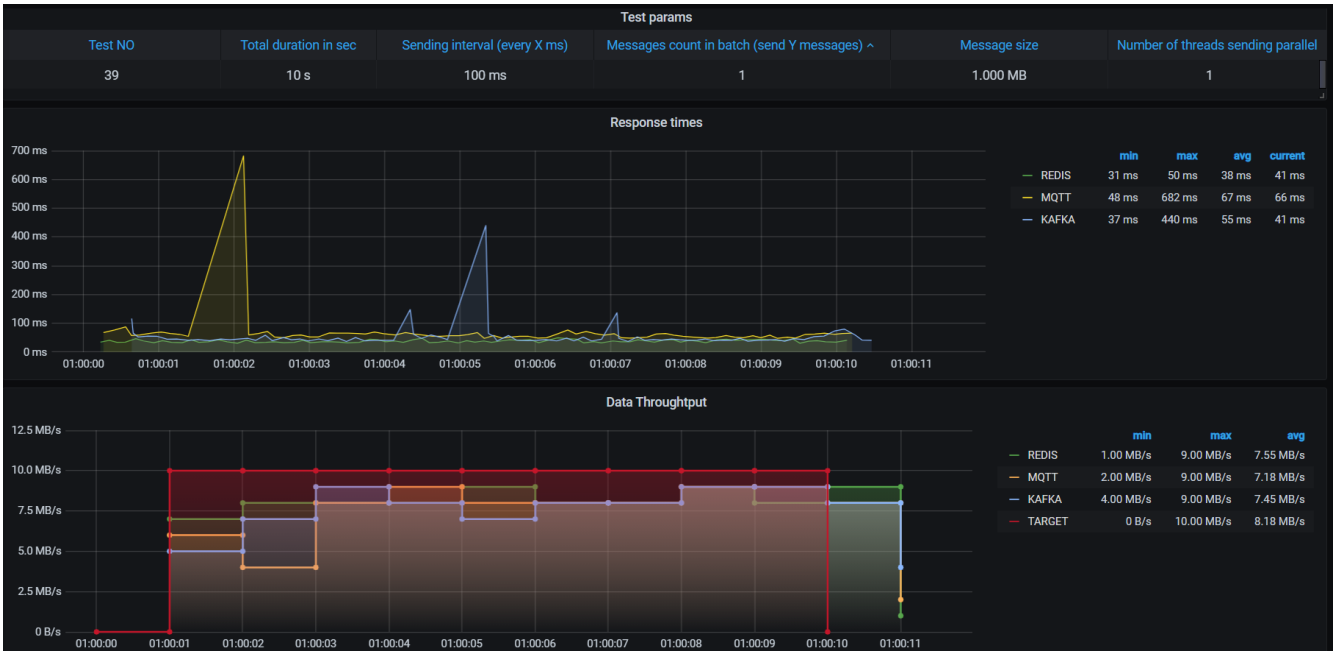


Slika 5.7 Test 36 ekvivalent testa 22 nakon izemen QoS-a.



Slika 5.8 Test 37 ekvivalent testa 24 nakon ismene QoS-a

Prethodna 2 testa su pokazatelj spremnosti okruženja i aplikacije za visoko zahtevno parametrisanje okruženja. Tako test 39 prikazan na slici 5.9 predstavlja prvi test koji je može da se kategorizuje kao visoko zahtevan jer je u pitanju prvi test gde se protok prebacuje na nivo megabajta. *U trajanju od 10s se šalje po jedna poruka veličine 1MB na svakih 100ms sa jedne niti*.



Slika 5.9 Prvi visoko zahtevan test. Test broj 39

Target linija na grafikonu količine protoka jasno ukazuje da se cilja protok od 10MB/s. Sudeći po grafici ovakvo parametrisanje testova ipak nije predstavilo nikakav problem nijednom od brokera. Sva 3 brokera su poruke prosleđivali u proseku za oko 50ms sa variranjem od 40%. Prisutna su 2 slučaja većeg kašnjenja poruke. To su 2 vrha sa grafikona brzine protoka i to žuti i plavi. Žuti je MQTT koji je dostigao kašnjenje od 682ms. Plavi vrh je Kafka čije je kašnjenje bilo 440ms. S obzirom da su prisutna samo 2 skoka kašnjenja na različitim tipovima brokera i bez učestalog ponavljanja, ovi skokovi su beznačajni.

5.3 Rezultati visokih zahteva

Ova tačka predstavlja usku srž ovog rada. Ideja pokretač cele teme je prikaz rezultata iz ove tačke. U pitanju su testovi koji pokušavaju da pogode maksimum od svakog brokera. Cilj je podizanje nivoa zahtevnosti simuliranog okruženja do najviših granica podnošenja brokera. Odnosno, ovim testovima se ciljaju momenti kada brokeri više ne mogu da podnesu tu količinu podataka i taj pritisak.

U cilju standardizacije svi testovi ove oblasti su podešeni na optimalno trajanje od 30 sekundi.

Prvi u nizu prezentovan, a samo jedan od pokušaja je test broj 67. Ciljan je protok od 500KB po sekundi upakovanih u 500 poruka i svi se šalju sa jedne *publisher* komponente, odnosno slanje se vrši sa jedne niti. Prosečne dužine protoka poruke su izuzetno malo, po 2 ms za Kafku i Redis, dok je MQTT malo

sporiji sa 4ms. Na grafiku brzine protoka se javlja jedan vrh koji značajno odstupa. U pitanju je Kafka gde se pojavila poruka čiji je protok trajao čak 440ms.

Slika 5.10 Jedan od pokušaja dostizanja maksimuma. Test broj 67

Na grafiku količine protoka se primećuje kašnjenje završetka testa kod Redis servera. 6 sekundi je duže trajao ceo test nego što bi idealno trebalo da traje. Zanimljivost na ovom grafiku predstavlja konstantan rast ukupne količine protoka poruka kod Kafka servera. Na slici 5.11 je prikazan isti grafikon na kome su ostavljene samo linija *Kafka* i *Target.* Docrtana je žuta linija kao usrednjeno kretanje u cilju prikazivanja rasta.

Slika 5.11 Ubrzanje Kafke na testu broj 67

Nakon ovog testa je pokrenut još jedan sa većinom jednakim parametrima, tačnije svim osim broja niti sa kojih se šalje. Test broj 68 slanje vrši sa 5 niti paralelno.

Rezultat testa 68, u poređenju sa rezultatima testa 67, ukazuje na to da je neophodno slanje s više niti paralelno. Naime, povećanjem broja niti, ovaj test je prošao bez kašnjenja ijednog brokera, svi su završili u 31 sekundi od pokretanja. Prosečno vreme protoka poruka je donekle veće. Iz ove 2 činjenice se zaključuje da je povećanjem broja niti prebačeno procesorsko opterećenja sa test aplikacije na broker. Povećanjem broja niti broker malo uspori, ali je test precizniji parametrima. S obzirom da ovo testiranje namerava da istraži kapacitete brokera a ne napisane test aplikacije, budući testovi moraju da

parametrišu više niti od 1 jer će povećanjem ostalih parametara samo postati značajnija razlika u tome koji deo se opterećuje i koji od njih usporava.

Slika 5.12 Ekvivalent testa broj 67 uz više paralelnih klijenata. Test broj 68

Podizanjem broja poslatih poruka na 2000 u sekundi veličine 100B se javljaju prva značajna kašnjenja. Na slici 5. 13 je prikazan rezultat ovog testa. Značajna je dužina trajanja testa.

Slika 5.13 Prva značajna kašnjenja. Test broj 70

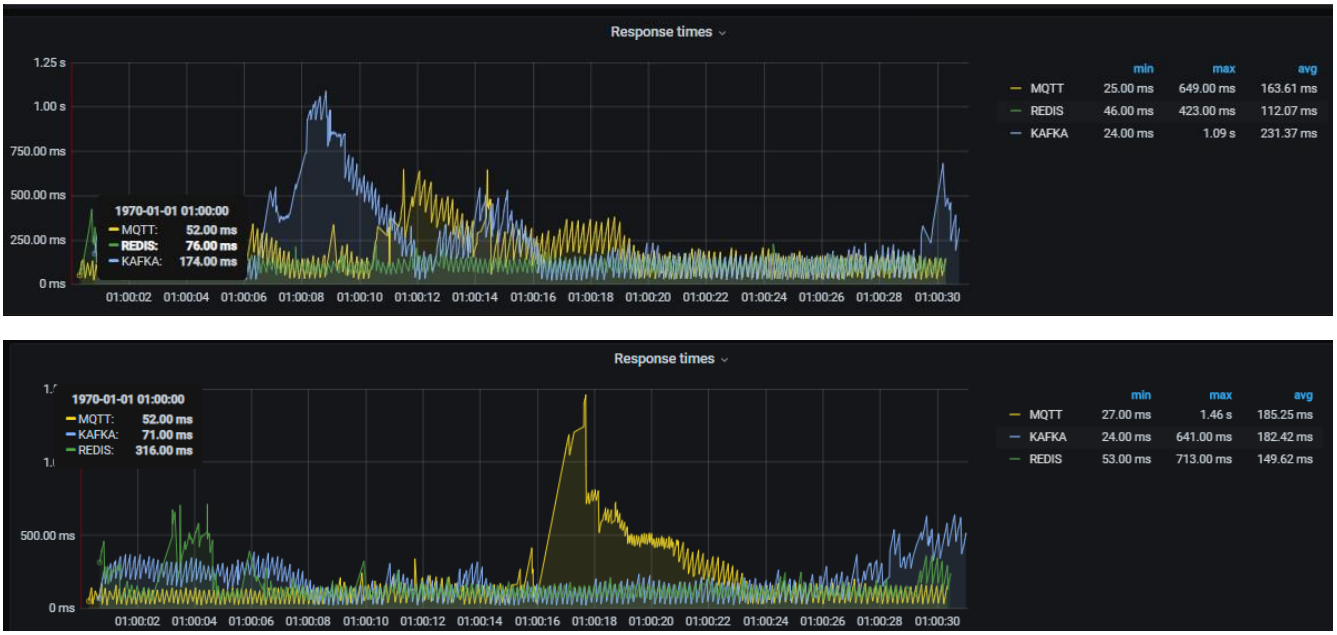
Naime, Redis nije stizao da preuzima podatke dovoljno brzo, slanje se značajno sporije dešavao pa je ceo test trajao preko 60 sekundi. Razlog tome je što po završetku trajanja testa (parametar **s** kao trajanje testa izraženo u sekundama) *queue* nije bio potpuno ispražnjen. Dakle, u ovom slučaju Redis poprilično brzo prebaci poruku od *publisher* do *subscriber* komponente, ali se javlja usko grlo na preuzimanju poruka.

Testovi broj 71, 72 i 73 dokazuju da do začepljenja dolazi prilikom preuzimanje poruka od *publisher* komponente. Naime, ova 3 testa su pokušaj pokretanja sa istim parametrima. U odnosu na test 70, ovde je smanjena veličina poruke na 10B, ali je broj poruka za slanje povećan na 10.000 poruka u sekundi i to slanjem 100 poruka na svakih 10 milisekundi. Redis nije izdržao ovu brojnost poruka. Rađena su 3 pokušaja i kod sva 3 se desilo da posle određenog vremena Redis prijavljuje problem. Ovim testovima je ispitan pravac orijentisan ka velikom broju poruka manje veličine.

Test broj 80 prikazan na slici 5.14 predstavlja ponašanje brokera pri protoku od 50MB po sekundi. Ciljani protok je parametrisan veličinom poruka od 500KB i slanjem po 10 poruka na 10 puta u sekundi. Prosečno vreme vremena protoka poruka je značajno povećano te se javljaju i značajne razlike. Redis i MQTT su povećali prosečno potrebno vreme na 90-110ms, dok je kod Kafka servera prosek postavljen čak na 300ms. Razlog povećanog proseka se može videti i na grafiku. Naime, plava linija u prvoj trećini testa ima značajno visoke vrednosti. Nakon kritične trećine, Kafka se vrati u okvir kretanja ispod 200ms, ali je prva trećina dovoljna da naruši prosek.

Slika 5.14 Velike količine podataka. Test broj 80

Ponavljanjem istog testa još 2 puta javlja se povremeno kašnjenje i MQTT-a. Na slici 5.15 su prikazani grafici brzine protoka poruka za testove 81 i 82 čije parametrisanje je ekvivalent testa 80.



Slika 5.15 Protok 50MB/s . Testovi 81 I 82

Test broj 124 je parametrisan da za simulaciju okruženja sa mnogo *publisher* komponenti i mnogo poruka. *Šalje se na svakih 50ms (*dakle 20puta u sekundi*) svaki put po 100 poruka veličine 1KB sa 50 različitih niti*. U ovakvoj simulaciji postignuto je zagušenje na sva 3 brokera. Zagušenje je najčešće prisutno u trenutku trenucima slanja poruke i to prvom polovinom testa. Na slici 5.16. je pokazan rezultat opisanog testa.

Slika 5.16 Broj paralelnih slanja 50. Test 124.

Iz table kraj legende grafika brzine protoka je uočljivo da je prosek protoka poruke na svim brokerima preko 15 sekundi. Redis serveru je prosek čak 36sekundi. Konstantna uzlazna linija na sva 3 brokera kod grafikona brzine protoka poruka je pokazatelj direktnog zagušenja od strane brokera. Naime, nove poruke stižu konstantno na obradu, a broker je zauzet trenutnima. Usred konstantnog pristizanja novih poruka javlja se konstantno povećanje broja poruka u brokeru koje treba obraditi. Nakon uspešne obrade dolazi do rasterećenja. Kod navedenih testova rasterećenju je doprineo prekid pristizanja novih poruka usred završetka testa.

Konačno, test broj 131 prikazan na slici 5.17 predstavlja najzahtevniji test ovog eksperimenta. Ciljan je protok od 100MB/s i *to slanjem po 20 poruka veličine 0,5MB na 100 milisekundi paralelno sa 12 niti.* Ovaj test je ponavljan mnogo puta u nadi pronalaska uslova pod kojima će svi brokeri dati uspešan i zadovoljavajući rezultat. Međutim, to nije uspelo. Test 131 je prikaz rezultata samo jednog od mnogih koji ciljaju 100MB/s.

Slika 5.17 Protok od 100MB/s. Test broj 131

MQTT sa QoS 0 je uspeo da obradi poruke bez gubitaka i bez značajnog kašnjenja. Naime, najbolje se pokazao u ovom testu sa prosečnim vremenom protoka od 500ms.

Kafka je konstantno povećavala kašnjenje i dostigla je čak 8s po protoku poruke. Prosečno vreme je 5s što je mnogo i van granica tolerancije.

Redis nije izdržao ni 20 sekundi prijema poruka. U početku je brzina protoka bila korektna, što se vidi na grafiku vremena protoka, zelena linija (Redis) usko prati plavu (Kafku), međutim posle 15ak sekundi Redis broker nije izdržao pritisak i odbijao je prijem novih poruka.

Poglavlje 6

Diskusija analize podataka

Prikaz rezultata implicira analizu te diskusiju istih. Ovo poglavlje posvećeno je izlaganju postignutog rezultata rada.

Pre svega, rad je po procenama autora izuzetno uspešno realizovan. Iako ovi rezultati nisu dovoljni za bilo kakav finalan zaključak, uspeh se ogleda pre svega u razvoju test aplikacije i njenom ponašanju. Njom je relativno uspešno pronađena granica podnošljivosti svakog od brokera. Intuitivno se pretpostavlja da je to zaista granica podnošljivosti tog tipa brokera. Međutim, nije. Obrazloženje se može pronaći u prvom delu ovog rada i samom naslovu rada. Tema je orijentisana na distribuiranim sistemima, dok je konkretan eksperiment rađen na laptopu jednog studenta. Iz tog razloga je ovaj rad više uspešan kao razvoj aplikacije za test nego kao dokaz ponašanja brokera.

Precizno i odgovorno ukazivanja na ograničenja brokera bi moglo da se postavi kada bi se test radio u distribuiranom sistemu. Konkretno, kada bi se postavilo minimalno 2 hardver: prvi kao klijent, odakle se šalju i primaju poruke i drugi kao server, mesto gde se nalazi sam broker i radi svoj posao. Na serveru bi usred testova različitog brokera bilo poželjno ugasiti druga 2 brokera prilikom testiranja jednog kako bi se izbeglo naticanje brokera za resurse. Teorijski, broker koji je neaktivan i nema klijenata koji razmenjuju poruke kroz njega, ne bi trebalo da zauzima resurse ili da to bude minimalno, ali to je samo teorijski tako. U praksi, odnosno realno, svaki broker, kao i skoro svaki softver radi neki posao u pozadini.

Još precizniji test bi se dobio ako bi svaki broker bio na zasebnom hardveru. Tada bi se moglo pratiti u realnom vremenu brzina i količina protoka jer bi se test mogao vršiti paralelno na sva 3 brokera. Ilustracija hardverske raspodele je prikazana na slici 6.1. Radi ravnopravnosti neophodno je da hardver bude ravnopravan na sva 3 brokera. Ovakvom distribucijom bi se moglo precizno odrediti ograničenja brokera.

Slika 6.1 Ilustracija harverske rapodele

Svakako hardver na kome je pokrenut broker mnogo utiče na njegovo ponašanje. Uticaj nije značajan ako se broker koristi prosto za razmenu poruka. Ali u primeni ovog rada gde se testiraju kapaciteti brokera, uticaj hardvera je jako velik. Shodno tome, pokazani rezultati ovim radom ne smeju se uzimati kao pokazatelj kapaciteta, već samo kao razlike u međusobnom ponašanju različitih brokera. Na primer, u eksperimentalnom delu je uspešan test bio sa 50MB/s, dok 100MB/s je označen kao neuspešan, što ne

znači da Redis ne može da obradi protok od 100MB/s, već da Redis precizno u ovim uslovima nije uspeo da ispuni taj kapacitet. Na nekom serverskom računaru ozbiljnijih performansi bi verovatno mogao bez problema i 10 puta više od toga. Ovim radom su pokazane njihove međusobne razlike u različitim simuliranim uslovima.

6.1 MQTT

MQTT se generalno dobro pokazao. Dokazan je epitet *lightgweight*. Uspeo je da bez problema obradi količinu podataka s kojom su druga 2 brokera imali značajnih poteškoća. Uslov za to je da se koristi QoS 0. QoS 0 ima najmanju sigurnost razmena poruka, i baš zbog toga se najbolje i pokazao. Naime, svaka poruka je prosto samo poslata, bez bilo kakve potvrde ili bilo kakvog obezbeđivanja. Očekivano je bilo da QoS 0 biti brži od QoS 1 i 2, ali nije bilo očekivano da će razlika biti kao na testu broj 32 (slika 5.5).

Razlika u QoS-u se izrazila samo u brzini, odnosno prikazana je samo prednost za QoS 0, mane istog se nisu ukazivale. Naime, nije dolazilo do gubitka poruka. Razlog tome je lokalna komunikacija. Na istom hardveru je pokretan test i bio smešten MQTT broker.

Postavlja se pitanje da li bi isti rezultata bio kada bi komunikacija bila preko interneta ili samo da je drugi hardver u istoj lokalnoj mreži. U tim slučajevima se pretpostavlja mogućnost gubitka pojedinih poruka, a QoS 0 ne definiše načine zaštite gubitka poruka.

Rađen je pokušaj protoka u rangu naprednijih testova i protokom od 1MB/s a MQTT-om podešenim da radi sa QoS 1 i nije izdržao pritisak već se broker pao usred serverske greške (*app crash, failure*). Naime, dovoljan je pokazatelj već naveden test broj 32 (slike 5.5) koji predstavlja značajno kašnjenje pri protoku od 5KB/s.

Iz navedenih razloga MQTT se po predstavljenim testovima pokazao kao najbolji, što nije tačno u svim slučajevima. Kao i drugi brokeri, i MQTT ima svojih prednosti i mana te je najbitnije odabrati odgovarajući alat za odgovarajuću okolnost.

6.2 Redis

Redis predstavlja dvostruko iznenađenje. Samim početkom eksperimenta, u ranim fazama razvoja i testovima nižih zahteva Redis je bio iznenađujuće najbrži. Najmanje se očekivalo od njega tolika brzina. Naime, na većini ranih izvršenih testova linija koja pokazuje brzinu Redis servera je uvek bila najniže i najmirnija u smislu konstantnosti i minimalnih oscilacija u dužini obrade poruke.

Najmanja očekivanja su se javila jer je uloga *publish-subscribe* brokera sekundaran za Redis. Redis je primarno alat za keširanje, dok je broker poruka samo nešto što nudi kao rešenje.

Izučavanjem teorije i načina funkcionisanja Redis servera pronađen je odgovor zašto je on brži od ostalih. Kako je prethodno i opisano, on koristi isključivo radnu memoriju te je jasno zašto je značajno brži. Razmatranjem ove informacija se naslutio i potencijalni problem: šta se dešava kada količina podataka bude veća od raspoložive radne memorije.

Drugo iznenađenje se moglo naslutiti. Čim se stiglo do testova koji pokušavaju da simuliraju protok izražen u megabajtima Redis serveru se javljaju problemi:

- Velik broj omanjih poruka je Redis obrađivao sa zakašnjenjem, odnosno spor proces slanja poruke
- Velike poruke u cilju velikog protoka su dovele do toga da Redis broker padne i bude privremeno nedostupan
- Veliki broj klijenata u smislu paralelnog slanja je kod Redis servera rezultiralo sa sporom obradom poruka

Najslabija tačka Redis brokera je što se često javljao problem da padne i ne bude dostupan. Tada bi u realnom sistemu komunikacija između celina u potpunosti prestala što bi moglo prouzrokovati i ozbiljnije probleme celog distribuiranog sistema.

Slično kao MQTT i Redis ima svojih prednosti i mana. Naime, predstavljena 2 iznenađenja su baš najizraženija prednost i mana. Izuzetna brzina u sistemima niskih zahteva i problemi dostupnosti servera kod sistemima visokih zahteva.

Zaključno, za Redis bi možda bilo pogodnije uraditi testiranje kao alat za keš memoriju. Tu se očekuje veće ostvarenje i uspešniji rezultat. Kao prednost alat za keš memoriju bi se navelo da radi i kao izuzetno brz broker poruka.

6.3 Kafka

Kafka je alat od kojeg je krenula inicijalna ideja ovog rada. Time je najveće očekivanje stavljeno baš na ovaj broker. Sumarno, ovaj test nije ispunio očekivanja. Verovatno jer su očekivanja bila visoka.

Interesantno zapažanje je da Kafka nikad nije izgubila poruku niti je Kafka server pao. Pod velikim pritiskom u smislu velikog obima podatak javilo se kašnjenje poruka. Neke poruke su kasnile i do 10 sekundi. Ali Kafka broker nije pao i uvek je sve poruke isporučio čime se druga 2 brokera ne mogu pohvaliti. Ovim se Kafka može označiti kao najizdržljiviji broker u ovom testu.

Međutim, Kafka čuva još jednog „džokera u rukavu“. Predstavljeni rezultati vezani u konceptu Kafka brokera su nepogodni. Naime, Kafka je namenjena da radi u okviru klastera. To znači da postignuta gornja granica ovog brokera u nekom okruženju predstavlja njegovo ograničenje kao takvog, ali uvek je moguće dodati još jedan broker u klaster, koji će deliti posao sa prethodnim, rasteretiti ga i time povećati kapacitet celog klastera.

U cilju testiranja kapaciteta brokera prethodno iskazanu informaciju je nužno uzeti u obzir. Naime, postavljanjem visoko zahtevnog okruženja sa velikom količinom prometa podataka neophodno je uzeti u obzir mogućnost ovog alata za horizontalno skaliranje. Ovaj podatka je neophodno imati na umu i u vreme sklapanja arhitekture distribuiranog sistema. Druga dva predstavljena brokera nemaju ovu mogućnost. Time Kafka daleko odskače u poređenju brokera. Na žalost, resursi praktičnog dela ovog rada nisu pokrili mogućnost testiranja i poređenja performansi na više hardverskih jedinica.

Osim mogućnosti skaliranja, bitno je podsetiti da Kafka radi i raspodelu poruka po particijama čime obezbeđuje mogućnost raspodele posla *subscriber* komponentama na nivou brokera. Ova prednost Kafka brokera nije uključena u testove. Razlog tome je nedostatak hardverskih modula, odnosno nedostatka realne distribucije. Iz istog razloga nema smisla ni podizati Kafka server u klasteru, jer više Kafka servera na istom hardveru neće raditi brže nego jedan samostalno.

Tako predlog testiranja sa početka ovo poglavlja da svaki broker koristi po jednu hardversku jedinicu treba proširiti kako bi Kafka mogla da se podigne u klasteru.

.

Poglavlje 7

Zaključak

Nezahvalno i nerealno je na osnovu prikazanog testa doneti zaključak da je neki broker bolji od drugog. Sve i da se sprovedu bilo kakvi testovi između ova 3 brokera, i dalje ne može generalni poredak da se radi jer svaki od brokera ima svoje prednosti i mane, svaki ima nešto što ga ključno razlikuje od ostalih i zbog čega je dobar sam. Brokeri, kao i svi drugi alati, koji su generalno lošiji od drugih, bivaju zaostali i zamenjeni boljima. Ova 3 brokera su konstantno aktivna u upotrebi i koriste ga mnogi sistemi.

Zaključno je neophodno još jednom napomenuti uticaj hardvera na rezultate testa. Naime, eksperimentalni deo je rađen na računaru sa 8GB RAM memorije i Intelovim procesorom od 12 logičkih i 6 fizičkih jezgara kapaciteta 2.2GHz koji radi na *Windows* operativnom sistemu. Zbog broja jezgara se pokušaj postizanja protoka od 100MB/s vršilo sa 12 niti u cilju raspodele po logičkim jezgrima procesora.

Za realne produkcione potrebe isti ovaj ekosistem za testiranje bi se mogao izvršiti uz sitne izmene. Pre svega bi se smanjio uticaj hardvera pa, kao što je navedeno u poglavlju 6, svaki broker bi trebao biti aktivan na zasebnoj hardverskoj jedinici. Zatim na tim hardverima smanjiti minimalno opterećenost od strane drugih alata. Što se ostatka ekosistema za testiranje tiče, ostalo bi slično. Baza podataka i Grafana skladištenje i analizu rezultata, a aplikacija razvijena za ovaj rad bi se mogla koristiti jednako kao i u ovom radu.

Bilo kako da se osmisli arhitektura distribuiranog sistema i koji god broker da se koristi najbitnije je imati čist i nezavisno odvojen kod. Arhitektura ove aplikacije predstavlja primer dobro organizovanog koda gde je jasno podeljen kod po nivoima apstrakcije. Ukoliko i distribuirani sistem održi nezavisno odvojene nivoe, zamena tipa brokera ne sme da predstavlja prevelike poteškoće. Zbog toga je bitno u ranim fazama razvoja sistema razmišljati o svim mogućim pravcima napretka, te ostaviti mogućnost za prelazak na naprednije alate.

Konačno, najbitnije u odabiru brokera je usaglasiti zahteve, mogućnosti i potrebe. Kao što je i dokazano u ovom radu, svaki broker ima svojih prednosti i mana. Neophodno je odabrati ono što najviše odgovara potrebama sistema, time će se dobiti najbolji rezultati. Osim toga, dobro napisan kod u dobro osmišljenoj arhitekturi neće imati poteškoća da se preorijentiše na upotrebu drugog brokera, ako dođe do potrebe za tim. Svakako, najbitnije je dobro odabrati alat za konkretno okruženje i konkretnu problematiku.

Literatura

[1] Robert Sabo i mentor dr Srđan Škrbić,
Skalabilan mikroservis orijentisan sistem namenjen berzi kriptovaluta,
Prirodno-matematički fakultet, Univerzitete u Novom Sadu, 2018.
Pristupljeno: 22.08.2020. Dostupno na:
https://github.com/robertsabo0/AWS_CryptOffer/blob/master/DiplomskiRad_RobertSabo_v2.4.pdf

[2] Geoffrey Winn i Neil G. Young,
Message oriented middleware with integrated rules engine
Sjedinjene Američke države, 07. mart 2019
Pistupljeno: 08.09.2020. Dostupno na:
https://patentimages.storage.googleapis.com/04/57/f1/e30873a39f15d8/US10592312.pdf

[3] Patrick Th, Eugster, Pascal A. Felber, Rachid Geurraoui i Anne'Marie Kermarrec
The many faces of Publish/Subscribe
Švajcarski savezni tehnološki institut, 2003
Pristupljeno: 09.09.2020. Dostupno na:
http://128.232.0.20/research/srg/netos/papers/2007_YONEKI_MSWIM.pdf

[4] Vaquar Khan,
Difference between scaling horizontally and vertically
github.com 30.01.2017,
Pristupljeno: 09.09.2020. Dosupno na:
https://github.com/vaquarkhan/vaquarkhan/wiki/Difference-between-scaling-horizontally-and-vertically

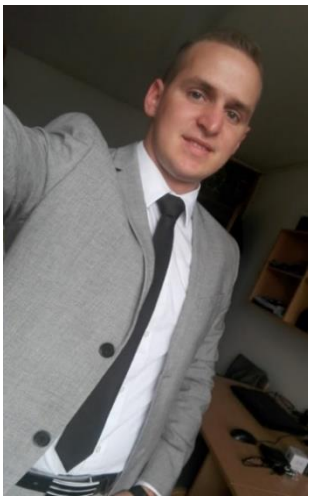
[5] Shweta Khare, Hongyang Sun,, Kaiwen Zhang, Julien Gascon-Samson i Aniruddha Gokhale
Ensuring Low-Latency and Scalable Data Dissemination for Smart-City Applications
Univezitet u Britanskoj Kolumbiji, Vankuver, 20.04.2018
Pristupljeno: 09.09.2020. Dostupno na:
http://www.dre.vanderbilt.edu/~gokhale/WWW/papers/IoTDI18_Poster.pdf

[6] Sefi Itzkovich
Reids, Kafka or RabbitMQ: Which MicroSerivces Message Broker to choose?
otonomo 29.05.2019
Pristupljeno: 09.09.2020. Dostupno na:
https://otonomo.io/blog/redis-kafka-or-rabbitmq-which-microservices-message-broker-to-choose/

[7] Margarete Rouse
MQTT (MQ Telemetry Transport)
internetofthingsagenda.techtarget.com, januar 2020

Prisutpljeno: 14.09.2020. Dostupno na:
https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport
[8] Paho projekat,
Eclipse foundation
Pristupljeno: 17.09.2020, dostupno na:
https://www.eclipse.org/paho/
[9] Mosquitto
Eclipse foundation
Pristupljeno: 18.09.2020, dostupno na:
https://mosquitto.org/
[10] Tiago Macedo i Fred Oliveira
Redis Cookbook,
O'reilly, Avgust 2011
[11] Redis specifikacija klastera
redis.io
Pristupljeno: 18.09.2020, dostupno na:
https://redis.io/topics/cluster-spec
[12] Neha Nerkhede, Gwen SHapira i Todd Palino
Kafka, Vodič definicije
O'reilly, Jul 2017
[13] The Apache Software Foundation
Pristpuljeno: 18.09.2020, dostupno na:
https://www.apache.org/
[14] Java dokumentacija, „The try-with-resources Statement"
Oracle, 2020.
Pristupljeno: 21.09.2020. dostupno na:
https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
[15] Apache Maven
Apache, septembar 2020.
Pristupljeno 21.09.2020. dostupno na:
https://maven.apache.org/
[16] Redis
redis.io
Pristpuljeno 21.09.2020. dostupno na:
https://redis.io/

Biografija



Robert Sabo, rođen je 22. Septembra 1995. godine u Somboru, Srbija. Osnovnu školu “Petar Kočić” u Riđici završio je 2010. godine, iste godine upisuje Srednju ekonomsku školu u Somboru. Srednju školu uspešno završava 2014. godine. Nakon završetka odlučuje da se upiše na akademske studije. 2014. godine upisuje Prirodno-matematički fakultet Univerziteta u Novom Sadu, na departmanu za matematiku i informatiku. Po diplomiranju 2018. godine upisuje master studije na istom departmanu.

UNIVERZITET U NOVOM SADU

PRIRODNO MATEMATIČKI FAKULTET

DEPARTMAN ZA MATEMATIKU I INFORMATIKU

KLJUČNA DOKUMENTACIJA

Redni broj:

RBR

Indentifikacioni broj:

IBR

Tip dokumentacije: **TD**	Monografska dokumentacija
Tip zapisa: **TZ**	Tekstualni štampani materijal
Vrsta rada: **VR**	Master rad
Autor: **AU**	Robert Sabo
Mentor: **MN**	dr Danijela Boberić Krstićev
Naslov rada: **NS**	Komunikacija u distribuiranim sistemima orijentisana porukama

Jezik publikacije: **JP**	Srpski/latinica
Jezik izvoda: **JI**	Srpski
Zemlja publikacije: **ZP**	Srbija
Uže geografsko područje: **UGP**	Vojvodina
Godina: **GO**	2020
Izdavač: **IZ**	Autorski reprint
Mesto i adresa: **MA**	Prirodno – matematički fakultet, Trg Dositeja Obradovića 4, Novi Sad
Fizički opis rada: **FO**	9 poglavlja, 75 strana, 38 slika, 6 listinga, 4 diagrama
Naučna oblast: **NO**	Informatika
Naučna disciplina: **ND**	Distribuirani sistemi

Predmetna odrednica/ **PO**	Kafka, Redis, MQTT
Ključne reči: **UDK:**	Komunikacija, brokeri poruka, MQTT, Kafka, Redis
Čuva se: **ČU**	Biblioteka Departmana za matematiku i informatiku, Novi Sad
Važna napomena:	Nema
Izvod **IZ**:	Poređenje različitih brokera orijentisanih porukama publish-subscribe paternom
Datum prihvatanja teme: **DT**	
Datum odbrane: **DO**	

Članovi komisiji:

1. dr Miloš Savić, vanredni profesor PMF-a, predsednik
2. dr Danijela Boberić Krstićev, vanredni profesor PMF-a, mentor
3. dr Jovana Vidaković, vanredni profesor PMF-a, član

UNIVERSITY OF NOVI SAD

FACULTY OF SCIENCES

DEPARTMENT OF MATHEMATICS AND INFORMATICS

KEY WORDS DOCUMENTATION

Accession number:

ANO

Identification number:

INO

Document type: Monograph type

DT

Type of record: Printed text

TR

Contents code: Master thesis

CC

Autor: Robert Sabo

AU

Mentor: PhD Danijela Boberić Krtićev

MN

Title: Message oriented communication
In distributed systems

XI

Language of text: **LT**	Serbian/Latinica
Language of abstract: **LA**	Serbian
Country of publication: **CP**	Serbia
Locality of publication: **LP**	Vojvodina
Publication year: **PY**	2020.
Publisher: **PU**	Author's reprint
Publication place: **PP**	PMF, Trg Dositeja Obradovića 4, Novi Sad
Physical description: **PD**	9 chapters, 75 pages, 38 pictures, 4 codes, 4 diagrams
Scientific field: **SF**	Informatics
Scientific discipline: **SD**	Distributed systems

Key words: **KW**	Communication, message broker, Redis, Kafka MQTT,
Holding data: **SD**	Library of Department of mathematics and informatics, Novi Sad
Note:	None
Abstract: **AB**	Comparing diffenet message oriented brokers based on publish-subscribe patern

Accepted by the Scientific Board on:

Defended on:

Thesys defend board:

1. PhD Miloš Savić, associate professor, Faculty of Sciences, Novi Sad, president
2. PhD Danijela Boberić Krstićev, associate professor, Faculty of Sciences, Novi Sad, menthor
3. PhD Jovana Vidaković, associate professor, Faculty of Sciences, Novi Sad, member