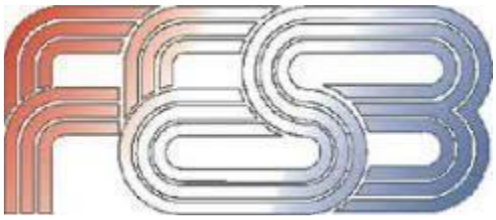


SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

KVALIFIKACIJSKI ISPIT

Moderna razmjena poruka za distribuirane računalne sustave

Student: Željko Šeremet

Split, zimski semestar 2016/2017.

SADRŽAJ

1 SAŽETAK I KLJUČNE RIJEČI

1.1 Sažetak

Rijetke moderne softverske aplikacije žive u izolaciji, a danas je to uobičajena praksa da se oslanjaju na usluge ili da koriste informacije sa udaljenih entiteta. U takvoj distribuiranoj arhitekturi, integracija je ključ. Više od jednog desetljeća, razmjena poruka je referenca rješenja za rješavanje izazova distribuirane prirode, kao što je mrežna nepouzdanost, snažno povezivanje proizvođača i potrošača i heterogenost aplikacija. Zahvaljujući jakoj zajednici i zajedničkim naporima na izradi standarda, brokeri poruka su danas transportni sloj građevnih blokova u mnogim projektima i uslugama, kako u zajednici fizičara tako i izvan nje. Štoviše, u posljednjih nekoliko godina se pojavila nova generacija usluga razmjene poruka s naglaskom na niske latencije i visoke performanse slučaja korištenja, pomičući granice aplikacija za razmjenu poruka. Ovaj rad će predstaviti rješenja razmjene poruka za distribuirane aplikacije kroz pregled glavnih koncepata, tehnologija i usluga.

1.2 Ključne riječi

Red poruka, MQ, ZeroMQ, broker poruka

2 UVOD

Tehnologije posredničkog sloja usmjerenog porukom (engl. Message Oriented Middleware – MOM) su vrlo važne za izgradnju složenih softverskih sustava. Ove tehnologije se koriste u povezivanju neovisnih softverskih sustava za kreiranje jedinstvenog sustava. Odnosno ove tehnologije dozvoljavaju razvoj umreženih softverskih sustava korištenjem različitih tehnologija i platformi kako bi se pokrenule na različitim hardverskim i/ili softverskim konfiguracijama.

Posrednički sloj usmjeren porukama se zapravo koristi za povezivanje postojećih s novorazvijenim softverom ili za komunikaciju između softverskih sustava različitih kompanija.

Red poruka je sustav za razmjenu poruka koji dozvoljava aplikacijama razmjenu podataka i drugih informacija. Odnosno redovi poruka se koriste za isporuku poruka od izvorišta do odredišta u slučaju skaliranja, što se smatra kritičnim dijelom korištenja reda poruke u produkcijskom sustavu pri povećanju opterećenja i nastavak operativnosti, čak i kad se pojavi kvar.

U radu je dan pregled koncepata, funkcionalnosti i modernih tehnologija razmjene poruka. U trećem poglavlju pojašnjena je razmjena poruka za distribuirane komunikacije i integraciju sustava. Zatim u četvrtom poglavlju je dan pregled glavnih značajki razmjene poruka, a u petom poglavlju pregled glavnih tehnologija za razmjenu poruka, od brokera do sustava „manjeg brokera“ (broker-less). Prije samog zaključka, u šestom poglavlju je dan popis uspješnih primjera korištenja razmjene poruka za rješavanje problema komunikacije za distribuirane aplikacije, a u sedmom poglavlju aktualna područja istraživanja.

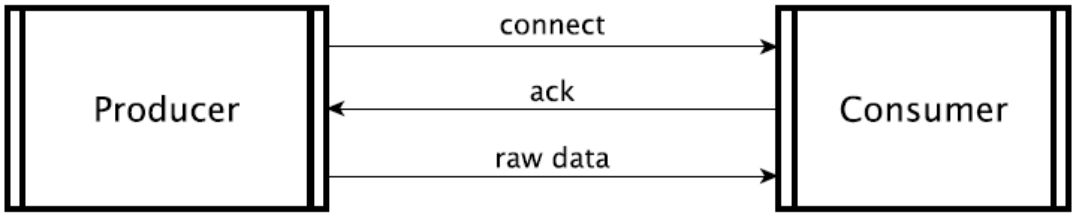
3 RAZMJENA PORUKA ZA SLABO POVEZANU KOMUNIKACIJU

Moderni distribuirani sustavi mogu biti sastavljeni od više stotina, ako ne i tisuća, aplikacija koje rade u više slojeva i pružaju jedni drugima različite usluge i funkcionalnosti. U takvoj distribuiranoj arhitekturi, postoje mnogi izazovi kao što je mrežna nepouzdanost, čvrsta povezanost proizvođača i potrošača i heterogenost aplikacija koje treba rješavati izgradnjom čvrstog i pouzdanog sustava.

3.1 Komunikacija usmjerena na spajanje (engl. Connection-oriented communication)

Komunikacija usmjerena na spajanje je jednostavno rješenje za razmjenu informacija između udaljenih entiteta. Na slici 3.1 je razmotreno otvaranje socketa preko protokola usmjerenog na spajanje (engl. connection-oriented), kao što je TCP/IP i prijenos sirovog podatkovnog toka preko njega. To bi bio brz i jeftin način za razmjenu informacija, ali u isto vrijeme to je čvrsto povezana komunikacija utemeljena na broju pretpostavki koje moraju biti zadovoljene da bi se komunikacija odvijala:

- **Vremenska ovisnost**: sve komponente moraju biti dostupne u isto vrijeme.
- **Lokacija**: svaka komponenta mora znati adrese drugih komponenti.
- **Struktura podataka i reprezentiranje**: u najjednostavnijoj implementaciji, sve komponente moraju imati isti format podataka i na binarnoj reprezentaciji.



Slika 3-1 Čvrsto povezana komunikacija

3.2 Razmjena poruka za slabo povezanu komunikaciju (engl. Messaging for loosely coupled communication)

Povezanost se može mjeriti kao broj pretpostavki koje se ostvaruju kada međusobno komuniciraju komponente sustava. Razmjena poruka je primjer slabo povezanog

komunikacijskog rješenja, gdje je *poruka* informacija građevnog bloka koja ima za cilj minimiziranje tih pretpostavki. Umjesto izravnog slanja informacija na određenu adresu, može biti poslano adresiranom *kanalu*, kako bi se riješila ovisnost o lokaciji. Da bi se odstranila vremenska ovisnost, kanal može biti pojačan postavljanjem informacije u red (engl. queue), dok udaljene komponente ne budu spremne da je prime. Na taj način, proizvođač može slati zahtjeve u kanal i dalje ih procesuirati bez brige o isporuci. Razmjene poruka ne daju nikakve pretpostavke zastupljenosti podataka. Tako da se standardni podatkovni format, npr. JSON ili XML što ih odlikuje samo-opisivanje i neovisnost o platformi, mogu koristiti za uklanjanje potrebe za dijeljenjem logike podatkovnog rukovanja između komponenti.

3.3 Scenarij razmjene poruka

Tipične upotrebe razmjene poruka su:

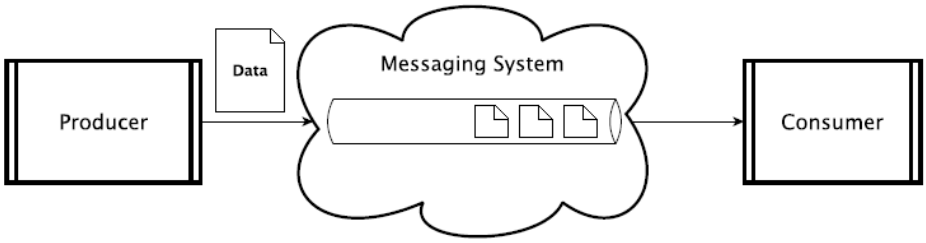
- *Information Publishing*: entitet objavljuje promjenjive informacije bez a-priori znanja o tome tko ih prihvaća (npr. senzor);
- *Information Storing*: entitet prikuplja informacije iz više izvora (npr. log collector);
- *Remote Procedure Call*: entitet šalje zahtjev prema jednom ili više udaljenih entiteta i očekuje odgovor.

3.4 Posrednički sloj razmjene poruka

Razmjena poruka je labavo povezano komunikacijsko rješenje koje minimizira ovisnost *proizvođač* i *potrošač*. Uklanjanjem te ovisnosti činimo cjelokupnu arhitekturu fleksibilnijom i tolerantnijom na promjene, ali s dodatnom složenosti. Dakle, dedicirani posrednički sloj razmjene poruka je razvijen tijekom godina osiguravanjem funkcionalnosti razmjene poruka bez potrebe da se bave unutarnjom složenosti. Sljedeće poglavlje opisuje glavne koncepte i principe sustava za razmjenu poruka.

4 SUSTAVI ZA RAZMJENU PORUKA

Sustav za razmjenu poruka, kao što je prikazano na slici 4.1, djeluje kao neusmjereni sloj između entiteta koji žele komunicirati. Obično se spominje kao *broker poruka*. On je odgovoran za prijenos podataka, kao poruka, iz jedne aplikacije u drugu, kako bi se proizvođač i potrošač mogli usredotočiti na ono što dijele, a ne na kako ga podijeliti. Mnoge druge tehnologije za razmjenu poruka temelje se na nekim osnovnim konceptima i svojstvima koji su dijeljeni između različitih usluga i implementacija.



Slika 4-1 Razmjena poruka za slabo povezanu komunikaciju

4.1 Poruka

Poruka je informacija o građevnom bloku. Ona se sastoji od *tijela*, koje je nepromjenjivo i sadrži strukturirane podatke (u JSON, XML, serijalizacijskim protokolima) komunikacijskog objekta i skupa *zaglavlja*, obično *key value pairs* koje broker može procesuirati i koristiti za usmjeravanje.

4.2 Komunikacijski modeli: tema i red

Sustavi za razmjenu poruka podržavaju različite komunikacijske modele, odnosno svatko definira kako se informacije razmjenjuju između proizvođača i potrošača. Najčešći zajednički komunikacijski modeli su *redovi* i *teme*. Red se koristi za implementaciju point-to-point komunikacije. U kojoj se ako ne postoje potrošači kada je proizvedena informacija, poruku čuva u kanalu za kasniju isporuku. Dok se ako ima više potrošača, poruka isporučuje samo jednom. Model tema se odnosi na klasični publish/subscribe scenarij, u kojem se, ako ne postoji potrošač, poruku odbacuje, a u slučaju više potrošača, sustav za razmjenu poruku dostavlja svakoj od njih. Osim *reda* i *teme* koje su široko podržane, kompleksnije semantike

isporuka postoje na razini protokola (npr. razmjena/čvorova iz *engl. Advanced Message Queuing Protocol – AMQP*), kao i mnoge druge koje su specifične na posredničkoj razini.

4.3 Protokoli

Nedostatak jedinstvenog standardnog načina interakcije sa brokerima poruka dugo godina je poznat problem za tehnologije razmjene poruka. AMQP protokol je dizajniran od strane glavnih aktera razmjene poruka, tvrtki i proizvođača softvera, kako bi riješili ograničenje. Ipak, unutarnja složenost jedinstvenog protokola koja definira: žicu komunikacije i isporuku semantika, zahtijeva glavni razvojni napor za sustav razmjene poruka da postane u potpunosti kompatibilan. Ovo podpoglavlje donosi pregled najčešćih standardnih protokola koji su danas podržani od strane glavnih sustava za razmjenu poruka. Izbor protokola je krucijalna dizajnerska odluka za arhitekturom orijentiranom porukom, u odnosu na čvrsta spajanja unutar aplikacije.

4.3.1 AMQP (engl. Advanced Message Queuing Protocol)

AMQP (engl. Advanced Message Queuing Protocol) [1] je rezultat standardizacijskog napora glavnih suradnika na sceni razmjene poruka (npr. Cisco, Microsoft, Red Hat, banke). Dizajniran je za interoperabilnost između različitih sustava za razmjenu poruka. Zapravo, AMQP pruža definiciju za binarne žičane protokole i kompletnu semantičku isporuku. Teoretski, AMQP klijenti za razmjenu poruka su u mogućnosti s lakoćom komunicirati s različitim broker implementacijama, pri čemu su kompatibilni sa AMQP. Usvajanjem najnovije stabilne verzije (1.1) protokola koja još nije opsežna, ali s obzirom da je već podržana od strane glavnih broker poruka, puno šira primjena očekuje se u nadolazećim godinama. [2][3]

4.3.2 STOMP (engl. Streaming Text Oriented Messaging Protocol)

STOMP (engl. Streaming Text Oriented Messaging Protocol) [4] je protokol baziran na tekstu što znači da će biti jednostavan i naširoko interoperabilan. On je uglavnom žičani protokol te dolazi sa osnovnom ugrađenom semantikom razmjene poruka (npr. bez podrške za komunikacijske modele, odredište je predstavljeno stringom u zaglavlju poruke). Zahtijevajući odgovarajuću konfiguraciju na razini sustava poruka (npr. odredište mora biti na odgovarajući način mapirano u red ili temu). Zahvaljujući njegovoj jednostavnosti postoji

opsežan skup klijenata dostupnih na mnogim jezicima, a podržani su od strane većine brokera. [5]

4.3.3 MQTT (engl. Message Queue Telemetry Transport)

MQTT (engl. Message Queue Telemetry Transport) [6] je lagani protokol dizajniran izvorno iz IBM-u. To bi značilo da je dizajniran za nisku propusnost i visoke latencije mreža. Ovaj protokol definira kompaktni binarni format sa vrlo ograničenim troškom na komunikaciji (nekoliko desetaka bajta) što ga čini pogodnim za *Internet stvari (engl. Internet of Things – IoT)* u stilu aplikacija (npr. mobilni telefoni, senzori) u jednostavnom *produce-and-forget* scenariju.

4.3.4 OpenWire

Protokol OpenWire je binarni format koji se koristi za prijenos naredbi preko transportnog sloja (kao što je TCP) brokera. Naredbe su poruke i poruke potvrde, kao i upravljanje i kontrola brokera. OpenWire protokol podržava klijente razvijene u različitim jezicima, kao što su: Java, C# ili nativni C. OpenWire je dizajniran za maksimalne performanse i značajke, najčešće korišten kao protokol u ActiveMQ. [3] [7]

4.3.5 Representational State Transfer (REST)

Fielding [8] je smislio termin „Representational State Transfer“ (REST) kao arhitektonski stil za distribuirane hipermedijske sustave. REST za resursno orijentirane arhitekture je postala ključna tehnika u Web i cloud uslugama kako bi se postigla jednostavnost, skalabilnost i djeljivost. U svom izvornom obliku REST je prilično apstraktan i nije ograničen na određene protokole. Međutim, načela su izvedena iz uspješnih Web arhitektura i ranih HTTP verzija. Zato se i danas REST prvenstveno odnosi na HTTP kao transportni mehanizam.

4.3.6 Extensible Messaging and Presence Protocol (XMPP)

XMPP [9][10][11] je namijenjen kao otvoreni standard za slanje izravnih poruka, informacija o prisutnosti i održavanju popisa kontakata u chat aplikacijama, također ima i svojstva posredničkog sloja. U osnovnoj specifikaciji XMPP razmjenjuje poruke kao XML

strofe u klijent-servis i servis-servis komunikaciji za udružene usluge. XMPP usluga, dakle preuzima ulogu posrednika.

XMPP je posebno atraktivan u MOM scenariju, gdje su Web agenti uključeni jer podržavaju HTTP kao transportni mehanizam, a većina web-preglednika i JavaScript okruženja su sposobni za obradu XML strofa. Nadalje, XMPP se također smatra prikladnim protokolom razmjene poruka za aplikacije *Internet stvari* [12]. Protokol je proširiv i ekstenzije su navedene u razvojnoj zajednici. MOM-specifične ekstenzije su:

- Prijenos kodiran na bazi 64-binarnog sadržaja sa dodijeljenim MIME media tipom [13];
- RPC preko XMPP [14];
- Otkrivanje usluga [15];
- Publish-subscribe [16] za broker scenarije, prošireno adresiranje [17] za usmjeravanje poruka i ekstenzija za obavijesti o događajima [18] [19];
- Pouzdani prijenos poruka [20]; i
- SSL/TLS zaštićeni transportni mehanizam i S/MIME [21] za end-to-end šifiranje poruka.

Prema zadanim postavkama, poruke u XMPP su XML strofe, a tijela su ograničena samo na tekst. Postoji pojam vrste poruke, ali je ograničeno na izravne aplikacije za razmjenu poruka.

XMPP može poslužiti i kao infrastruktura razmjene poruka za SOAP/WS-* Web usluge [22]. Uz izravnu razmjenu poruka, XMPP je uspješno angažiran u VIRTUS posredničkom sloju za aplikacije *Internet stvari* [23] koristeći kolaboracijski poslužiteljski OpenFire [24] softver u stvarnom vremenu. Drugi softver koji nudi XMPP razmjenu poruka preko Web-Socket je Kaazing WebSocketa Gateway [25].

4.4 Sposobnosti

Kao što je predstavljeno u poglavlju 3, sustav za razmjenu poruka može se smatrati srednji slojem koji je pojačan dodatnim značajkama, kao što je čekanje u redu, što poboljšava komunikacijsko iskustvo udaljenih entiteta. Tijekom godina, iako ne postoji formalni sporazum, različiti sustavi za razmjenu poruka konvergiraju preko zajedničkog skupa

sposobnosti koji postaju de-facto standard za posrednički sloj razmjene poruka. Popis značajki uključuje:

- *Postojanost* – sposobnost spremanja poruke na trajnu pohranu, kao što su datotečni sustavi ili baze podataka;
- *Nadilaženje grešaka* – omogućuje klijentima da se automatski ponovo spoje u slučaju kvara na brokeru;
- *Garantirana isporuka* – definira politiku za isporuku poruke (npr. *barem jedanput* ili *točno jednom*);
- *Naručivanje* – dostavljanje poruke koja je proizvedena;
- *Transakcija* – sposobnost da razmotri više zahtjeva kao dio distribuirane transakcije, sa roll-back opcijom;
- *Klasteriranje* – mogućnost stvaranja mreže broker poruka za visoku dostupnost i uravnoteženje opterećenja.

Ipak, svaki sustav za razmjenu poruka može primijeniti različite interpretacije za iste značajke. Postoje mnoge druge *jedinstvene* karakteristike specifičnog brokera, ali njihova uporaba podrazumijeva čvrsto povezivanje aplikacija sa određenom broker uslugom.

5 TEHNOLOGIJE RAZMJENE PORUKA

Posrednički sloj usmjeren porukom razvijen je prije više od jednog desetljeća kao bogat i čvrst ekosustav usluga i biblioteka. Brokeri poruka su kao srednje samostalne usluge koje nude sposobnost razmjene poruka u distribuiranim aplikacijama. Obično su najčešći zajednički tip sustava za razmjenu poruka. Brokeri poruka se intenzivno koriste već nekoliko godina za implementiranje komunikacije i integracije u distribuiranom sustavu [26], sa izuzetkom podatkovne intenzivnosti i visokih performansi slučaja korištenja, gdje postojanje srednjeg entiteta nije prikladno rješenje. U posljednjih nekoliko godina pojavile su se nove generacije sustava poruka, s naglaskom na niske latencije i korištenje visokih performansi slučajeva korištenja, pomičući granice aplikacija prema razmjeni poruka. Sljedeći dio će dati pregled glavnih tehnologija za razmjenu poruka.

5.1 Brokeri poruka

Brokeri poruka su najčešće zajedničke implementacije sustava za razmjenu poruka. Broker poruka je samostalni srednji entitet koji nudi funkcionalnost razmjene poruka putem standarda ili prilagođenog protokola. Mnogo broker poruka postoji, a razlikuju se u sposobnostima, protokolu, implementaciji jezika i podržanoj platformi. U ovom poglavlju dan je pregled open-source rješenja, kao i komercijalni softver poduzeća.

Brokeri poruke su većinom značajka bogatih tipova sustava za razmjenu poruka, u terminu sposobnosti podrške protokola kao što je opisano u poglavlju 3. Brokeri mogu biti *poliglot*, dopuštajući da proizvođač i potrošač koriste različite protokole (npr. pošiljatelj na AMQP, primatelj na STOMP) i oni mogu podržati transformaciju poruke (npr. transformirajuću poruku nosivosti iz XML u JSON) da se dodatno razdvoje aplikacije.

5.1.1 ActiveMQ

ActiveMQ je jedan od najšire usvojenih open-source brokera poruka. To je Apache projekt, napisan u Javi, a komercijalno je podržan od strane Red Hat. ActiveMQ ima veliku podršku protokola (npr. AMQP, STOMP, MQTT, Openwire, HTTP i mnogi drugi), pruža mnoge poprečne jezične klijente te je u potpunosti JMS kompatibilan. ActiveMQ nudi mnoge napredne mogućnosti, kao što je bogata dostava semantike (npr. virtualni redovi, kompozitno odredište, zamjenski znakovi), JDBC spremište poruka (npr. stalne poruke u bilo kojoj JDBC

kompatibilnoj bazi podataka) i napredna konfiguracija klasteriranja (npr. master-slave, mreža brokera). ActiveMQ je značajka kompletnog rješenja za razmjenu poruka koje se mogu koristiti za implementaciju velikog broja komunikacija i integracija uzoraka [26].

5.1.2 RabbitMQ

RabbitMQ je lagani open-source broker poruka napisan u Erlangu koji profitira iz sposobnosti razmjene poruka iz nižih jezika. RabbitMQ arhitektura je duboko modularna, što uglavnom podržava AMQP i STOMP, ali i dodatne protokole koji mogu se učitati kao plug-in (npr. MQTT, HTTP). Podržava glavne sposobnosti razmjene poruka, kao što su perzistentnost, klasteriranje, visoka dostupnost i federacija. RabbitMQ ostaje lagano rješenje razmjene poruka koje se može naći ugrađeno u nekoliko projekata (npr. Logstash) sa svojom jednostavnošću i pouzdanošću. [27][28][29]

5.1.3 IBM Websphere Message Broker

WebSphere Message Broker se koristi u implementacijama arhitektura aplikacijske integracije, jer pruža mehanizam za povezivanje, preusmjeravanje i transformiranje poslovnih podataka iz različitih prijenosa bez ikakvih promjena na osnovnoj aplikaciji koja generira podatke.

WebSphere Message Broker poboljšava protok i distribuciju informacija omogućujući transformaciju i inteligentno usmjeravanje poruka bez potrebe za promjenama aplikacija koje generiraju poruke ili aplikacija koje ih koriste. U WebSphere Message Broker, povezivost pruža aplikacijama komuniciranje pomoću slanja i primanja poruka.

WebSphere Message Broker također ima sljedeće ključne sposobnosti koje su vrijedna rješenja za poslovne integracije:

- Raspoređuje bilo koju vrstu podataka preko i između više različitih sustava i aplikacija, osiguravanjem isporuke ispravnih podataka u ispravnom formatu i u ispravnom trenutku;
- Smanjuje broj point-to-point međupovezivanja i pojednostavljuje programiranje aplikacija uklanjajući integracijsku logiku iz aplikacija

- Usmjeravanje informacije u stvarnom vremenu se temelji na temi i sadržaju do bilo koje krajnje točke korištenjem snažnog publish/subscribe mehanizma razmjene poruka
- Potvrđivanjem i pretvaranjem poruke u letu između bilo koje kombinacije različitih formata poruka, uključujući i Web servise, te ostale XML i ne-XML formate
- Usmjeravanje poruka se temelji na (procjenama) poslovnih pravila prema jednakim sadržajima podataka i poslovnih procesa
- Poboljšava poslovnu agilnost dinamičkim rekonfiguriranjem informacija distribucije obrazaca bez reprogramiranja krajnje točke aplikacija
- Kontrola pristupa na siguran način dostavlja personalizirane informacije na pravo mjesto u pravom trenutku [30]

5.1.4 Xively (formerly Cosm)

Xively IoT Platforma se sastoji od modela skalabilnog objekt direktorija nazvanog Blueprint. Baziran na brzom MQTT brokeru poruka i sigurnom procesu opremanja, koji podržava milijune sigurnih veza između ljudi, uređaja i podataka širom svijeta. Xively koristi Heroku [31] za razvoj daljinskog povezivanja u nekim aplikacijama, što omogućuje korisnicima da kontroliraju klijent uređaje iz web preglednika ili mobilnog uređaja s bilo kojeg mjesta u svijetu koji je spojen na Internet.

Karakteristike platforme:

- Značajke za povezivanje

Xively IoT Platforma je munjevito brza i beskrajno skalabilna. Platforma posjeduje softverska i hardverska rješenja kako bi pomogli procesuirati preko 86 milijardi poruka dnevno. I sve te veze su izgrađene kroz trinaest godina nasljeđa sigurnosti i pouzdanosti.

- Upravljanje značajkama

Xively smanjuje složenost pokretanja povezanog poslovanja i pruža sve što je potrebno za svoje povezane proizvode, uključujući rezerviranja, praćenje, ažuriranje i upravljanje korisnicima.

- Angažiranje značajki

Xively pretvara podatke i odnose u mjerodavnim uvidima koji omogućuju da se pokrene spojna tvrtka proizvoda. Xively ne pretpostavlja da su podaci proizvoda na dohvat ruke, ali se integriraju s drugim poslovnim sustavima za pogon nove poslovne automatizacije i vrijednosti.

- Profesionalne usluge

Korištenje prvog poslovnog pristupa, može se ubrzati transformacija tvrtke u povezani posao, tako da se može brzo povećati prodaja, optimizirati usluga i oduševiti korisnike i kupce. [32]

5.1.5 Hive MQ

HiveMQ je MQTT broker posebno prilagođen za poduzeća koja se nalaze u dobi nastajanja Machine-to-Machine komunikacije (M2M) i *Internet stvari*. Izgrađen je iz temelja uz maksimalnu skalabilnost i poslovno spremne sigurnosne koncepte. HiveMQ implementira Message Queue Telemetry Transport protokol koji predstavlja de facto M2M messaging standard (u potpunosti usklađen sa specifikacijom), vodeći kad je u pitanju profesionalno donošenje svih mogućnosti *Internet stvari* za tvrtke. [33]

5.1.6 IBM Lotus Expeditor micro broker

Lotus Expeditor mikro broker komponenta je mala broker poruka koja omogućuje razmjenu poruka za integriranje raznih dijelova rješenja. Posebno je usmjeren na resurs ograničenih okruženja, kao što su ona koja se mogu naći na „rubu mreže“ uređaja na kojima je pokrenuta aplikacija koja mjeri i kontrolira fizikalna svojstva (npr. temperatura, protok tekućine, …) pomoću senzora i aktuatora. Broker poruka osigurava da poruke pristižu na točne destinacije i pretvaraju se u format prikladan za svako odredište. Mikro broker je implementiran u Javi i radi na širokom rasponu standardnih uređaja, uključujući PDA, prijenosna računala i stolna računala. Osim toga, mikro broker radi na više specifičnih uređaja, kao što su programabilni logički kontroleri (PLC), informacijska čvorišta pametnih kuća (primjerice, TV set-top box) i auto-montirani uređaji.

Mikro broker je pogodan za ugradnju u aplikacijama i rješenjima koja imaju potrebu za razmjenom poruka, obavijesti i serivisnih događaja što omogućuje korištenje modela razmjene poruka publish-subscribe i point-to-point. Infrastruktura razmjene poruka pruža

lagane klijente razmjene poruka za komuniciranje jednih s drugima, unutar jednog uređaja preko mreže te za integraciju s poduzetničkim brokerima. Broker poruka omogućuje tvorničku end-to-end integraciju, dosegnuvši iz senzora i aktuatora uređaja za klijentske aplikacije do back-end aplikacija. Mikro broker pruža mogućnost integracije u blizini ruba mreže, kao i pružanje prolaza prema Enterprise Services Bus (ESB). [34]

5.1.7 Moquette

Moquette je Java implementacija MQTT 3.1 brokera. Njegova kodna baza je mala. U svojoj srži, Moquette predstavlja procesor događaja; što omogućuje da kodna baza bude jednostavna, izbjegavajući probleme dijeljenja niti.

Moquette broker je lagan i jednostavan za razumjeti kako bi se mogao ugraditi u druge projekte. Pretpostavljen je samostalan, ali može se integrirati u OSGi kontejner stvaranjem značajnijih integracija, na primjer može se pokrenuti unutar ugrađenog OSGi brokera kao što je Concierge. [35]

5.1.8 Mosquitto

Eclipse Mosquitto je open-source (EPL/EDL) licenca broker poruka koji implementira protokol verzije MQTT 3.1 i MQTT 3.1.1. MQTT pruža lagani način obavljanja razmjene poruka pomoću publish/subscribe modela. To ga čini pogodnim za razmjenu poruka *Internet Stvari* kao što su senzori male snage ili mobilni uređaji kao što su telefoni, ugrađena računala ili mikrokontroleri kao što je Arduino. [36]

5.1.9 RSMB – Really Small Message Broker

Really Small Message Broker je mali poslužitelj koji koristi MQ Telemetry Transport (MQTT) (verzija 3) za laganu, nisku potrošnju razmjene poruka. Really Small Message Broker omogućuje razmjene poruka između sitnih uređaja kao što su senzori i aktuatori preko mreža koje bi mogle imati nisku propusnost, visoke cijene i različitu pouzdanost. „Izdavači“ šalju poruke brokeru koji ih potom distribuira „pretplatnicima“, koji su zatražili primanje tih poruka.

Really Small Message Broker se odlikuje „mostom“, koji omogućuje povezivanje s drugim sposobnim MQTT poslužiteljima; ovaj most omogućuje prenošenje poruka između Really

Small Message Broker instanci, kao i na druge brokere kao što su Lotus Expeditor micro broker („Microbroker“) i WebSphere Message Broker.

Oba brokera, Microbroker i Really Small Message Broker, mogu biti pokrenuti u ugradbenim sustavima kako bi se osigurala infrastruktura za razmjenu poruka u udaljenim instalacijama i pervazivnim okruženjima. Međutim, Really Small Message Broker treba oko 100 puta manje memorije za rad od mikro brokera; dakle, može se dalje proširiti doseg MQTT infrastrukture razmjene poruka.

Really Small Message Broker također olakšava povezivanje brokera, bilo u u peer-to-peer ili hijerarhijsku infrastrukturu razmjene poruka. Really Small Message Broker ima laku konfiguraciju i niske zahtjeve za resurse pružajući veliku fleksibilnost u rješenjima razmjene poruka za ugradbena okruženja.

Really Small Message Broker pruža lagani poslužitelj dizajniran za distribuciju poruke između aplikcija. [37]

5.1.10 HornetQ

HornetQ je open-source projekt za izgradnju multiprotokolnog, ugradbenog, klasteriranog, asinkronog sustava vrlo visokih performansi za razmjenu poruka. HornetQ je primjer Posredničkog sloja usmjerenog porukom (MoM).

Razlozi korištenja HornetQ su slijedeći:

- 100% open-source softver. HornetQ je licenciran pod Apache Software License v2.0 kako bi se smanjile barijere usvajanja.
- HornetQ je dizajniran sa naglaskom na upotrebljivosti.
- Napisan u Javi. Radi na bilo kojoj platformi uz izvođenje Java 6+ i to od Windows računala prema IBM mainframe-ovima.
- Nevjerojatne performanse. Revolucionarno visoke performanse pružaju perzistentne performanse razmjene poruka po stopama u redovno viđenim neperzistentnim razmjenama poruka. Neperzistentne performanse razmjene poruka može se poistovjetiti s previše udaranja stijena u brod.

- Cijeli skup značajki. Sve značajke se očekuju u svakom ozbiljnom sustavu za razmjenu poruka, a drugi se neće pronaći nigdje drugdje.
- Elegantan dizajn s minimalnom zavisnosti treće strane. Pokrenuti HornetQ je samostalan, pokrenut u integriranom po izboru JEE aplikacijskom poslužitelju ili pokrenut ugrađen u vlastitom proizvodu.
- Bešavne visoke dostupnosti. Pruža visoko dostupno (engl. High-availability – HA) rješenje s automatskim oporavljanjem klijenta od pogrešaka, tako da se može osigurati dupliciranje u slučaju kvara poslužitelja.
- Iznimno fleksibilno klasteriranje. Stvaranje klaster poslužitelja koji znaju kako uravnotežiti poruke. Poveznicom geografski distribuiranih klastera preko nepouzdanih veza za formiranje globalne mreže. Konfiguracija preusmjeravanja poruka na izrazito fleksibilan način. [38]

5.1.11 ActiveMQ Apollo

Apache ActiveMQ Apollo dobiva konkurenciju iz vlastite tvrtke. Apollo je svoj razvoj utemeljio na iskustvu projekta ActiveMQ.

Uvedena je potpuno nova arhitektura da bi bio brži, robustniji i lakši za održavati nego ActiveMQ. Ta arhitektura se temelji na Scala programskom jeziku koja podržava i razvoj paralelnih sustava. Višenitnost od Apollo brokera bitno se razlikuje od ActiveMQ. Svi zadaci se izvršavaju asinkrono i na neblokirajući način koji pridonosi povećanju performansi i stabilnosti. To znači da su vještine višejezgrenih procesora bolje iskorištene.

Za Apollo postoje dva spremišta za perzistentne poruke. S jedne strane NoSQL baza podataka LevelDB od strane Googlea, a sa druge strane, preko Java Edition Berkeley DB.

Iako je Apollo napisan u Scali, broker se može koristiti u Java okruženju. S činjenicom da je Apollo realiziran u Scali, može se vidjeti da distribucija također sadrži i neke Scala biblioteke među mnogim Java bibliotekama.

Nema klijent biblioteke za samog Apolona. Stoga se ostali klijenti mogu koristiti za slijedeće protokole: MQTT, OpenWire i Stomp. [39]

5.1.12 Apache Qpid

Osim ActiveMQ i Apollo, postoji još jedan Apache Message Broker, Apache Qpid. Cilj projekta Qpid je potpuna kompatibilnost sa Advanced Message Queuing Protocol Standardom.

Qpid broker je dostupan za C++ i Java. Za Java klijente postoji JMS API. Za C++, Python i Microsoft .NET postoji *Qpid Messaging API*. Za perzistentnost poruka podržane su relacijske Apache Derby i Oracle Berkeley DB baze podataka. Na temelju Apache Qpid, Red Hat nudi Enterprise Messaging proizvod, MRG.

Proton (potprojekt od Qpid) je lagana implementacija AMQP protokola. Korištenjem protona također je moguće razviti klijente i poslužitelje. Proton je dostupan za C i Java programske jezike. Implementacija C također uključuje povezivanja za PHP, Python i Ruby. [40]

5.1.13 JBoss Messaging

JBoss Messaging je sustav za razmjenu poruka poduzeća iz JBoss. On je potpuno prepisan od JBossMQ, naslijeđen od JBoss Java Message Service (JMS) davatelja. JBoss Messaging je zadani JMS davatelj u JBoss Enterprise Application Platform 4.3 i 5. JBoss Messaging je cjelokupna strategija razmjene poruka Red Hat-a. Ona nudi poboljšanja performansi kako na jednom čvoru tako u klasteriranom okruženju, a isto tako ima modularnu arhitekturu, tako da se lako može dodati više značajki u budućnosti.

JBoss Messaging pruža open-source i bazirane standarde messaging platforme donesene klasi poduzeća za razmjenu poruka na širokom tržištu. JBoss Messaging implementira robustne, visokih performansi jezgre razmjene poruka, dizajnirane za podršku Service-Oriented Architectures (SOA), Enterprise Service Bus (ESBs) te drugih uvjeta za integraciju, bez obzira na razinu potražnje.

JBoss Messaging omogućuje distribuiranje aplikacija ravnomjerno u cijelom klasteru. On uravnotežuje svaki čvor CPU ciklusa bez ijedne točke kvara, pružajući visoku skalabilnost i visoke performanse implementacije klasteriranja .

JBoss Messaging uključuje Java Messaging Service (JMS) front-end, tako da su poruke dostavljene u formatu utemeljenom na standardima, a koji će omogućiti podršku za druge protokole za razmjenu poruka u budućnosti.

JBoss Messaging omogućuje sljedeće značajke:

- Snažan fokus na performanse, pouzdanost i skalabilnost uz visoku propusnost i nisku latenciju.
- Osnovan za JBoss ESB za SOA inicijative. (JBoss ESB koristi JBoss Messaging kao zadani JMS davatelj.)

JBoss Messaging također uključuje:

- publish-subscribe i point-to-point modele razmjene poruka;
- perzistentne i neperzistentne poruke;
- garantirana isporuka poruka koja osigurava da poruke stignu jednom i samo jednom gdje to bude zatraženo;
- transakcijska i pouzdana sučelja koje podržavaju ACID semantike;
- prilagodljivi JAAS bazirani sigurnosni okvir;
- potpuna integracija s JBoss Transactions (ranije Arjuna JTA) podržana sa potpunim transakcijskim oporavakom;
- opsežno sučelje za upravljanje JMX;
- podrška za većinu velikih baza podataka, uključujući Oracle, DB2, Sybase, Microsoft SQL Server, PostgreSQL i MySQL;
- HTTP prijenos, za korištenje firewall koji omogućuju samo HTTP promet;
- servlet prijenos omogućuje razmjenu poruka kroz namjenski servlet;
- SSL prijenos;
- podesivi Dead Letter Queues (DLQs) i Expiry Queues;
- statistike poruke koja pruža valjani povijesni pregled na poruke isporučene u redove i pretplate;
- automatsko straničenje poruka na prostoru za pohranu, što omogućuje upotrebu vrlo velikih redova koji će biti preveliki da bi se u potpunosti uklopili u memorijski sustav; i

- precizno naručivanje poruke što rezultira u porukama koje pripadaju u određene skupine poruka kako bi bile dostavljene prema redoslijedu njihovog dolaska na ciljani red.

JBoss Messaging također uključuje sljedeće značajke klasteriranja:

- Potpuno grupirani redovi i teme
  Logički redovi i teme se distribuiraju preko klastera. Mogu se poslati ili primiti u red ili temu iz bilo kojeg čvora na klasteru.
- Potpuno grupirane dugotrajne pretplate
  Posebno dugotrajnoj pretplati se može pristupiti s bilo kojeg čvora klastera, omogućujući proširenje obrade opterećenja od određene pretplate preko cijelog klastera.
- Potpuno klasterirani privremeni redovi
  Ako poslana poruka uključuje replyTo privremeni red, može se vratiti na bilo koji čvor klastera.
- Inteligentna preraspodjela poruke
  Poruke se automatski prebacuju između čvorova klastera kako bi iskoristili različite brzine potrošača na različitim čvorovima. To pomaže da se spriječi izgladnjivanje ili izgradnja poruka na određenom čvoru.
- Zaštita poruke redoslijedom
  Ovo omogućuje da redoslijed poruka proizvedenih od proizvođača bude identičan poretku poruka konzumiranih od potrošača. To funkcionira čak ako je aktivna preraspodjela poruke.
- Potpuno transparentan failover
  Kada poslužitelj ima grešku, sesija i dalje nastavlja bez iznimke na novi čvor. Ovo je također potpuno podesivo: ako se ne želi implementirati ovo failover ponašanje, možete ga isključiti i vratiti na iznimke koje mogu odbaciti i ručno ponovno povezati na novi čvor.
- Visoka dostupnost i bešavni failover
  Ako čvor ima grešku, automatski će failover proslijediti na drugi čvor bez gubitka bilo koje perzistentne poruke te može bez problema nastaviti sesiju. Jednom i samo jednom se isporučuje perzistentna poruka u svakom trenutku.

- Most poruke

  JBoss Messaging sadrži komponentu most poruke, koja omogućuje premošćivanje između bilo koja dva JMS 1.1 odredišta. To omogućuje povezivanje geografski odvojenih klastera i čini velike, globalno distribuirane logičke redove i teme. [41]

5.1.14 Sun Java System Message Queue

Sun JavaTM sustav reda poruka je vodeća poslovna integracija sustava za razmjenu poruka dizajnirana da pruži iznimnu pouzdanost i skalabilnost.

Red poruka je proizvod posredničkog sloja razmjene poruka koji implementira Java Message Service (JMS) standard. Osim toga, red poruka osigurava sposobnost poduzetničke čvrstoće, uključujući napredne integracije, administraciju, sigurnost i značajke visoke dostupnosti.

Red poruka može se koristiti kao samostalni servis za razmjenu poruka ili se može koristiti kao omogućujuća tehnologija. Red poruka raspoređen u Java EE aplikacijskom poslužitelju omogućuje asinkronu razmjenu poruka. On je sastavni dio omogućujuće tehnologije Glassfish Application Server (bivši Sun Java Application Server), a također je ključna komponenta posredničkog sloja Java enterprise softver sustava. [42]

5.1.15 SonicMQ

SonicMQ je jedan od novijih MOM-ova na tržištu. On je prvi implementirao JMS, a ima i izvrsne konkurentne performanse s postojećim MOM tehnologijama, kao što su IBM MQSeries. SonicMQ je napisan u potpunosti u Javi, podržava XML razmjenu poruka i HTTP tuneliranje kako bi omogućilo SonicMQ za rad preko Interneta. Temeljni mehanizam SonicMQ je njegov „broker“ koji olakšava kretanje poruka na mreži. To je Java izvršna datoteka koja zahtijeva pokretanje Java Virtual Machine (JVM). Komunikacijski protokoli koji se mogu koristiti sa SonicMQ su TCP, HTTP i SSL. Budući da koristi zajednički Internet protokol, SonicMQ može proširiti svoju implementaciju na Internet. Ona također pruža *mostove* na mnoge druge popularne MOM-ove koji dopuštaju da poruke budu poslane i primljene između SonicMQ i drugih MOM-ova. [43]

5.1.16 Microsoft's MSMQ

Tehnologija reda poruka (Microsoft Message Queuing – MSMQ) omogućuje pokretanje aplikacija u različitim vremenima za komunikaciju preko heterogenih mreža i sustava koji mogu biti privremeno odsutni. Aplikacije šalju poruke u redove i čitaju poruke iz redova.

Red poruka utemeljen na prioritetu osigurava zajamčenu isporuku poruka, učinkovito usmjeravanje, sigurnost i razmjenu poruka.

MSMQ se može koristiti za implementaciju rješenja za asinkrone i sinkrone scenarije koji zahtijevaju visoke performanse. Sljedeći popis pokazuje nekoliko mjesta gdje se mogu koristiti redovi poruka:

- Kritične financijske usluge: npr. elektronička trgovina
- Ugradbene i ručne aplikacije: npr. u osnovi komunikacija između ugrađenih uređaja koji usmjeravaju prtljagu do zračne luke pomoću automatskog sustava za prtljagu
- Vanjska prodaja: npr., aplikacije za automatiziranu prodaju na putovanjima prodajnih predstavnika
- Workflow (radni proces): Redanje poruka olakšava kreiranje radnog procesa koji ažurira svaki sustav. Tipičan dizajn obrasca je implementacija agenta za interakciju sa svakim sustavom. Korištenje agentske arhitekture radnog procesa također se smanjuje utjecaj promjena jednog sustava na druge sustave. Sa redanjem poruka, slaba povezanost između sustava čini jednostavniju nadogradnju pojedinačnih sustava. [44]

5.1.17 Oracle Streams Advanced Queuing (AQ)

Za razliku od većine upravljanja reda proizvoda koje su neovisne posredničke komponente, Oracle Streams AQ je redanje takvih objekata izgrađenih u Oracle Database. On je izgrađen na vrhu Oracle Streams, a omogućava širenje i upravljanje informacija u toku podataka, bilo unutar baza podataka ili između baza podataka. AQ se može pristupiti iz najpopularnijih programskih jezika putem API-ja za PL/SQL, Oracle Call Interface, Oracle objekte za OLE i proširene verzije JDBC i JMS koje pružaju pristup Oracle specifičnim značajkama kao što su u AQ. On također nudi web bazirani pristup putem SOAP kroz AQ XML servlet.

Neke od dodatnih značajki valja istaknuti:

- Poruka se može nalaziti u redu s eksplicitnim skupom primatelja, što poništava popis pretplatnika prema redu.
- Pozivatelj može gomilati višestruke stavke u red ili iz reda operacija, što je jeftinije od stavljanja u/iz reda jedne po jedne stavke.
- Na osnovu politike zadržavanja u redu, potrošač može koristiti poruku iz reda bez brisanja iz reda. Prvi izlazni red radi kao odabrani upit koji vraća snimku poruka za izlazni red. Naknadni izlazni redovi unutar iste transakcije se obavljaju na istoj snimci bez donošenja novog odabira.
- Pošiljatelj može podijeliti složenu poruku u grupe poruka, koje potrošač može atomički obraditi.
- Pozivatelj može slušati višestruke redove, čekanjem da poruka stigne. Ako slušaju uspješno vraćene operacije, onda pozivatelj mora izdati izlazni red radi preuzimanja poruka.
- Pozivatelj može poslati u izlazni red poruku bez dohvaćanja njezinog sadržaja. To je korisno za brisanje velike poruke čiji je sadržaj nebitan. [45]

5.1.18 Amazon Simple Queue Service (SQS)

Amazon SQS je dobro poznata komercijalna usluga koja pruža pouzdanu isporuku poruka sa velikom skalabilnošću. SQS je perzistentan, pouzdan i visoko dostupan kao i mnoge druge Amazon AWS usluge [46]. U potpunosti je distribuiran i iznimno skalabilan.

Neki sustavi koriste SQS kao međuspremnik za svoj poslužitelj pri obradi masivnog broj zahtjeva. Ostale aplikacije koriste SQS u praćenju, workflow aplikacijama, analitikama velikih podataka, log obradama i mnogim drugim scenarijima distribuiranih sustava. [47][48][49]

5.1.19 Performanse i skalabilnost

Za sustave za razmjenu poruka, kvantitativna mjera *poruka u sekundi (msg/s)* ima vrlo malo smisla bez detaljne kontekstualizacije. Korišteni protokol (npr. binarni ili tekstualni), igra veliku ulogu, ali postoje i mnogi drugi faktori latencije: *perzistentne* poruke mogu biti reda veličine sporije, *amplifikacijski faktor* (npr. broj tema potrošača) može utjecati na sustav sa višestrukom ulaznom memorijskom kopijom poruke, a isto vrijedi i za *veličinu nosivosti*.

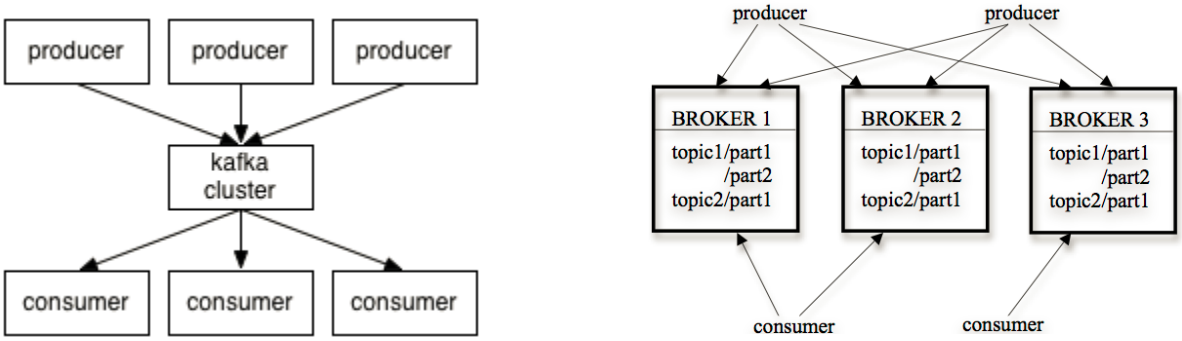
Sporedni klijenti mogu voditi velik broj *otvaranja/zatvaranja povezivanja* i čudno ponašanje potrošača što može dovesti do problema *niskog pretplatnika*, jednog od najčešćih pitanja za infrastrukturu razmjene poruka.

Usporedba predstavljena u [50], kada je nekoliko broker poruka ocijenjeno putem STOMP protokola u nekoliko komunikacijskih modela, pokazuje kako se u realnom scenariju performanse mogu razlikovati od 100.000 msg/s do 1.000 msg/s.

5.2 Apache Kafka

Apache Kafka je open-source projekt porijeklom iz LinkedIn-a, sada dio Apache fondacije. On je razvijen za *real-time* aktivnost analitičkog toka, za rješavanje potrebe za učinkovitim načinom pomicanja velikih količina podataka (npr. korisničke metrike, računalnog nadzora farme) od proizvođača do mnogih potencijalnih potrošača. Skaliranje i veličina podataka (milijarde poruka i stotine gigabajta po danu) i vremensko ograničenje čini slučaj neprikladnim za standardne brokere, kao što je uspoređeno u [51].

Inovativna ideja Kafka je da bude broker s gubicima, tako da se ne zadržavaju bilo kakve informacije o potrošačima. Kada je potrebno, potrošač mora zadržati svoje vlastito stanje (npr. informacije o posljednjim pročitanim podacima) i Kafka bazen novih podataka. To omogućava da Kafka ustraje neovisno o kopiji jedne poruke iz broja potrošača (npr. poruke se ne uklanjaju na potrošnji, nego na periodu zadržavanja ili drugoj politici), što bi dovelo do visoke propusnosti za operacije čitanja i pisanja. Kafka perzistentnost je implementirana kao distribuirani izvršni log, kao što je prikazano na slici 5.1, dizajniran kao distribuirani sustav jednostavan za skaliranje (baziran na Zookeeperu), koji omogućava automatsko uravnoteženje potrošač/proizvođač/broker.



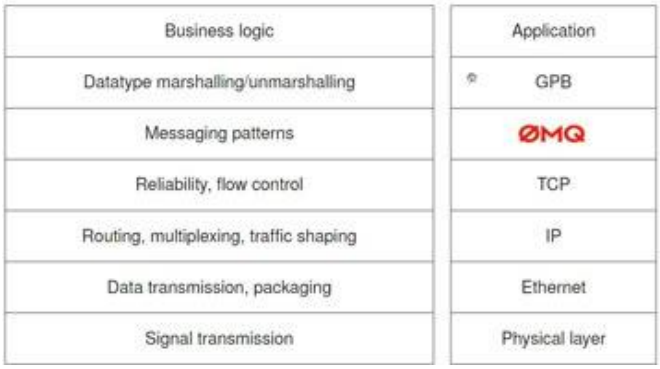
(a) Kafka cluster *(b) Kafka topic particioniranje*

Slika 5-1 Kafka arhitektura

Za razliku od standardnih broker poruka, Kafka pruža ograničene sposobnosti razmjene poruka (npr. uglavnom semantička tema, datotečni sustav kao jedinstvena trajna pohrana, strogo zajamčeno naručivanje). Iako su mnoge klijent biblioteke dostupne, što samo podržava svoj prilagođeni binarni format preko TCP. Kafka je optimalno rješenje za *podatkovno pomicanje*, često usvojeno kao *pipe* za različite sustave obrade (npr. Hadoop, Storm). [52][53]

5.3 ZeroMQ

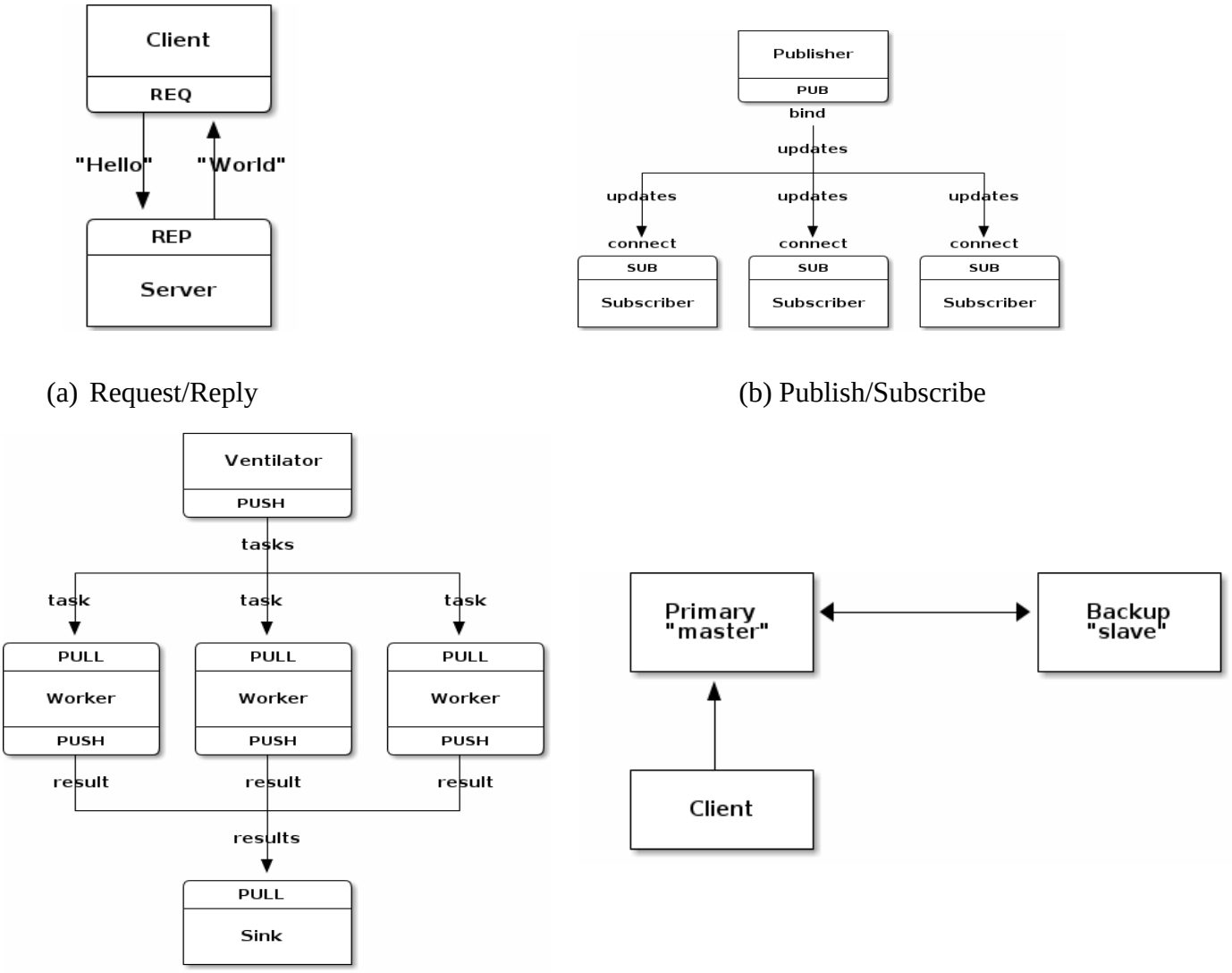
ZeroMQ (također poznat kao 0MQ ili ZMQ) [54] nije standardna broker poruka, ali je lagana biblioteka za razmjenu poruka koja pruža mogućnost razmjene poruka. Distribuirane aplikacije mogu koristiti ZeroMQ za velike propusnosti i nisku latenciju komunikacije, sa sposobnošću da implementira izravnu vezu između proizvođača i potrošača, a da nema uključene srednje entitete. Iako se to može činiti suprotno s jednom od glavnih pretpostavki razmjene poruka, ZeroMQ implementira slabo povezanu komunikaciju putem inovativnog pristupa, koji djeluje kao novi sloj na mrežnom stogu.



Slika 5-2 Mrežni stog

ZeroMQ proširuje koncept socketa sa sličnim API, ali sa poboljšanjem ugrađenih uzoraka razmjene poruka: *Request/Reply, Publish/Subscribe*, *Pipeline* i *Exclusive pair*, kao što je prikazano na slici 5.3. Za razliku od klasičnog socketa, svaki ZeroMQ socket dolazi sa unutarnjim redom što omogućava asinkronu komunikaciju. Rezultat toga je, npr. u *request/reply* slučaju koji se koristi za komunikaciju od točke do točke, ako podaci nastaju kada potrošači nisu pokrenuti. ZeroMQ biblioteka će se pobrinuti za odgodu isporuke, bez dodatnog opterećenja na strani proizvođača.

ZeroMQ omogućuje visoke performanse i nisku latenciju komunikacije, ali dolazi i sa dodatnom složenosti na aplikacijskoj razini. ZeroMQ uglavnom podržava vlastiti binarni protokol i pruža ograničene mogućnosti razmjene poruka (npr. failover, multicast podrška za 1-N topologiju). Iako se nekoliko značajki mogu lako implementirati koristeći ZMQ API (npr. potvrda), implementiranje naprednih mogućnosti razmjene poruka (npr. garantirana isporuka, perzistentnost) može zahtijevati znatan napor, što ga čini pogodnim za podatkovno-intenzivni slučaj gdje je potrebna jednostavna semantička razmjena poruka.



(a) Request/Reply (b) Publish/Subscribe

(c) Pipeline (d) Exclusive pair

Slika 5-3 Primjeri ZeroMQ soketa

5.4 Nanomsg

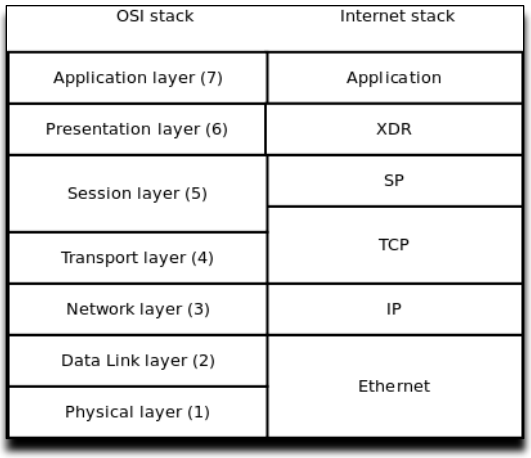
Nanomsg je socket biblioteka koja pruža nekoliko zajedničkih komunikacijskih obrazaca što čini mrežni sloj brzim, prilagodljivim i jednostavnim za korištenje. On je nastao kao reimplementacija ZeroMQ. Napisan je u C-u i funkcionira na širokom rasponu operacijskih sustava.

Komunikacijski obrasci, koji se nazivaju “skalabilni protokoli”, jesu osnovni blokovi za izgradnju distribuiranih sustava. Njihovim kombiniranjem može se stvoriti širok niz distribuiranih aplikacija. Sljedeći skalabilni protokoli su trenutno dostupni:

- PAIR – jednostavna jedan-na-jedan komunikacija
- BUS – jednostavna više-prema-više komunikacija
- REQREP – omogućuje izgradnju privremenih klaster usluga za obradu korisničkih zahtjeva
- PUBSUB – distribuira poruke prema velikom skupu zainteresiranih pretplatnika
- PIPELINE – skupljanje poruka iz više izvora i balanisiranje opterećenja između više odredišta
- SURVEY – omogućuje u jednom potezu postavljanje upita prema više aplikacija

Skalabilnost protokola je na sloju iznad transportnog sloja mrežnog stoga. Nanomsg biblioteka podržava sljedeće transportne mehanizme:

- INPROC – prijenos unutar procesa (između niti, modula i sl.)
- IPC – prijenos između procesa na istom stroju
- TCP – mreža za prijenos preko TCP



Slika 5-4 OSI i Internet stog

Biblioteka je izložena poput C API BSD socketa za ostale aplikacije. [55]

5.5 Nanomsg vs. ZeroMQ

Nedostatak ZeroMQ je da ne pruža API za nove transportne protokole, koji bitno ograničavaju transportne mehanizme (TCP, PGM, IPC i ITC-a). Nanomsg rješava taj problem pružajući utično (*engl. pluggable*) sučelje za prijenose i protokole razmjene poruka. Odnosno pruža podršku za nove prijenose (npr. tehnologija WebSockets) i nove obrasce razmjene poruka koji su izvan standardnog skupa PUB/SUB, REQ/REP, itd.

Nanomsg je također u potpunosti POSIX kompatibilan, dajući čišći API i bolju kompatibilnost. S tim nema više socketa predstavljenih kao void pokazivači i vezanih za jednostavan kontekst inicijaliziran novim socketom, a da se počinje koristiti u jednom koraku. Sa ZeroMQ, kontekst interno djeluje kao spremište mehanizma globalnog stanja i za korisnika, kao i za bazen I/O niti. Ovaj koncept je u potpunosti uklonjen iz nanomsg.

Osim POSIX kompatibilnosti, nanomsg će biti interoperabilan na API i na protokolarnim razinama, što bi omogućilo interoperabilnost sa ZeroMQ i drugim bibliotekama koje su ugrađene u ZMTP/1.0 i ZMTP/2.0.

ZeroMQ ima temeljni nedostatak u svojoj arhitekturi. Njegovi socketi nisu *thread-safe*. Samo po sebi, to nije problematično, u stvari, koristan je u nekim slučajevima. Izoliranjem svakog objekta unutar vlastite niti, potreba za semaforima i mutexima je uklonjena. Niti se ne dodiruju, a konkurentnost se postiže prosljeđivanjem poruka. Ovaj uzorak dobro radi za radne niti koje upravljaju objektima, ali se raspada kada se objektima upravlja u korisničkim nitima. Ako nit izvršava neki drugi zadatak, objekt je blokiran. Nanomsg uklanja jedan-na-jedan vezu između objekata i niti. Umjesto da se oslanja na prosljeđivanje poruka, interakcije su modelirane kao skup stanja strojeva. Slijedom toga, nanomsg socketi su *thread-safe*.

Nanomsg ima niz drugih internih optimizacija s ciljem poboljšanja učinkovitosti korištenja memorije i CPU. ZeroMQ koristi jednostavnu Trie strukturu za pohranu i PUB/SUB pretplate. Ona se izvršava do sub-10.000 pretplata, a iznad tog broja performanse počinju padati. Nanomsg koristi optimiziran prostor za *Trie* naziva *Radix tree* za pohranu pretplata. Za razliku od svog prethodnika, biblioteka nudi istinski *zero-copy* API što uveliko poboljšava performanse, omogućujući kopiranje sadržaja između memorija dva stroja potpuno zaobilazeći aktivnost CPU.

ZeroMQ provodi balansiranje opterećenja pomoću round-robin algoritma. Dok pruža jednaku distribuciju rada, on ima svoja ograničenja. Pretpostavimo da postoje dva podatkovna centra, jedan u New Yorku i jedan u Londonu, a svaka stranica sadrži instancu „foo“ usluge. U idealnom slučaju, zahtjev napravljen prema foo usluzi u New Yorku ne bi trebao biti preusmjeren prema Londonskom podatkovnom centru i obrnuto. Sa ZeroMQ-ovim round-robin balansiranjem to je nažalost moguće. Jedna od novih korisničko-naklonjenih značajki koje nanomsg nudi je prioritetno usmjeravanje na odlazni promet. Problem latencije izbjegavamo dodjeljivanjem prioriteta jedan na foo usluzi smještenoj u New Yorku za aplikacije koje su tamo postavljene, a prioritet dva dodijeljen foo usluzi u Londonu, dajući nam oporavak od pada u slučaju da foo usluga u New Yorku bude nedostupna. [56]

6 PRIMJERI KORIŠTENJA

Ovo poglavlje predstavlja nekoliko slučajeva korištenja na mjestima gdje je uspješno usvojena komunikacija bazirana na razmjeni poruka kako bi se riješio problem razmjene informacija u distribuiranom sustavu.

6.1 CERN Beam Control middleware

Beam Control odjel u CERN laboratoriju koristi razmjenu poruka za visoko pouzdanu kontrolu/nadzor/alarm aplikaciju Large Hadron Collider (LHC). Od 2005. godine klaster ActiveMQ broker u spremištu i proslijeđena konfiguracija se koriste za prikupljanje kritičnih podataka generiranih od strane sigurnosnih sustava (npr. 30 proizvođača, 2MB/s, 4,5K poruka/s) te ih prosljeđuju potrošačima (npr. alat za nadzor, dashboards). Budući da je sigurnosna podatkovna misija kritična, spremište i proslijeđene konfiguracije omogućuju da u potpunosti odvoje podatke proizvođača od potrošača te na taj način spriječe neispravno funkcioniranje klijenta što utječe na prikupljanje i arhiviranje podataka. [57] Štoviše, LHC Control framework je nedavno premješten iz CORBA na ZeroMQ kao komunikacijski sloj [58].

6.2 DAQ Online Monitoring

Razmjena poruka se isto intenzivno koristi u nekoliko alata za nadzor kod prikupljanja podataka (Data Acquisition – DAQ) sustava. Oni su odgovorni za filtriranje i prikupljanje podataka iz detektora (npr. high energy physic eksperimenata) do spremišnih objekata.

6.2.1 The ATLAS TDAQ shifter assistant project

ATLAS TDAQ *shifter assistant project* [59] oslanja se na razmjeni poruka do distribucije operativnih alarma iz privatne TDAQ mreže prema GPN do brojnih heterogenih potrošača. ActiveMQ klaster se koristi u master/slave konfiguraciji, kako bi se smanjio utjecaj na potrebnu konfiguraciju vatrozida na jednoj odlaznoj vezi.

6.2.2 The STAR Online framework

STAR Online okvir oslanja se na AMQP baziranom sustavu za fleksibilnu i labavo povezanu distribuciju detektor meta podataka korištenjem razmjene poruka kao jedinstveni

transportni sloj za procesuiranje, pohranu i nadzor. Osim toga, istraživanje je učinjeno kako bi ponovno napisali kontrolni okvir preko MQTT, profitirajući iz fleksibilnosti protokola i interoperabilnosti [60].

6.3 WLCG Messaging Service

Razmjena poruka se isto uspješno koristi u velikoj, skalabilnoj, geografsko-distribuiranoj infrastrukturi. WLCG (Worldwide LHC Computing Grid) usluga razmjene poruka je okosnica transportnog sloja koja se koristi za praćenje WLCG stranica i usluga u cijelom svijetu, sa više od 50.000 klijenata i prosječnoj stopi poruke od 100 KHz. Infrastruktura za nadzor se temelji na STOMP sa JSON nosivosti. Zahvaljujući interoperabilnosti STOMP protokola preko nekoliko broker usluga, heterogeni klasteri broker poruka (ActiveMQ, Apollo ili RabbitMQ) koriste se u situaciji gdje klijent aplikacije proizvode svima i upotrebljavaju sve [61].

6.4 FastFlow Programming Framework

FastFlow je strukturirano-paralelno programsko okruženje implementirano u C++ na vrhu sa Pthreads bibliotekom i ciljanom dijeljenom memorijom više jezgri. FastFlow pruža programerima s preddefinirane prilagodljive farme zadatke i cjevovode paralelnih uzoraka. Okruženje je u početku zamišljeno i implementirano da se vrlo učinkovito izvršavaju sitno zrnaste paralelne aplikacije [62].

U FastFlow se ZeroMQ koristi kao vanjski prijenos za *ff_dnode* konkurentni entitet. Konkretno, na vrhu ZeroMQ su izgrađeni svi oblici komunikacijskih uzoraka pomoću DEALER i ROUTER socketa koje nudi biblioteka. ROUTER socket omogućuje usmjeravanje poruka na određenu vezu, pod uvjetom da je identifikator peera poznat. DEALER socket umjesto toga može biti korišten za fair-queuing na ulazu, a za izvođenje raspoređivanja opterećenja na izlazu prema bazenu veze. Unutar FastFlow ne koristi se značajka raspoređivanja opterećenja; umjesto toga se koristi povezivanje *ff_dnode* na ROUTER socket drugog *ff_dnode*. Jednostavnost uporabe ZeroMQ je faktor izbora za implementaciju distribuiranog transportnog sloja. [63]

6.5 ALICE HLT Framework

ALICE High Level Trigger (HLT) je online rekonstrukcija, okida i komprimira podatke sustava u okviru ALICE eksperimenta na CERN-u. Za ALICE HLT u Run II potrebno je realizirati nove komponente kako bi bila omogućena brža rekonstrukcija događaja i potrebne značajke za online kalibraciju. Neka od tih komponenti je Feedback Look. Ona nam omogućuje da preko dvije komponente šaljemo i primamo podatke preko ZeroMQ okvira, koji je izvan podatkovnog prijenosnog okvira. Ove komponente se oslanjaju na asinkronoj obradi značajki, tako da ne blokiraju podatkovni tok. [63]

6.6 Okidanje događaja sa GPU-ovima kod ATLAS-a

Masivni paralelizam okidanih algoritama implementiran je na GPU-ovima kao način za povećanje računalne snage HLT farme. APE (engl. Accelerator Process Extension) je okvir razvijen za integriranje GPU-ova u Athena višeprocesnom ATLAS programskom okviru. Dijelovi sustava (ID tracking, Calorimeter TopoClustering i Muon tracking) koji iziskuju CPU intenzivni rad su paralelizirani na GPU. U koraku podatkovne pripreme *ID tracking* je uspješno integriran u Athena i ubrzan oko 21x u odnosu na CPU implementaciju.

ATLAS koristi višeprocesnu obradu događaja u HLT-u. Jedan HLT proces (HLTPU) se izvodi na jednoj virtualnoj CPU jezgri stroja. HLTPU se omata ATLAS rekonstrukcijskim softverom, Athena, i dodiruje sučeljem sa DAQ sustavom. Kako bi se upravljali i dijelili GPU između više HLTPU procesa, odabran je klijent-server pristup. Klijent strana se implementira u HLTPU, a poslužiteljska strana kao zaseban proces pod nazivom Accelerator Process Extension (APE).

Komunikacija između servera i klijenta unutar stroja, kao između strojeva je ostvarena korištenjem yampl [65] biblioteke. Yampl apstrahira različite komunikacijske tehnologije kao što su socketi, cjevovodi, dijeljena memorija i ZeroMQ [54]. Modularna struktura poslužitelja omogućuje korištenje različitih vrsta računalnih resursa kao što su CPU, GPU, Xeon Phi ili FPGA akcelerator kartice bez ikakvih promjena na strani klijenta. [66]

6.7 NASA Information And Data System (NAIADS)

NASA Information And Data System (NAIADS) je okvir prototipa za sljedeću generaciju višesenzorske fuzije i procesuiranja podataka znanosti o Zemlji. Cilj NAIADS je pružiti novi pristup za značajno poboljšanje učinkovitosti obrade i analiza više senzorskih velikih podataka znanosti o Zemlji implementirajući konceptualno novi radni tok i state-of-the-art softverske tehnologije.

NASA Information And Data System (NAIADS) je integriran sa CLARA okvirom [67] razvijenim na Jefferson Lab i razmjenom poruka na temelju ZeroMQ socket biblioteke [54]. Tehnologija je osmišljena za rad na više čvorova i višejezgrinim sustavima za masivnu obradu podataka. [68]

6.8 Subtle Noise: sonification of distributed computing operations

Funkcioniranje distribuiranih računalnih sustava zahtijeva sveobuhvatan nadzor kako bi se osigurala pouzdanost i robusnost. U većini sustava za nadzor postoje dvije komponente: jedna vizualno bogata grafovima vremenskih serija i druga kao notifikacijski sustavi za obavijesti upozorenja operaterima pod određenim unaprijed definiranim uvjetima. Ultrazvuk za monitoring poruka istražen je pomoću arhitekture koja se lako uklapa u postojeće infrastrukture na temelju zrelih open-source tehnologija, kao što su ZeroMQ, Logstash i Supercollider (a synth engine). Atributi poruka se preslikavaju na audio atribute na temelju širokog klasificiranja poruke (kontinuirane ili diskretne metrike), ali imajući audio tok suptilne prirode. Izbor sustava za razmjenu poruka (ZeroMQ) je određen potrebom za visokim performansama i skalabilnim publish-subscribe modelom. [69]

6.9 MAD – Monitoring ALICE Dataflow

Novi softverski alat je razvijen u okviru projekta za prikupljanje podataka sa poboljšanjem nadzora eksperimentalnog podatkovnog toka, od očitavanja podataka u DAQ farmi do svoje isporuke na CERN-ov glavni računalni centar. Ovaj softver, nazvan ALICE MAD (Monitoring ALICE Dataflow), koristi MonALISA okvir kao osnovni modul za prikupljanje, procesuiranje, agregiranje i distribuiranje vrijednosti za nadzor iz različitih procesa koji su pokrenuti u distribuiranoj DAQ farmi. Podaci nisu samo uzeti iz izvora

podataka do MAD-a, oni zapravo mogu biti usmjereni prema dediciranim podatkovnim kolektorima ili procesima podatkovnog izvora.

Prikupljanje podataka se izvodi preko dedicirane serverske niti. Lokalna predmemorija ograničava izvore podataka. *WSServer* se povezuje sa tri različita izvora podataka:

- *logbookDaemon*
- *Logbook REST API*
- *MonALISA Service: osigurava vrijednosti za praćenje protoka podataka. Komunikacija se izvršava putem ZeroMQ (publish-subscribe obrazac).*

Monalisa pruža mogućnost prilagođenih podatkovnih filtera koji se pretplaćuju na dani skup vrijednosti za praćenje (na temelju popisa iskazivanja koji definiraju koje bi trebale biti primljene vrijednosti), izvršavanje bilo koje željene agregacije i ponovno upisane nove vrijednosti na Monalisa usluzi. Ovi filtri su razvijeni u Javi i izvršavaju se od strane bazena niti upravljeni uslugom.

Osim toga, poseban ALICE Podatkovni Filter pretplaćuje se na vršnu razinu ALICE vrijednosti i objavljuje ih pomoću ZeroMQ [54] biblioteke (publish-subscribe obrazac). [70]

6.10 Omogućavanje GPU aceleratora s posredničkim slojem razmjene poruka

Dizajn prototip modela za omogućavanje korištenja GPU akceleratora sa posredničkim slojem razmjene poruka je široko primjenjiv. Kao dokaz koncepta primijenjeno je k-means klasteriranje na GPU-u i udaljeni pristup pomoću standardne platforme za razmjenu poruka, ZeroMQ [54]. Pristup GPU je implementiran u request-reply modu, kao i u [71], te također u publish-subscribe načinu rada, kako bi procijenili praktičnost korištenja grafičkog procesora za ubrzavanje temeljne mreže aplikacija za real-time procesuiranje obrade podataka. ZeroMQ se koristi kao posrednički sloj za razmjenu poruka, jer je lako dostupna open-source platforma koja pruža laganu implementaciju razmjene poruka s niskom end-to-end latencijom. U modelu nisu korištene jedinstvene značajke ZeroMQ, stoga se mogu očekivati slični rezultati korištenjem nekih drugih platformi za razmjene poruka. [72]

6.11 Web Liquid Streams framework

Uvođenjem peer-to-peer komunikacije između preglednika sa WebRTC, procesuiranje tokova u stvarnom vremenu može biti raspoređeno na preglednike, pored toga, i na izvršavanje u tradicionalno okruženje poslužitelja. Web Liquid Streams je okvir za izgradnju i izvođenje topologija toka procesuiranja sposobnog za prikupljanje podataka iz Web senzora i obrade kroz JavaScript operatere raspršenih širom peer-to-peer Cloud computing peerova: i) podrška za proizvoljne topologije i podatkovne tokove, ii) raspoređivanje na heterogene Web uređaje, iii) transparentno zaprimanje toka preko WebRTC, WebSocketa i ZeroMQ protokola, iv) stateful i stateless operatori. WLS brine o raspoređivanju topologije na raspoložive resurse, dok korisnici samo trebaju implementirati operatore i opisati topologije grafa pomoću JSON. Strukture topologija se mogu dinamički prilagoditi bez zaustavljanja toka kroz njih.

Komunikacijske veze koriste različite protokole (WebSocket-i, ZeroMQ, WebRTC) za prijenos elemenata podatkovnog toka. Razvijen od strane Operatora ne treba brinuti o stvarnom protokolu. Na temelju opisa Topologije, Web Liquid Streams je okruženje za izvršavanje zadatka, koji se raspoređuju na raspoložive resurse i vodi brigu o apstrahiranju složenosti i heterogenosti komunikacijskih kanala. [73]

6.12 FairMQ – Baza za podatkovni prijenosni sloj u FairRoot

Korištenjem FairMQ[74][75][76] za prijenos podataka osiguran nam je fleksibilan i učinkovit komunikacijski alat koji omogućuje mrežnim komponentama da ostanu neovisne od korištenih mrežnih tehnologija. To je jedan modul FairRoot okvira, koji omogućujuće korisniku da pokrene procesuirane komponente koje su u interakciji sa asinkronim sustavom za razmjenu poruka. FairMQ nudi apstraktno prijenosno sučelje koje se implementira preko dviju komunikacijskih biblioteka – ZeroMQ [54] i nanomsg [55], omogućujući prijenos preko mreže, međuprocesnu i međunitnu komunikaciju. Isto tako pruža korisnički pristup za nekoliko komunikacijskih obrazaca, kao što su Publish-Subscribe, Push-Pull i Request-Reply, dopuštajući fleksibilan dizajn transportnog sustava. [77]

Integracija FairMQ je zastupljena u idućim projektima:

- Integracija u Jülich Digital Readout System za ASICs (JDRS)

Readout system za prednji dio ASIC razvoja karakterizira različite vrste ASICova. Njegovi su zahtjevi:

- Brz za visoke podatkovne stupnjeve,
- Fleksibilan za različite vrste ASIC-ova
- Jednostavan za usvajanje

Jülich Digital Readout System nam omogućava analizu podataka iz ASIC prototipa sa FairRoot i kreiranje podatkovnih staza sa FairMQ za spremanje i nadzor podatkovnog toka. [78]

- Korištenje GPU kao procesora visokih performansi za FairRoot [78][79][80]

U okviru P̄ANDA Experiment [81][82][83] pri FAIR-u razvijeni su algoritmi Triplet Finder i Circle Hough za rekonstrukciju staza nabijenih čestica. Algoritmi su optimizirani korištenjem grafičkih procesnih jedinica opće namjene (General-Purpose Graphic Processing Units – GPU). Konceptualni sustav za odašiljanje podataka prema algoritmima za praćenje koriste Message Queues (FairMQ). [80]

6.13 The new CERN tape software – getting ready for total performance

CASTOR (CERN Advanced System Storage) se koristi za pohranu kopija svih podataka fizike prikupljenih iz prošlih i sadašnjih CERN eksperimenata. CASTOR je hijerarhijski sustav upravljanja pohrane podataka koji ima prednji dio diskovno baziran i stražnji dio baziran na trakama. Softver odgovoran za kontrolu stražnjeg dijela trake je redizajniran i obnovljen tijekom 2014. godine te je postavljen u produkciju početkom 2015. godine.

U novoj verziji CERN tape software primijenjena su dva protokola za tape poslužitelje: protokol za komunikaciju od procesa sesije na glavni proces za statističko propagiranje i protokol na novom *demon* za kontroliranje Oracle tape biblioteka. U oba slučaja se koriste ZeroMQ [54] za prijenos i Google protocol buffers [84] za serijalizacijsku nosivost. Za ostatak komunikacije, naslijeđene C biblioteke su reimplementirane u C++ sa postojećim CASTOR komponentama. Ova reimplementacija standardizira prethodna

heterogena rješenja za rukovanje pogreškom i propagacijom što novije dijelove CASTOR-a čine jednostavnim za ponovno korištenje. [85]

6.14 Archiving tools for EOS

Arhiviranje podataka na traku je kritična operacija za bilo koji sustav pohrane podataka, a posebno za EOS sustav u CERN-u koji posjeduje produkcijske podatke za sve glavne LHC eksperimente. Svaka kolaboracija pripada jednom dijelu koji se može koristiti u bilo kojem trenutku, dakle, potreban je mehanizam za arhiviranje „starih“ podataka, tako da prostor za pohranu razotkriva online analizu operacija. Alat za arhiviranje koji je predložen za EOS ima za cilj pružiti snažno klijentsko sučelje za premještanje podataka između EOS i CASTOR (tape backed storage system), dok je provođenje najbolja praksa kada je u pitanju integritet podataka i verifikacija.

Svi prijenosi podataka obavljaju se pomoću mehanizma kopiranja treće strane (third-party copy) što osigurava point-to-point komunikacija između izvora i odredišta, čime se osigurava maksimalni ukupni protok. Korištenje ZMQ paradigme prosljeđivanja poruka i procesno bazirani pristup omogućava postizanje maksimalne iskoristivosti resursa i arhitekture slabog stanja, koja se lako može podešavati za vrijeme rada. Modularni dizajn i implementacija napravljena na jeziku visoke razine kao što je Python, omogućava nam jednostavno proširivanje kodne baze za rješavanje novih zahtjeva kao što nude potpune i inkrementalne backup mogućnosti.

Klijent koristi EOS sučelje naredbenog retka za slanje zahtjeva prema MGM (EOS meta podatak usluge). MGM odlučuje na temelju identiteta klijenta, da se zahtjev može odobriti i proslijediti prema arhivskom *demon* koristeći prilagođeni komunikacijski protokol. Arhivski *demon* zatim registrira zahtjev za čekanje u red i ovisno o politici konfiguracije odlučuje kada će početi sa izvršavanjem prijenosa.

Komunikacija između MGM i arhivskog *demon* je ostvarena pomoću ZMQ [54]. U trenutnoj implementaciji oba *demona* izvode se na istom fizičkom stroju, dakle, komunikacija se obavlja pomoću IPC (među-procesna komunikacija) mehanizma. Korištenjem ZMQ fleksibilnosti može se lako u potpunosti promijeniti TCP stog bez većih modifikacija osnovnog koda. Još jedna značajka ZMQ jest da se u potpunosti iskorištavaju jednostavnost

usmjeravanja poruka putem JSON formata. Izgradnjom ovog modela, osigurala bi se laka proširivost cijelog sustava sa različitim vrstama poruka, kao i poboljšanje trenutne poruke sa dodatnim informacijama. Protokol je zasnovan na request-reply modelu, gdje arhivski *demon* ili potvrđuje primitak određenog zahtjeva ili zapravo pruža informacije natrag MGM-u kao što je to slučaj za unos statusa u postupku transfera. [86]

6.15 Big Stream Processing Systems

Flink Streaming je proširenje jezgre Flink API za visoke propusnosti i nisku latenciju obrade podatkovnog toka. Sustav se može povezati sa procesima podatkovnih tokova iz različitih izvora podataka (npr. Flume, ZeroMQ), gdje se podatkovni tokovi mogu transformirati i modificirani pomoću funkcija visoke razine sličnim onima koji se nalaze u API skupnoj obradi. [87]

7 AKTUALNA PODRUČJA ISTAŽIVANJA

7.1 Prošireni ZeroMQ i/ili NanoMSG sa podrškom za SCIF

Kao novi prijenosni sloj na stroju (Xeon Phi) predstavljao bi prošireni ZeroMQ sa SCIF-om. Postupak proširivanja ZeroMQ-a sa SCIF nije nitko do sada pokušao realizirati jer:

- Virtual Machine Communication Interface (VMCI) je podržan tek početkom prošle godine
- NACK-Oriented Reliable Multicast (NORM)[88] je podržana prije tri godine

SCIF pruža dosta veliku brzinu prijenosa podataka u odnosu ZeroMQ i NanoMSG [89].

Za proširivanje ZeroMQ potrebno je implementirati 4 modula:

1. Address (myaddr:// address resolution)
2. Connecter
3. Listener
4. Engine (send/recv)

Postupak proširenja NanoMSG-a sa SCIF je:

- dosta težak postupak
- još nije primijenjen u produkciji
- razvoj je stao (ne postoji zajednica za NanoMSG)
- dodavanje značajki na nestabilnom baziranom kodu predstavlja problem [90].

7.2 Novi nanomsg transport baziran na libfabric

U suradnji s CISCO-om, user-space NICs (usNIC) je korišten za testiranja novog transportnog sloja. usNIC procesuiranje nam omogućuje zaobilaženje Linux Kernela te komuniciranje izravno sa User Space. U novom transportnom sloju se zahtijeva novo sučelje sa ALICE software-om bez njegove modifikacije. Odabran je nanoMSG iz FairMQ kao biblioteka sa kojom će se proširiti FairMQ kao novi transportni sloj. Iz razlog što ima čišći i jednostavniji API za proširivanje. Kod proširivanja glavni fokus je na prijenosu. Neki od nedostataka prije realizacije su: jezgra ZeroMQ i NanoMSG su osmišljeni oko UNIX socketa

i usNIC API, koji je zatvoren za MPI i RDMA. Iz tih razloga se koristi libfabric koja se nalazi negdje između dva navedena nedostatka. OpenFabrics Interfaces je okvir fokusiran na izvođenju fabričkih komunikacijskih servisa prema aplikacijama. OFI je najbolje opisan kao skup biblioteka i aplikacija koji se koriste za izvoz fabričkih usluga. Ključne komponente OFI su: aplikacijsko sučelje, pružatelj biblioteka, kernel usluge, daemoni i testne aplikacije.

Tipovi aktivnih krajnjih točaka:

- FI_DGRAM – Unreliable Datagrams (npr. UDP)
- FI_RDM – Reliable Datagrams (npr. RDMA)
- FI_MSG – Connection-aware message passing (npr. TCP)

Visoka razina API je zatvorena za socket API. Davatelj usluga implementira fragmentaciju i kontrolu toka. Koriste se jednostavne funkcije: *fi_send(), fi_recv()* te se koristi RDMA iznad svih scena. Memorijska registracija se bazira na OFI prijenosu koji ima ponovno korištene memorijske „grupe“. Ako pokazivač koji šalje pripada registriranoj regiji, MR opis iz regije će biti korišten. Inače će starija grupa ili banka biti deregistrirana i popunjena sa novim informacijama pokazivača.

OFI transport se označava: ofi://ip:port[@fabric[:provider]] te je beševno prenosiv i kod drugih davatelja. Odnosno isti kod radi sa infiniband, omnipath, usnic, itd..

Za Zero-Copy u nanomsg bufferi su organizirani u chunkove. Svaki chunk ima referentni brojač koji se povećava umjesto kopiranja. Kada se podaci iz raw pokazivača pošalju, oni se kopiraju u novi chunk. nn_allocmsg funkcija trebala bi biti pozvana za alokaciju novog chunka unaprijed. Dok u trenutnoj veziji nije moguće alocirati chunk iz postojećeg podatka.

Dodaci nanoMSG:

- Ulančanost chunk destruktora: Da bi se omogućio prijenos prati se kada je blok slobodan kako bi poništili registraciju memorije.
- Stvaranje *nn_msg* od korisničkog pokazivača: Umjesto da pustimo nanomsg alokaciju tijela poruke je funkcija *nn_allocmsg_ptr* koja omogućuje stvaranje poruka zero-copy iz postojećih podataka

Postignuta je brzina 42 Gbit/s. NanoMSg OFI prijenos omogućava sučelje za sockete za visoke performance RDMA fabrike, kao Infiniband, Omni-Path, usNIC, itd. [91]

7.3 Unificirana kontrola u P2P komunikacijskom posredničkom sloju

Složene distribuirane usluge aplikacija zahtijevaju proširivanje obrasca za razmjenu poruka. Tradicionalnim sinkronim Request-Response i Publish-Subscribe potrebno je proširenje koje će pružiti asinkroni pristup sa trenutačnom P2P (Peer-to-Peer) komunikacijskom bibliotekom, kao što su ZeroMQ i NanoMsg. Opisom obrasca poruka u osnovi je ograničena na jednu vrstu akcije za grupu povezanih peerova. Poslovna logika često zahtijeva promjene u shemama balansiranja opterećenja za najučinkovitije zadatke distribucije. Predložena je lagana biblioteka i skup modula koji su postavljeni na vrhu zajedničke P2P komunikacije. Ona osigurava ogromne obrasce proširivanja s podrškom za sudjelovanje poslovne logike u balansiranju opterećenja. Odnosno pruža proširivost na razini komponenata pomoću interakcija i kontrole distribucije opterećenja preko poslovne logike, čime se dobiva potpuna shema povezanosti između čvorova sustava. [92][93]

8 ZAKLJUČAK

Razmjena poruka je u osnovi pragmatična reakcija na problem distribuiranih sustava [26]. Razmjena poruka, kao što je prikazano u poglavlju 3, dozvoljava slabo povezanu komunikaciju djelujući kao srednji sloj između proizvođača i potrošača. S tim donosi mnoge prednosti u distribuiranim aplikacijama: fleksibilnost i skalabilnost sa obuhvaćenom aplikacijom i infrastrukturnom složenosti.

Tehnologije sustava za razmjenu poruka još se razvijaju, kao što je prikazano u poglavlju 4. Sa AMQP standardizacijskim naporom ide se u dobrom smjeru, ali još uvijek s djelomičnim usvajanjem. Brokeri poruka su čvrsta i pouzdana tehnologija korištena kao transportni sloj za gradivne blokove u mnogim projektima i uslugama, kako u zajednici fizičara tako i izvan nje. U posljednjih nekoliko godina, nova generacija sustava promiče razmjenu poruka za niske latencije, visoku propusnost i podatkovno-intezivnu komunikaciju, kao što je prikazano u poglavlju 6. Sužavanjem korištenih slučajeva i oslabljivanjem pretpostavki pomiču se granice aplikacija za razmjenu poruka prema novim domenama.

LITERATURA

[1] AMQP (Advanced Message Queuing Protocol) http://www.amqp.org

[2] AMQP Working Group. Advanced Message Queuing Protocol (AMQP) version 0–10 Specification. Specification; 2009, http://www.amqp.org.

[3] Snyder B, Bosanac D, Davies R. ActiveMQ in action. Manning Publications; 2009.

[4] STOMP (Simple Text Oriented Messaging Protocol) http://stomp.github.io

[5] Strachan J. Stomp Protocol Specification, Version 1.0. Specification, FuseSource; 2005, /http://stomp.codehaus.org/Protocol.

[6] MQTT (MQ Telemetry Transport) http://mqtt.org

[7] Apache Software Foundation. OpenWire Version 2 Specification. Specification, Apache ActiveMQ; 2009, http://activemq.apache.org/openwire-version-2-specification.html

[8] Fielding RT. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine; 2000.

[9] Saint-Andre P (2011) Extensible messaging and presence protocol (XMPP):Address Format. RFC 6122 (Proposed Standard). http://www.ietf.org/rfc/rfc6122.txt

[10] Saint-Andre P (2011) Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard). http://www.ietf.org/rfc/rfc6120.txt

[11] Saint-Andre P (2011) Extensible messaging and presence protocol (XMPP): Instant messaging and presence. RFC 6121 (Proposed Standard). http://www.ietf.org/rfc/rfc6121.txt

[12] XMPP: IoT Systems (2013). http://wiki.xmpp.org/web/Tech_pages/IoT_systems. Accessed 2014–07-24

[13] Saint-Andre P, Šimerda P (2008) XEP-0231: Bits of Binary. http://xmpp.org/extensions/xep-0231.html. Accessed 25 July 2014

[14] Adams D (2011) XEP-0030: Service Discovery. http://xmpp.org/extensions/xep-0009.html. Accessed 23 July 2014

[15] Hildebrand J, Millard P, Eatmon R, Saint-Andre P (2008) XEP-0009: Jabber-RPC. http://xmpp.org/extensions/xep-0030.html. Accessed 23 July 2014

[16] Millard, P., Saint-Andre, P., Meijer, R.: XEP-0060: Publish-Subscribe (2010). http://xmpp.org/extensions/xep-0060.html. Accessed 23 July 2014

[17] Hildebrand J, Saint-Andre P (2004) XEP-0033: Extended stanza addressing. http://xmpp.org/extensions/xep-0033.html. Accessed 23 July 2014

[18] Saint-Andre P, SmithK(2010) XEP-0163: PersonalEventing Protocol. http://xmpp.org/extensions/xep-0163.html. Accessed 23 July 2014

[19] Waher P (2014) XEP-0337: Event Logging over XMPP. http://xmpp.org/extensions/xep-0337.html. Accessed 2014–07-23

[20] Miller M, Saint-Andre P (2005) XEP-0079: Advanced message processing. http://xmpp.org/extensions/xep-0079.html. Accessed 23 July 2014

[21] Ramsdell B, Turner S (2010) Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard). http://www.ietf.org/rfc/rfc5751.txt

[22] Forno F, Saint-Andre P (2005) XEP-0072: SOAP Over XMPP. http://xmpp.org/extensions/xep-0072.html. Accessed 23 July 2014

[23] Conzon D, Bolognesi T, Brizzi P, Lotito A, Tomasi R, Spirito M (2012) The virtus middleware: an xmpp based architecture for secure iot communications. In: 21st international conference on computer communications and networks, ICCCN'12, pp 1–6

[24] ignite realtime: Openfire (2014). http://www.igniterealtime.org/projects/openfire/. Accessed 23 July 2014

[25] Kaazing: WebSocket Gateway – XMPP (2014). http://kaazing.com/products/editions/kaazing-websocket-gateway-xmpp/. Accessed 23 July 2014

[26] G Hohpe and B Woolf 2003 Enterprise Integration Patterns Addison-Wesley Professional

[27] A. Videla, J.W. Williams, RabbitMQ in Action: Distributed Messaging for Everyone, MEAP Edition Manning Early Access Program, 2011.

[28] AMQP Advanced Message Queuing Protocol, Protocol Specification, Version 0-9-1, http://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf 13 November 2008.

[29] M. Albano, L.L. Ferreira, L.M. Pinho, A.R. Alkhawaja, "Message-oriented middleware for smart grids", Computer Standards & Interfaces (2015), vol.38, pp. 133-143, Elsevier, DOI 10.1016/j.csi.2014.08.002

[30] Saida Davies; Laura Cowen; Cerys Giddings and Hannah Parker (2005.) WebSphere Message Broker Basics, International Business Machines Corporation 2005.

[31] Heroku, 2016. https://www.heroku.com/home

[32] Xively, 2014. https://xively.com/.

[33] Hivemq, 2016. http://www.hivemq.com/docs/hivemq/latest/

[34] Lotus Expeditor micro broker, 2016. http://mqtt.org/infocenter/

[35] Moquette, 2016. https://projects.eclipse.org/proposals/moquette-mqtt

[36] Mosquitto, 2016. https://mosquitto.org/

[37] The Internet of Things and the cool IBM Really Small Message Broker https://www.ibm.com/developerworks/community/blogs/iic-san-mateo/entry/the_internet_of_things_and_the_cool_ibm_really_small_message_broker?lang=en, pristupljeno 17/8/2016

[38] HornetQ User Manual, https://docs.jboss.org/hornetq/2.2.14.Final/user-manual/en/html_single/#messaging-concepts, pristupljeno 17/8/2016

[39] Apache Foundation, 2012. ActiveMQ Apollo. http://activemq.apache.org/apollo/.

[40] Apache Foundation, 2011. Apache QPid. http://qpid.apache.org/.

[41] Red Hat, 2011. JBoss Messaging, https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform_Common_Criteria_Certification/5/html-single/JBoss_Messaging_User_Guide/index.html#about

[42] „Sun Java System Message Queue 4.3 Technical Overview“, (2010.), Pristupljeno: 31.08.2016. g., URL: http://docs.oracle.com/cd/E19340-01/820-6424/aerap/index.html, Oracle Corporation and/or its affiliates

[43] Maheshwari, P. and Pang, M. (2005), Benchmarking message-oriented middleware: TIB/RV versus SonicMQ. Concurrency Computat.: Pract. Exper., 17: 1507–1526. doi:10.1002/cpe.881

[44] Message Queuing (MSMQ), (2016.) Pristupljeno 31.08.2016. g. URL: https://msdn.microsoft.com/en-us/library/ms711472(v=vs.85).aspx

[45] Philip A. Bernstein; Eric Newcomer (2009), „Principles of Transaction Processing (Second Edition)“, A volume in The Morgan Kaufmann Series in Data Management Systems, 2009, Pages 99–119

[46] Amazon Elastic Compute Cloud (Amazon EC2), Amazon Web Services, [online] 2013, http://aws.amazon.com/ec2/

[47] W. Vogels.“ Improving the Cloud – More Efficient Queuing with SQS” [online] 2012, http://www.allthingsdistributed.com/2012/11/efficient-queueing-sqs.html

[48] R. Ramesh, L. Hu,andK.Schwan.“Project Hoover:auto-scaling streaming mapreduce applications”. Proceedings of (MBDS '12). ACM, USA, 7-12. 2012

[49] C. Dumitrescu, I. Raicu, I. Foster. "The Design, Usage, and Performance of GRUBER: A Grid uSLA-based Brokering Infrastructure", International Journal of Grid Computing, 2007

[50] Chirino H STOMP benchmark http://hiramchirino.com/stomp-benchmark

[51] Kreps J, Narkhede N and Rao J Kafka: A Distributed Messaging System for Log Processing. NetDBWorkshop (Athens, GREECE)

[52] Manuel Díaz, Cristian Martín, Bartolomé Rubio (2016.), „State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing“, Journal of Network and Computer Applications, Volume 67, May 2016, Pages 99–117

[53] Garg, N.. Apache Kafka 2013; URL:http://dl.acm.org/citation.cfm?id=2588385

[54] Hintjens P. (2014) ZeroMQ: The Guide [Online]: http://zeromq.org

[55] Sustrik M. (2014) nanomsg [Online]; http://nanomsg.org/

[56] „A Look at Nanomsg and Scalability Protocols (Why ZeroMQ Shouldn’t Be Your First Choice)“, [Online]: http://bravenewgeek.com/a-look-at-nanomsg-and-scalability-protocols/, Pristupljeno: 31.08.2016.

[57] Ehm F Running a Reliable Messaging Infrastructure for CERN’s Control System. Proceedings of ICALEPCS2011 (Grenoble, FRANCE)

[58] Dworak A, Ehm F, Sliwinski W and Sobczak M 2011 Middleware Trends and Market Leaders 2011. Proceedings of ICALEPCS2011 (Grenoble, FRANCE)

[59] Kazarov A, Miotto G L and Magnoni L 2012 The AAL project: automated monitoring and intelligent analysis for the ATLAS data taking infrastructure. Journal of Physics: Conference Series, Volume 368

[60] Arkhipkin D, Lauret J and Betts W 2011 A message-queuing framework for STARs online monitoring and metadata collection. Journal of Physics: Conference Series, Volume 331

[61] Cons L and Paladin M 2011 The WLCG Messaging Service and its Future. Journal of Physics: Conference Series, Volume 396

[62] Aldinucci, M., Danelutto, M., Kilpatrick, P., Torquati, M.: Fastflow: high-level and efficient streaming on multi-core. In: Programming Multi-core and Many-core Computing Systems. Parallel and Distributed Computing. Wiley (2012)

[63] Aldinucci, M.; Campa, S.; Danelutto, M.; Kilpatrick, P. and Torquati, M. (2012) „Targeting Distributed Systems in FastFlow“, I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 47–56, 2013., Springer-Verlag Berlin Heidelberg 2013

[64] David Rohr, Sergey Gorbunov, Mikolaj Krzewicki, Timo Breitner, Matthias Kretz, Volker Lindenstruth for the ALICE Collaboration (2015) „Fast TPC Online Tracking on GPUs and Asynchronous Data Processing in the ALICE HLT to facilitate Online Calibration“

[65] Yampl Library URL https://github.com/vitillo/yampl

[66] Kama, S.; Soares, J. Augusto; Baines, J.; Bauce, M.; Bold, T.; Conde Muino, P.; Emeliyanov, D.; Goncalo, R.; Messina, A.; Negrini, M.; Rinaldi, L.; Sidoti, A.; Tavares Delgado, A.; Tupputi, S.; Vaz Gil Lopes, L., „Triggering events with GPUs at ATLAS“, Journal of Physics: Conference Series, Volume 664, Issue 9, article id. 092014 (2015).

[67] Gyurjyan, V., D. Abbott, J. Carbonneau, G. Gilfoyle, D. Heddle, G. Heyes, S. Paul, C. Timmer, D. Weygand, E. Wolin, “CLARA: A Contemporary Approach to Physics Data Processing,” J. of Physics: International Conference on Computing in High Energy and Nuclear Physics, Conference Series 331, doi:10.1088/1742-6596/331/3/032013, 2011.

[68] C. Lukashin, A. Bartle, E. Callaway, V. Gyurjyan, S. Mancilla, R. Oyarzun, and A. Vakhnin, „Earth Science Data Fusion with Event Building Approach“, 2015 IEEE International Conference on Big Data (Big Data)

[69] P. A. Love Subtlenoise: sonification of distributed computing operations, J. Phys.: Conf. Ser. 664 062034, 2015.

[70] V. Chibante Barroso, F. Costa, C. Grigoras and A. Wegrzynek for the ALICE Collaboration, „MAD – Monitoring ALICE Dataflow“, J. Phys.: Conf. Ser. 664 082003, 2015.

[71] King, G.H., Cai, Z.Y., Lu, Y.Y., Wu, J.J., Shih, H.P., Chang, C.R.: A High-Performance Multi-user Service System for Financial Analytics Based on Web Service and GPU Computation. In: International Symposium on Parallel and Distributed Processing with Applications (ISPA 2010), pp. 327–333 (2010)

[72] Duran, R. E.; Zhang, L. and Hayhurst, T. (2011), „Enabling GPU Acceleration with Messaging Middleware“, Informatics Engineering and Information Science Volume 253 of the series Communications in Computer and Information Science pp 410-423

[73] Babazadeh, M.; Gallidabino, A. and Pautasso, C. (2015), „Decentralized Stream Processing Over Web-Enabled Devices“, Service Oriented and Cloud Computing Volume 9306 of the series Lecture Notes in Computer Science pp 3-18

[74] M. Al-Turany et al. The FairRoot framework J. Phys.: Conf. Ser. 396 022001, 2012.

[75] M. Al-Turany FairRoot: http://fairroot.gsi.de.

[76] M. Al-Turany, D. Klein, A. Manafov, A. Rybalchenko, F. Uhlig: Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data, 2014, J. Phys.: Conf. Ser. 513 022001.

[77] M. Al-Turany, D. Klein, A. Manafov, A. Rybalchenko and F. Uhlig (2014.) Extending the FairRoot framework to allow for simulation and reconstruction of free streaming dana, Journal of Physics: Conference Series 513 (2014) 022001

[78] S. Esch, „FairMQ with FPGAs and GPUs“, https://indico.gsi.de/getFile.py/access?contribId=12&resId=0&materialId=slides&confId=2943

[79] L. Bianchi, Forschungszentrum Jülich(2014.) „Combination Of Message Queues And GPUs For The Event Building of the PANDA Experiment“, International Conference on Science and Technology for FAIR in Europe 2014, Worms, Germany, 13 Oct 2014 – 17 Oct 2014

[80] L. Bianchi, A. Herten, J. Ritman, T. Stockmanns, A. Adinetz, J. Kraus and D. Pleiter „Online Tracking Algorithms on GPUs for the P̄ANDA Experiment at FAIR“, Journal of Physics: Conference Series, Volume 664, Online computing

[81] Lutz M et al. (PANDA) 2009 (Preprint 0903.3905)

[82] Erni W et al. (PANDA) 2012 (Preprint 1207.6581)

[83] Erni W et al. (PANDA) 2013 Eur.Phys.J. A49 25 (Preprint 1205.5441)

[84] Protocol buffers: Google’s mechanism for serializing structured data https://developers.google.com/protocolbuffers/

[85] Cano, E.; Murray, S.; Kruse, D. F.; Kotlyar V. and Côme, D. (2015), „The new CERN tape software – getting ready for total performance“, J. Phys.: Conf. Ser. 664 042007

[86] Sindrilaru, E. A.; Peters, A. J. and Duellmann, D. (2015), „Archiving tools for EOS“, Journal of Physics: Conference Series 664 (2015) 042049

[87] Bajaber, F.; Elshawi, R; Batarfi, O.; Altalhi, A.; Barnawi, A. and Sakr, S.(2016) „Big Data 2.0 Processing Systems: Taxonomy and Open Challenges“, J Grid Computing, doi 10.1007/s10723-016-9371-1, Springer Science+Business Media Dordrecht 2016

[88] NORM Transport Notes, http://zeromq.org/topics:norm-protocol-transport

[89] A. Santogidis (2015) „Optimizing the transport layer of the ALFA framework for the Intel® Xeon Phi coprocessor“, 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015), 13–17 April 2015, Okinawa, Japan

[90] ALICE CWG13 Meeting (2016.) „Extending ZeroMQ and/or NanoMSG with support for SCIF“, https://indico.cern.ch/event/502892/contributions/2021784/attachments/1234777/1812281/CWG13_February.pdf

[91] Charalampidis, I., „New libfabric based transport for nanomsg“, Online:01.04.2016. https://indico.cern.ch/event/508147/contributions/1187613/attachments/1251165/1844875/Charalampidis-AliceOfflineWeek-v4.pdf

[92] Iakushkin, O. and Grishkin, V. (2015), „Unification of control in P2P communication middleware: Towards complex messaging patterns“, AIP Conference Proceedings 1648, 040004 (2015); doi: 10.1063/1.4912360

[93] Iakushkin, O.; Sedova, O. and Valery, G. (2016), „Application Control and Horizontal Scaling in Modern Cloud Middleware“, Springer-Verlag Berlin Heidelberg, M.L. Gavrilova and C.J. Kenneth Tan (Eds.): Trans. on Comput. Sci. XXVII, LNCS 9570, pp. 81–96, 2016.; doi: 10.1007/978-3-662-50412-3 6

PRILOG – Kazalo slika