



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Мића Ћетковић

Реализација скалабилног послужиоца за Internet of Things екосистеме заснованог на TR-069 протоколу

МАСТЕР РАД

Нови Сад, 2014



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	Монографска документација
Тип записа, ТЗ :	Текстуални штампани материјал
Врста рада, ВР :	Дипломски – мастер рад
Аутор, АУ :	Мића Ћетковић
Ментор, МН :	доц. др Милан Бјелица
Наслов рада, НР :	Реализација скалабилног послужеоца за Internet of Things екосистеме заснованог на TR-069 протоколу
Језик публикације, ЈП :	Српски / латиница
Језик извода, ЈИ :	Српски
Земља публикавања, ЗП :	Република Србија
Уже географско подручје, УГП :	Војводина
Година, ГО :	2014
Издавач, ИЗ :	Ауторски репринт
Место и адреса, МА :	Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, ФО : (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/71/52/8/29/0/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	Кластер, Internet of Things, оптимизација, послужеоц
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	У раду је представљена реализација послужеоца који обезбеђује скалабилан и ефикасан начин за надзор и конфигурацију уређаја потрошачке електронике у Internet of Things екосистемима. Имплементација послужеоца је заснована на TR-069 протоколу за даљинско управљање и надгледање уређаја. Послужилац обезбеђује апликативне спреге којима се дистрибуирају функционалности послужеоца. Скалабилност послужеоца је постигнута оптимизацијом постојећег решења, али и креирањем кластера система и интегрисањем послужеоца у креирани кластер.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник: др Иштван Пап
	Члан: др Растислав Струхарик
	Члан, ментор: др Милан Бјелица
	Потпис ментора



UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :	Monographic publication		
Type of record, TR :	Textual printed material		
Contents code, CC :	Master Thesis		
Author, AU :	Mića Četković		
Mentor, MN :	Milan Bjelica, PhD		
Title, TI :	Implementation of a scalable server for Internet of Things ecosystem based on TR-069 protocol		
Language of text, LT :	Serbian		
Language of abstract, LA :	Serbian		
Country of publication, CP :	Republic of Serbia		
Locality of publication, LP :	Vojvodina		
Publication year, PY :	2014		
Publisher, PB :	Author's reprint		
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6		
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/71/52/8/29/0/0		
Scientific field, SF :	Electrical Engineering		
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems		
Subject/Key words, S/KW :	Cluster, Internet of Things, optimization, server		
UC			
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia		
Note, N :			
Abstract, AB :	This paper describes the implementation of the server that provides scalable and efficient way for configuration and control of consumer electronics devices in Internet of Things ecosystems. The implementation is based on the TR-069 communication protocol for remote control and monitoring devices. Server provides APIs for distributing functionalities of the server. Scalability of the server is achieved by optimizing the server, creating a cluster system and integrating the server into the system.		
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	Ištván Pap, PhD	Menthor's sign
	Member:	Rastislav Struharik, PhD	
	Member, Mentor:	Milan Bjelica, PhD	

Zahvalnost

Zahvaljujem se svom mentoru, doc. dr. Milanu Bjelici na stručnoj pomoći, savetima i podršci tokom izrade rada.

Posebno se zahvaljujem rukovodstvu Instituta RT-RK na ukazanoj prilici da se bolje upoznam sa načinom rada u inženjerskom okruženju i budem uključen u proces razvoja novih programskih rešenja.

Zahvaljujem se Ivani na moralnoj podršci i strpljenju tokom izrade ovog rada.
Na kraju se zahvaljujem svim svojim kolegama koji su na bilo koji način doprineli izradi ovog završnog rada.

SADRŽAJ

1. Uvod.....	1
2. Teorijske osnove	4
2.1 Internet of Things koncept	4
2.2 TR-069 protokol.....	7
2.3 Klasteri	11
2.3.1 Arhitektura klastera	14
2.3.2 Primeri klastera	16
2.3.3 Primena klastera u IoT ekosistemu	18
2.4 Cloud tehnologije	18
2.4.1 IoT i Cloud.....	20
3. Analiza problema	21
3.1 Analiza tehničkih zahteva	22
3.2 Analiza problema optimizacije poslužioca	23
3.3 Analiza problema kreiranja klastera.....	25
4. Koncept rešenja.....	27
4.1 Implementacija TR-069 protokola	27
4.2 Optimizacija implementiranog poslužioca.....	30
4.2.1 Optimizacija upotrebe IMDG struktura podataka	30
4.2.2 Optimizacija upotrebe baze podataka	33
4.3 Implementacija klastera.....	34
4.3.1 Klasterovanje eksternih resursa	34
4.3.1.1 Klasterovanje Infinispan-a.....	35
4.3.1.2 Klasterovanje JBoss aplikativnog poslužioca	37

4.3.1.3	Klasterovanje NGINX raspoređivača.....	39
4.3.2	Prilagođenje implementiranog poslužioca za rad u klasteru	39
4.3.2.1	Upotreba sistema datoteka.....	39
4.3.2.2	Upotreba EJB vremenskih okidača	41
4.3.2.3	Upotreba statičkih promenljivih koje nemaju konačnu vrednost	42
5.	Programsko rešenje.....	43
5.1	Klasterovanje eksternih resursa.....	43
5.1.1	Konfigurisanje Infinispan-a	43
5.1.2	Konfigurisanje JBoss aplikativnog poslužioca.....	47
5.1.3	Konfigurisanje Nginx raspoređivača opterećenja.....	48
5.2	Programsko rešenje implementiranog poslužioca.....	50
5.2.1	Biznis logika	50
5.2.2	Prilagođenje implementiranog poslužioca.....	52
5.2.2.1	Modul za rukovanje sistemom datoteka	52
5.2.3	Modul za distribuirane vremenske okidače	54
5.2.4	TR-069 komunikacioni sloj	56
6.	Ispitivanje i rezultati	58
6.1	Ispitivanje tehničkih zahteva.....	59
6.2	Ispitivanje optimizacija postojećeg rešenja.....	61
6.3	Ispitivanje skalabilnost klastera	64
6.4	Ukupni rezultati.....	65
7.	Zaključak	68
8.	Literatura.....	69

SPISAK SLIKA

Slika 1. Mobilni telefon kao posrednik između „stvari“ i Interneta [1].....	6
Slika 2. Arhitektura TR-069 sistema na najvišem nivou	9
Slika 3. Konfiguracija klaster sistema bez deljenja diska [18]	12
Slika 4. Konfiguracija klaster sistema sa deljenjem diska [18]	14
Slika 5. Arhitektura klaster sistema	14
Slika 6. Arhitektura najvišeg nivoa implementiranog IoT ekosistema	21
Slika 7. Komponente implementiranog klastera	25
Slika 8. Arhitektura najvišeg nivoa implementiranog ACS-a.....	28
Slika 9. Arhitektura jezgra ACS-a	29
Slika 10. Neoptimalni dizajn upotrebe IMDG struktura 1 - Dodaje se element 2 -Dodaju se elementi 3 – Dodaje se element 4 - Dodaje se element 5 - Čita se i uklanja element 6 - Promena elementa 7 - Dodaje se element 8 - Uklanja se 9 - Uklanja se element p1 – Čitaju se elementi p2 – Uklanjaju se elementi p3 – Dodaje se elementi p4 – Menjaju se elementi	31
Slika 11. Optimizovan dizajn upotrebe IMDG struktura 1 – Dodaje se ili menja element 2 – Dodaje se element 3 – Dodaje se element 4 – Proverava se element 5 – Proverava se ili uklanja element 6 – Uklanja se element p – Proveravaju se i uklanjaju elementi	32
Slika 12. Režim sa kopiranjem sadržaja u Infinispan-u	35
Slika 13. Režim sa brisanjem sadržaja u Infinispan-u	36
Slika 14. Režim sa distribuiranjem sadržaja u Infinispan-u.....	37
Slika 15. Preuzimanje uloge rukovodećeg čvora u klaster režimu JBoss-a.....	38
Slika 16. Primer JCR strukture podataka tipa stabla.....	40
Slika 17. Rukovanje otkazima pri izvršavanju HA <i>Singleton</i> servisa.....	41
Slika 18. IoT ekosistem kreiran za ispitivanje rešenja	58

Slika 19. Lista prijavljenih CPE uređaja u GWT aplikaciji	60
Slika 20. Stablo modela podataka CPE uređaja u GWT aplikaciji	60
Slika 21. Primeri servisa koji koriste usluge ACS-a	61
Slika 22. Potrošnja RAM memorije pri opsluživanju 500 CPE klijenata	62
Slika 23. Potrošnja RAM memorije pri opsluživanju 1000 CPE klijenata	62
Slika 24. Potrošnja RAM memorije pri opsluživanju 1500 CPE klijenata	63
Slika 25. Opterećenje procesora pri opsluživanju 500 CPE klijenata	63
Slika 26. Opterećenje procesora pri opsluživanju 1000 CPE klijenata	63
Slika 27. Opterećenje procesora pri opsluživanju 1500 CPE klijenata	64
Slika 28. Grafik skalabilnost implementiranog klastera	65
Slika 29. Teorijska skalabilnost implementiranog poslužioca	67

SPISAK TABELA

Tabela 1. Načini rešavanja IoT zahteva u okviru TR-069 protokola.....	8
Tabela 2. HP klasifikacija klastera [18]	13
Tabela 3. Funkcionalni moduli biznis logike	51
Tabela 4. Implementirani vremenski okidači	55
Tabela 5. Funkcionalni moduli u TR-069 komunikacionom sloju [48].....	56
Tabela 6. Ispitivanje tehničkih zahteva	59
Tabela 7. Performanse fizičke arhitekture pri ispitivanju optimizacije	62
Tabela 8. Performanse fizičke arhitekture pri ispitivanju skalabilnosti	65

SKRAĆENICE

ACS	- <i>Auto-Configuration Server</i> , poslužilac za automatsko upravljanje
API	- <i>Application Programming Interface</i> , aplikativna programska sprega
CPE	- <i>Customer Premises Equipment</i> , potrošačka oprema
CWMP	- <i>CPE WAN Management Protocol</i> , CPE WAN protokol upravljanja
HA	- <i>High Availability</i> , visoka raspoloživost
HTTP	- <i>Hypertext Transfer Protocol</i> , mrežni protokol
HTTPS	- <i>Hypertext Transfer Protocol Secure</i> , bezbedni mrežni protokol
IMDG	- <i>In-Memory Data Grid</i> , mreže podataka koja se čuva u memoriji
IoT	- <i>Internet of Things</i> , Internet stvari
JCR	- <i>Java Content Repository</i> , Java repozitorijum sadržaja
IP	- <i>Internet Protocol</i> , Internet protokol
RPC	- <i>Remote Procedure Call</i> , poziv udaljenih procedura
SOAP	- <i>Simple Object Access Protocol</i> , komunikacioni protokol
STB	- <i>Set-Top Box</i> , digitalni TV prijemnici
SQL	- <i>Structure Query Language</i> , relacioni upitni jezik
TCP	- <i>Transmission Control Protocol</i> , protokol za sigurni prenos podataka
URL	- <i>Uniform Resource Locator</i> , jednoobrazni lokator resursa
WAN	- <i>Wide Area Network</i> , računarska mreža širokog područja

1. Uvod

U ovom radu je opisana realizacija sistema za opsluživanje u “Internet stvar” (*Internet of Things*, *IoT*) ekosistemu koji se zasniva na TR-069 (eng. Technical Report 069) komunikacionom protokolu. Predstavljeno rešenje je implementirano u okviru klaster sistema kako bi se obezbedio visok stepen skalabilnosti samog poslužioca.

Koncept IoT ima potencijal da uvede novu revoluciju na Internetu. IoT predstavlja vezu između stvari iz realnog sveta sa stvarima iz virtualnog sveta, drugim rečima većina objekata iz naše okoline će biti umreženi i imati svoju predstavu u digitalnom svetu. Umreženost svakodnevnih objekata omogućava praćenje njihovih stanja dok su informacioni sistemi u mogućnosti da prikupljaju ažurne podatke sa povezanih objekata i procesiraju ih. IoT koncept omogućava praćenje realnog sveta do nivoa detalja koji do sada nikad nije bio moguć po zanemarljivoj ceni. Takav pristup omogućava ne samo bolje razumevanje osnova procesa iz realnog sveta nego i bolju kontrolu i upravljanje samim objektima. Takođe, omogućuje da reakcije na kompleksne i kritične događaje iz realnog sveta budu automatske i pravovremene ali i da se optimizuju i poboljšaju poslovni procesi što može dovesti do poboljšanja u ekonomskim i socijalnim aspektima ljudskih života [1].

IoT ekosistemi se sastoje od velikog broja uređaja čija je fizička priroda veoma različita. Takvi ekosistemi zahtevaju potrebu za poslužiocima sposobnim da opsluže sve uređaje nezavisno od tipa uređaja. Iz tog razloga se javila potreba za realizacijom skalabilnog poslužioca koji je zasnovan na TR-069 protokolu.

TR-069 protokol predstavlja komunikacioni protokol za daljinsko nagledanje i upravljanje uređajima. TR-069 protokol omogućuje sledeće funkcionalnosti:

- praćenje rada velikog broja uređaja svih tipova.
- prikupljanje statističkih i dijagnostičkih informacija za grupu uređaja.

- obavljanje automatske ili manuelne konfiguracije uređaja.
- razmenu datoteka.

Protokol opisuje način razmene poruka između klijentske strane (eng. Customer Premises Equipment, CPE) i poslužioca (eng. Auto-Configuration Server, ACS). Svaki CPE uređaj sadrži model podataka koji jednoznačno određuje tip krajnjeg uređaja. Model podataka predstavlja skup parametara čija vrednost se skladišti u poslužiocu. Poslužilac je dalje u mogućnosti da te podatke procesira ili ih prosleđuje trećoj strani kako bi podaci imali interpretacionu vrednost. Dodatno, protokol omogućuje poslužiocu da registruje potencijalne probleme u radu krajnjih uređaja kao i mogućnost da te probleme reši na više načina.

Problem koji je rešavan u ovom radu, pored pokrivanja svih funkcionalnosti TR-069 protokola, bilo je omogućavanje poslužioca da radi u velikim i opterećenim mrežama, odnosno da poslužilac može da vrši nadgledanje velikog broja uređaja. Usled takvog zahteva razvijen je klaster sistem u kome se ceo sistem ponaša kao da postoji jedan jedinstveni poslužilac ali se u stvarnosti sastoji od N broja računarskih jedinica, tj. radnih stanica. Pored toga bilo je potrebno maksimalno optimizovati pristupe eksternim resursima kao što su strukture podataka koje se čuvaju u memoriji ili baza podataka. Time je postignut visok novi skalabilnosti poslužioca.

Ispitivanje sistema sa ranije opisanim zahtevima je izuzetno zahtevan zadatak. Za potrebe konkretnog ispitivanja potrebno je kreirati simulacioni IoT ekosistem. Za potrebe kreiranja takvog ekosistema korišćena je implementirana CPE biblioteka sa simulacijom rada digitalnog TV prijemnika (eng. Set-Top Box, STB). U toku ispitivanja skalabilnosti sistema prati se zauzetost resursa radnih stanica i maksimalan broj simulacionih klijentskih aplikacija koje sistem može da opsluži u zavisnosti od broja radnih stanica koja se koristi u klaster sistemu. Funkcionalni testovi su vršeni pomoću prezentacionih aplikacija kao što je Web i Android aplikacija. Ispitivanjem je utvrđeno da su zadovoljeni svi funkcionalni zahtevi poslužioca kao i da je postignut visok nivo skalabilnosti poslužioca. Potrebno je napomenuti da je ispitivanje vršeno u laboratorijskim uslovima i da bi došlo do manjeg odstupanja prilikom ispitivanja u realnim uslovima.

Rad se sastoji iz sledećih poglavlja:

- Teorijske osnove – sadrži osnovne principe IoT koncepta, osnove TR-069 protokola i principe klastera kao i primere implementacije klastera;
- Analiza problema – sadrži analizu potrebnih funkcionalnosti poslužioca, probleme optimizovanja poslužioca kao i analizu kreiranja klaster okruženja;
- Koncept rešenja - sadrži opis funkcionalnih modula poslužioca, način na koji je izvršena optimizacija i realizaciju klaster sistema;

- Programsko rešenje – sadrži detaljan opis funkcionalnih modula poslužioca i načine konfigurisanja svih komponenti klastera;
- Ispitivanje i rezultati – sadrži prikaz ispitivanja funkcionalnosti poslužioca, kao i opis i rezultate ispitivanja skalabilnosti poslužioca u različitim uslovima;
- Zaključak – sadrži kratak pregled implementiranog rešenja, zajedno sa mogućnostima daljeg napretka rešenja.

2. Teorijske osnove

U IoT ekosistemima prisutan je veliki broj „stvari“ iz realnog sveta koje su međusobno mrežno povezane ili su povezane sa servisima koji mogu biti smešteni u *Cloud* okruženje. Prema tome opterećenje na mreži je veliko, odnosno stvaraju se mreže velike gustine čime se nadgledanje svih objekata u ekosistemu znatno otežava. Postoji izvestan broj protokola koji teže da minimizuju opterećenje mreže ali i opterećenje uslužnih servisa kako bi se omogućilo nadgledanje i praćenje rada IoT ekosistema. Primeri takvih protokola su **UPnP** (*Universal Plug and Play*) protokol [2] ili **SNMP** (*Simple Network Management Protocol*) protokol [3]. Takođe za navedenu svrhu se koristi i TR-069 protokol [4] koji se zasniva na **SOAP** (*Simple Object Access Protocol*) protokolu [5].

2.1 Internet of Things koncept

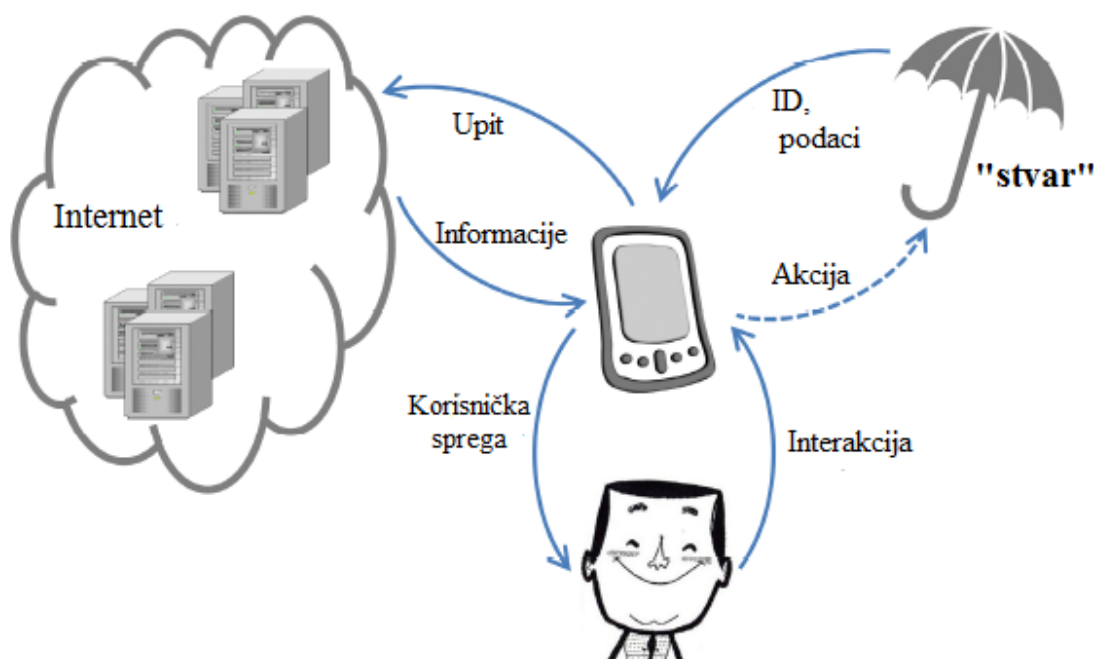
Internet of Things predstavlja koncept u kome se Internet i virtualni svet proširuju na stvarni, opipljivi svet [6]. Fizičke stvari poseduju i svoje virtualne predstave usled mogućnosti da se povežu na Internet i da budu jednoznačno identifikovane u celokupnom ekosistemu. IoT koncept predstavlja računarstvo kao jednu sveprisutnu nauku. Koncept je inicijalno osmišljen još 1990. godine od strane Mark Weiser-a [7]. Kasnije je termin „Internet of Things“ popularizovan u okviru RFID (*Radio-Frequency Identification*) projekta od strane Auto-ID centra sa MIT univerziteta [8].

IoT koncept je nastaje kao rezultat povezivanja nekoliko tehničkih dostignuća sa ciljem da se premoste razlike između realnog i virtualnog sveta [1]. Uključena su sledeća dostignuća:

- **Komunikacija i kooperacija** – objekti imaju sposobnost da se povežu na Internet ili da se međusobno povežu. Koriste se sledeće tehnologije: GSM, UMTS, WiFi, Bluetooth, ZigBee itd.

- **Adresiranje** – unutar IoT ekosistema objekti se mogu locirati i adresirati u svrhu udaljenog praćenja i konfigurisanja.
- **Identifikacija** – objekti se jedinstveno identifikuju. RFID, NFC(*Near Field Communication*) i optički čitljivi bar kodovi su neki od primera tehnologija koje se koriste.
- **Praćenje okoline** – objekti koriste senzore radi skupljanja informacija iz svoje okoline.
- **Pozicioniranje** – „pametni“ objekti poseduju znanje o svojoj trenutnoj geografskoj poziciji ili imaju sposobnost da budu locirani.
- **Korisnički interfejs** – upotrebom korisničkih interfejsa ljudi su mogućnosti da komuniciraju sa objektima iz IoT ekosistema direktno ili indirektno.

U većini slučajeva, specifičnim aplikacijama nisu potrebne sve nabrojane tehnologije već njihov podskup, ne samo zbog suvišne upotrebe određenih tehnologija, već i nemogućnosti da se različite tehnologije integrišu u jedan proizvod kao i zbog cene takvih proizvoda [9]. Vremenom bežični komunikacioni moduli postaju sve manji i jeftiniji, Internet protokol (eng. Internet Protocol, IP) verzija IPv6 se sve više koristi, kapaciteti memorijskih čipova rastu, a energetski zahtevi procesora po instrukciji nastavljaju da padaju. Mobilni telefoni danas poseduju ugrađene čitače bar kodova, NFC tehnologiju i ekrane osetljive na dodir i time se stvaraju mogućnosti preuzimanja uloge posrednika između ljudi, svakodnevnih objekata i Interneta kao što je prikazano na Slici 1. Sve nabrojano doprinosi IoT paradigmi: sa udaljenom identifikacijom objekata i Internetom, mi se krećemo ka sistemu gde „pametni“ objekti stvarno komuniciraju sa korisnicima, Internet servisima ali i međusobno. Ove mogućnosti omogućavaju nove funkcionalnosti, ali sa druge strane uvode nova pitanja koja se odnose na osnovne tehnologije i infrastrukturu. U stvari, infrastruktura IoT ekosistema mora biti ne samo efikasna, prilagodljiva, pouzdana i sigurna već mora biti u skladu sa političkim i društvenim očekivanjima, biti široko primenjena uz ograničena ekonomska sredstva[10].



Slika 1. Mobilni telefon kao posrednik između „stvari“ i Interneta [1]

Pored svih prednosti koje IoT koncept donosi postavlja se pitanje tehnološke implementacije takvog koncepta. Usled velikog broja činilaca IoT ekosistema tehnologija koja se u njih integriše mora biti jednostavna i jeftina. Tehnološki izazovi [1] sa kojima se susreće IoT koncept su:

- **Skalabilnost** – komunikacioni moduli i brojni servisi unutar IoT ekosistema treba da funkcionišu kako u mrežama male gustine tako i u mrežama velike gustine.
- **„Prijava i konfiguracija“** – „pametne“ stvari iz realnog života ne treba da zahtevaju ljudsko upravljanje već je potrebno da poseduju sposobnost samostalnog, automatskog prijavljivanja na IoT servise kao i mogućnost samostalnog konfigurisanja.
- **Interoperabilnost** – zbog velike različitosti objekata iz realnog sveta svaki tip objekta zahteva prikupljanje različitih informacija, različite nivoe mrežne propusnosti, performanse fizičke arhitekture ali i napajanje energijom.
- **Razotkrivanje** – u dinamičkom ekosistemu, objekti moraju posedovati sposobnost da se razotkriju novim servisima u ekosistemu što zahteva unapred definisanu semantiku radi opisivanja sopstvenih funkcionalnosti. Npr. kupci kada uđu u neku prodavnicu ili robnu kuću bi preko svojih uređaja (mobilnih telefona) mogli da saznaju informacije vezane za neki proizvod i koriste mehanizme pretrage da bi pronašli odgovarajući proizvod ili informaciju.

- **Kompleksnost programske podrške** - programska podrška na „pametnim“ stvarima mora biti jednostavna i minimalistička, u skladu sa principima namenskih (eng. embedded) sistema, što dovodi do potrebe postojanja pozadinskih poslužilaca u ekosistemu koji će posedovati kompleksnu programsku podršku u svrhu upravljanja „pametnim“ stvarima i obezbeđivanjem raznih servisa.
- **Veličina podataka** – podaci koji se razmenjuju između poslužioca i korisničkih uređaja mogu dostići ogromne količine usled čega su potrebni ogromni memorijski prostori na poslužiocima kako bi se mogli smestiti svi neophodni podaci.
- **Interpretacija podataka** – usled velike različitosti „pametnih“ stvari i različitosti podataka koje generišu, servisi IoT ekosistema se snabdevaju različitim tipovima podataka koje je potrebno interpretirati i procesirati na različite načine, kako bi sirovi podaci dobili smisljeno značenje.
- **Bezbednost i privatnost podataka** – veoma bitan izazov koji je poznat od samog nastanka Interneta. IoT koncept unosi još veću potrebu za obezbeđivanjem bezbednošću podataka ali i za selekcijom prava pristupa različitim informacijama i servisima.
- **Tolerancija na greške** – IoT koncept predstavlja dinamički ekosistem sa mnoštvom objekata različitih karakteristika što unosi velike mogućnosti za pojavu velikih promena stanja unutar ekosistema i pojavu grešaka, pa je takvom ekosistemu potreban visok nivo tolerancije na greške.

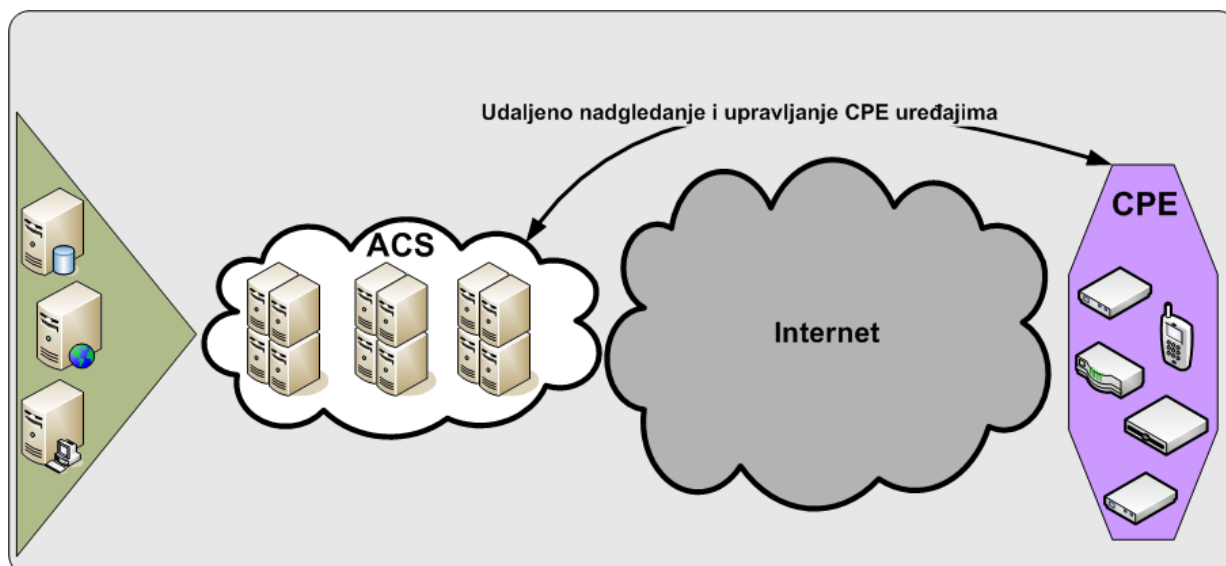
2.2 TR-069 protokol

Navedeni zahtevi koji su potrebni da se ispune kako bi IoT ekosistem pravilno funkcionisao se u većoj meri mogu obezbediti upotrebom TR-069 protokola. U Tabeli 1 navedeni su zahtevi IoT koncepta i načini kako se oni rešavaju u okviru TR-069 protokola.

TR-069 protokol je tehnička specifikacija definisana od strane Broadband Forum-a. Protokol opisuje CPE WAN protokol upravljanja (eng. CPE WAN Management Protocol, CWMP) namenjen za komunikaciju između klijentske aplikacije (CPE) i poslužioca za automatsku konfiguraciju (ACS). Na Slici 2 prikazana je arhitektura sistema na najvišem nivou zasnovanog na TR-069 protokolu.

Naziv zahteva	Načini rešavanja
Skalabilnost	Upotrebom periode prijavljivanja i definisanjem politike praćenja rada uređaja moguće je smanjivati ili povećavati frekvenciju prijavljivanja uređaja i time regulisati opterećenje mreže.
„Prijava i konfiguracija“	Upotrebom metoda definisanih u okviru TR-069 protokola ACS je u mogućnosti da automatski detektuje pojavu novog uređaja u ekosistemu i izvrši njegovu konfiguraciju.
Interoperabilnost	Upotrebom atributa parametara u modelu podataka, o čemu će biti kasnije reči, ACS je u mogućnosti da podešava politiku praćenja rada uređaja na osnovu tipa uređaja.
Razotkrivanje	Upotrebom mehanizma za uspostavu veze, takođe više reči u daljem tekstu, ACS je u mogućnosti da uspostavi prevremeno prijavljivanje uređaja i sprovede potrebne akcije.
Kompleksnost programske podrške	Programska podrška integrisana u klijentske uređaje zahteva malo resursa iz razloga minimalnog broja funkcionalnosti koje obezbeđuje i koje se prevashodno odnose na komunikaciju sa ACS-om.
Veličina podataka	Zahtev se najviše odnosi na memorijske kapacitete poslužioca. Međutim TR-069 protokol omogućava definisanje politike praćenja rada uređaja u zavisnosti od tipa čime se može smanjiti količina podataka koja se prosleđuje.
Interpretacija podataka	Upotrebom definisanih modela podataka koji zavise od tipa uređaja ACS je u mogućnosti da na pravi način interpretira sve podatke pristigle sa uređaja.
Bezbednost i privatnost podataka	Upotrebom mehanizama šifrovanja komunikacionih poruka kao i upotrebom potvrde identiteta (eng. Authentication) i kreiranjem vremenski ograničenih sesija postiže se visok nivo bezbednosti i privatnosti podataka.
Tolerancija na greške	TR-069 protokol definiše i poruke greške kojima se prosleđuju identifikatori grešaka i njihovi opisi.

Tabela 1. Načini rešavanja IoT zahteva u okviru TR-069 protokola



Slika 2. Arhitektura TR-069 sistema na najvišem nivou

TR-069 protokol propisuje skup poziva udaljenih procedura (eng. Remote Procedure Call, RPC). SOAP protokol koji je zasnovan na EXtensible Markup Language (XML) jeziku koristi se za enkodovanje pomenutih RPC poziva. Pozivi se prosleđuju između ACS i CPE upotrebom Hypertext Transfer Protokol (HTTP) ili HTTP Secure (HTTPS) protokola. Jedna od osnovnih RPC metoda jeste *Inform* metoda. *Inform* metoda se prosleđuje isključivo od strane CPE-a ka ACS-u i njenom upotrebom CPE obaveštava ACS o svom prisustvu. *Inform* zahtev predstavlja začetak sesije unutar koje ACS i CPE prosleđuju različite RPC zahteve u jednom ili drugom smeru. Sesija inicijalizovana od strane CPE-a započinje uspostavom veze sa ACS-om i slanjem *Inform* RPC metode. Protokol zahteva da CPE poznaje jednoobrazni lokator resursa (eng. Uniform Resource Locator, URL) ACS-a kako bi iniciranje sesije bilo moguće. Razlozi uspostave sesije mogu biti sledeći:

- Prvi put kada se CPE prijavi u ekosistemu.
- Pri ponovnom pokretanju CPE-a.
- Kada istekne vreme zadato *InformPeriodicInterval* parametrom – periodično prijavljivanje na ACS.
- Kada CPE primi zahtev za uspostavom veze od strane ACS-a.
- Kada se promeni URL ACS-a.
- Kada se promeni vrednost parametara na čije praćenje je ACS pretplaćen.
- Kada se uspešno ili neuspešno izvrši preuzimanje ili otpremanje datoteka.
- Kada je sesija neuspešno okončana.

TR-069 protokol takođe definiše i mehanizam za uspostavu veze (eng. Connection Request) kojim je ACS u mogućnosti da inicira prevremeno prijavljivanje CPE-a na ACS,

odnosno prevremeno slanje *Inform* RPC zahteva. Definisani su različiti mehanizmi za uspostavu veze:

- *TR-069 Connection Request* – primenjuje se kada su ACS i CPE u istoj lokalnoj mreži (eng. Local Area Network, LAN), odnosno kada su IP adrese ACS-a i CPE-a u istom domenu.
- *STUN* - primenjuje se kada su ACS i CPE nisu u istoj LAN mreže pri čemu se IP adrese nisu iz istog domena. Način korišćenja *STUN* mehanizma opisan je u okviru TR-111 specifikacije [11].
- *Extensible Messaging and Presence Protocol (XMPP)* - primenjuje se istim slučajevima kao i STUN mehanizam. Definisan je u okviru verziji 5 TR-069 protokola [12].

Skup RPC metoda je definisan TR-069 specifikacijom i njihov broj se menja kroz verzije specifikacije, kako dodavanjem novih tako i izbacivanjem postojećih iz prethodnih verzija. Funkcionalnosti koje su omogućene upotrebom RPC metoda su sledeće:

- Prosleđivanje strukture modela podataka od strane CPE-a.
- Prosleđivanje i postavljanje vrednosti parametara iz modela podataka.
- Prosleđivanje i postavljanje vrednosti atributa parametara iz modela podataka.
- Preuzimanje i otpremanje datoteka sa navedene URL adrese.
- Ponovo pokretanje CPE-a sa ili bez vraćanja CPE-a u inicijalno stanje.

Model podataka (eng. Data Model) ima veoma važnu ulogu u realizaciji TR-069 protokola. Model podataka je jedini element protokola koji je zavisao od tipa uređaja i kao takav jedini način da se uređaji razlikuju i identifikuju u okviru TR-069 protokola. Dok TR-069 specifikacija propisuje način komunikacije između klijentske strane i poslužioca, TR-106 specifikacija [13] opisuje način na koji se definišu modeli podataka, odnosno sadrži XML šemu (eng. XML Schema Definition, XSD) koja definiše pravila kreiranja modela podataka. Pored toga TR-106 specifikacija propisuje i skup elementarnih parametara kako bi protokol pravilno funkcionisao i kako bi se svaki CPE mogao jedinstveno identifikovati u ekosistemu. Broadband Forum je definisao i nekoliko modela podataka za najčešće korišćene uređaje kao što su: TR-135 za digitalne TV prijemnike [14], TR-098 za uređaje mrežnog prolaza (*Internet Gateway Devices*) [15], TR-140 za uređaje za skladištenje podataka [16] itd.

Model podataka predstavlja skup parametara organizovanih u obliku strukture podataka tipa stabla (eng. Tree). Stablo je organizovano tako da parametar uvek predstavlja njen list dok su čvorovi stabla objekti koji kao svoje potomke mogu da sadrže nove objekte ili parametre. Objekti mogu biti definisani kao višestruki, što znači da može postojati više instanci jednog

objekta. Dodavanje i uklanjanje višestrukih objekata je moguće izvršiti upotrebom RPC poziva. Svaki parametar pored svoga naziva i vrednost ima definisan svoj tip i ograničenja, npr. maksimalnu ili minimalnu vrednost parametra. Takođe parametar sadrži i attribute čijom upotrebom se može podesiti politika praćenja rada uređaja kao i prava pristupa. Definisani su sledeći atributi:

- **Nivo obaveštavanja** – definišu se tri nivoa obaveštavanja.
 1. Onemogućeno obaveštavanje - promena vrednosti parametra se ACS-u nikada ne prijavljuje osim na eksplicitan zahtev ACS-a.
 2. Pasivno obaveštavanje - promena vrednosti parametra se ACS-u prijavljuje pri prvoj narednoj sesiji.
 3. Aktivno obaveštavanje - promena vrednosti parametra se ACS-u prijavljuje čim se desila, pri čemu je potrebno inicirati sesiju.
- **Lista prava pristupa** – sadrži listu pretplatnika koje imaju prava da menjaju vrednost parametra. Lista se ne odnosi na ACS koji uvek ima pravo upisa vrednosti ukoliko je parametar definisan kao upisiv.

2.3 Klasteri

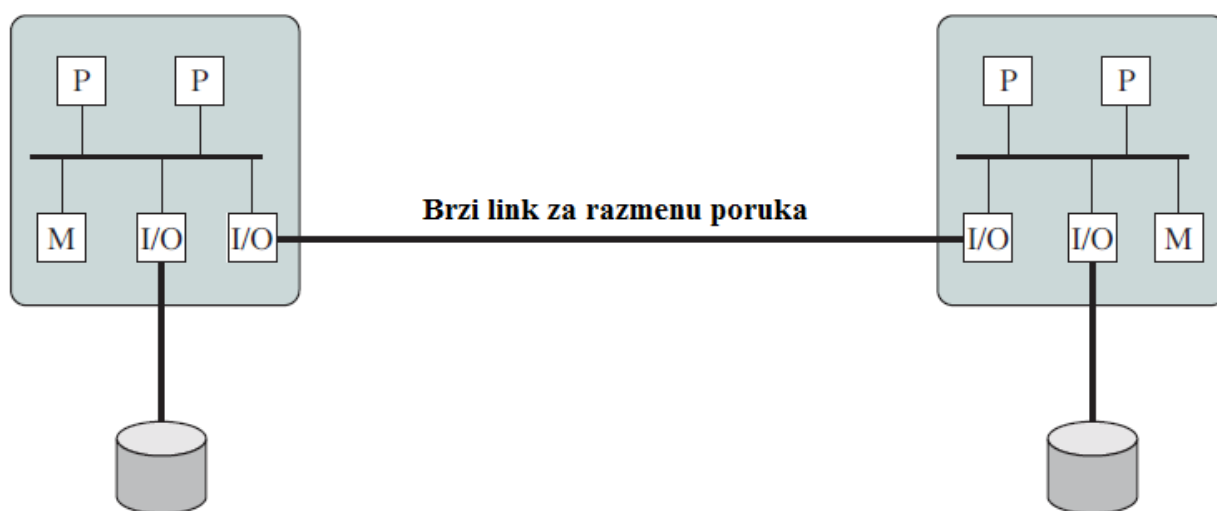
Klasterovanje je kao rešenje alternativa simetričnoj višeprocenoj obradi (eng. Symmetric Multiprocessing, SMP) kojim se obezbeđuju visoke performanse i visoka raspoloživost [17]. Klasterovanje je rešenje koje je našlo veliku primenu u implementaciji poslužioca. Klasterovanjem poslužioca se u velikoj meri poboljšava skalabilnost poslužioca kao i sposobnost opsluživanja velikog broja klijenata. U velikim i gustim IoT ekosistemima primena klastera je gotovo neophodna da bi se ispunili svi zahtevi takvog ekosistema.

Klaster predstavlja grupu radnih stanica koje se nazivaju čvorovi. Klaster se može definisati kao grupa međusobno povezanih čvorova koji rade zajedno kao ujedinjen računarski sistem čime se stvara iluzija postojanja jednog računara. Čvor unutar klastera predstavlja jedan računarski resurs koji može nezavisno da radi van klaster sistema. Prednosti koje donosi upotreba klastera su sledeće:

- **Apsolutna skalabilnost** – kako se klaster sistem može sastojati od velikog broja čvorova njihova računarska snaga može nadmašiti i najveće samostalne računare;
- **Povećavajuća skalabilnost** – klaster sistemi su proširivi jer se novi čvorovi mogu lako dodavati uz mali napor;

- **Visoka raspoloživost** - kako je svaki čvor nezavisan u sistemu, otkaz jednog čvora ne znači i otkaz celog sistema. Posredna programska podrška klastera je zadužena za oporavak od grešaka kao i eventualnog gubitka podataka.
- **Bolji odnos cena/performance** – moguće je kreirati klaster sa istom ili većom računarskom snagom u odnosu na zaseban super računar po mnogo nižoj ceni.

Klasifikacija klastera može se izvršiti na najjednostavniji način prema tome da li čvorovi u klasteru dele pristup do istih diskova. Na Slici 3 prikazan je klaster koji se sastoji od dva čvora, čija je jedina međusobna veza veoma brzi link koji se koristi za razmenu poruka kako bi se uskladio rad oba čvora. Kao brzi link može se koristiti LAN mreža ili namenski uređaj za međusobno povezivanje. U drugom slučaju, jedan ili više računara iz klastera moraju imati vezu sa LAN ili regionalnom računarskom mrežom (eng. Wide Area Network, WAN) kako bi klijentski računari, koji nisu deo klastera, mogli da koriste usluge klastera. Na Slici 4 prikazan je klaster sa zajedničkim diskom. Pored linka za poruke između čvorova prisutan je i zajednički podsistem diskova – redundantni niz nezavisnih diskova (eng. Redundant Array of Independent Disks, RAID). Upotreba redundantne tehnologije diskova je uobičajena u klasterima kako bi se obezbedila visoka raspoloživost, jer u slučaju otkaza diska gube se svi podaci ukoliko ne postoje kopije podataka.

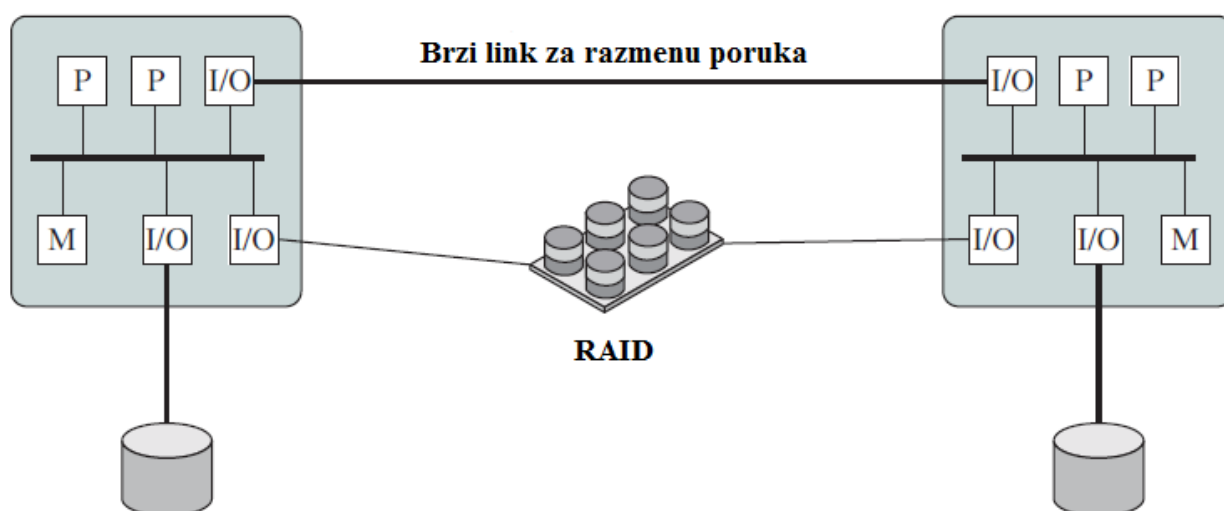


Slika 3. Konfiguracija klaster sistema bez deljenja diska [18]

U Tabeli 2 prikazana je klasifikacija klastera na osnovu rada klastera definisana od strane Hewlett Packard (HP) kompanije.

Metode klasterovanja	Opis	Prednost	Ograničenja
Pasivno stanje pripravnosti	U jednom trenutku je aktivan samo jedan čvor dok su ostali u pripravnosti čekajući da preuzmu posao u slučaju otkaza.	Lako za primenu.	Visoka cena pošto su sekundarni čvorovi nedostupni za ostale poslove.
Aktivan sekundarni čvor	Sekundarni čvor se takođe koristi za obradu.	Smanjeni troškovi pošto se sekundarni čvorovi mogu koristiti za obradu.	Povećana složenost.
Zasebni čvorovi	Zasebni čvorovi imaju sopstvene diskove. Da bi se obezbedilo upravljanje otkaza podaci se neprekidno kopiraju sa primarnog čvora na sekundarne.	Visoka raspoloživost.	Veliko opterećenje mreže i čvorova zbog potrebe kopiranja podataka.
Čvorovi povezani sa diskovima	Čvorovi su povezani kablovima sa istim diskovima, ali diskovi su podeljeni u particije gde svaka particija pripada pojedinom čvoru. Ako jedan čvor otkaze, njegove diskove preuzimaju ostali čvorovi.	Smanjeno opterećenje mreže i čvorova izazvano uklanjanjem operacije kopiranja.	Zahteva preslikavanje diskova ili upotrebu RAID tehnologije kako bi se nadoknadio rizik od otkaza diska.
Čvorovi dele diskove	Više čvorova istovremeno pristupa deljenim diskovima, odnosno svaki čvor može da pristupi svim podacima u bilo kom trenutku.	Malo opterećenje mreže i čvorova. Smanjena opasnost od praznog hoda izazvanog otkazom diska.	Zahteva programsku podršku za zaključavanje pristupa deljenim diskovima kako ne došlo do trke podataka. Zahteva preslikavanje diskova ili upotrebu RAID tehnologije kako bi se nadoknadio rizik od otkaza diska

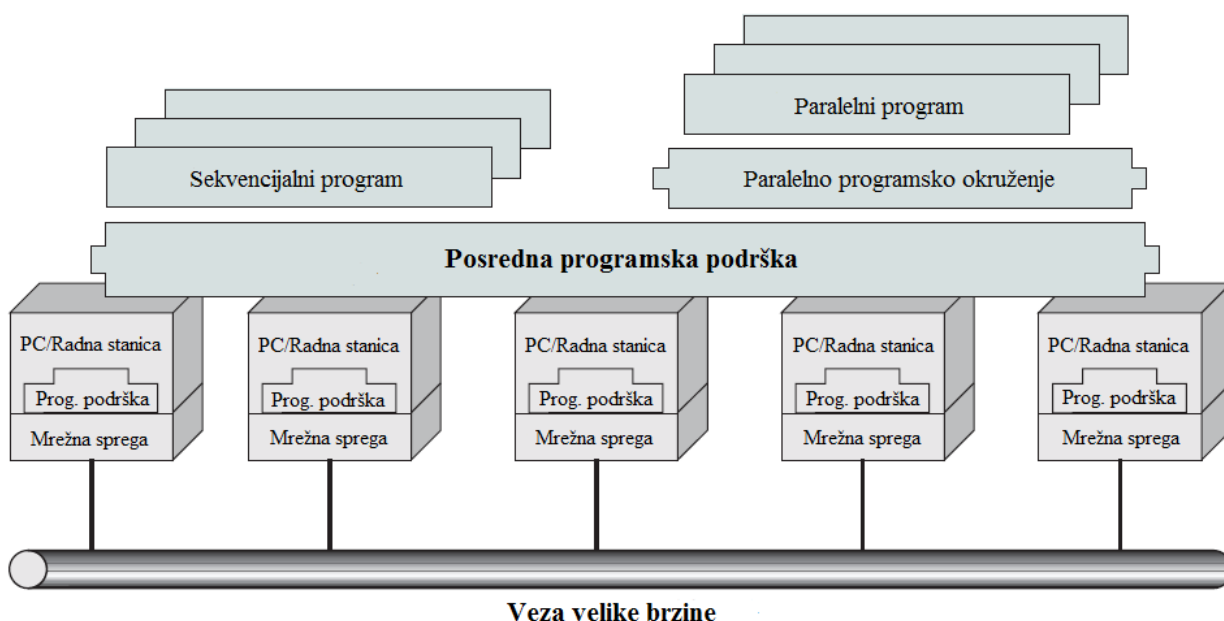
Tabela 2. HP klasifikacija klastera [18]



Slika 4. Konfiguracija klaster sistema sa deljenjem diska [18]

2.3.1 Arhitektura klastera

Na Slici 5 je prikazana je tipična arhitektura klastera. Pojedinačni čvorovi povezani su nekom vrstom brze LAN mreže ili komutacionim računarskim resursom. Svaki računar može da radi nezavisno. Pored toga, posredna programska podrška nalazi se na svakom računaru kako bi se omogućio rad klastera. Posredna programska podrška pruža objedinjenu sliku sistema, odnosno sliku jedinstvenog sistema. Posredna programska podrška je takođe odgovorna za obezbeđivanje visoke raspoloživosti, uravnoteženja opterećenja i obezbeđivanjem oporavka klastera i slučajevima otkaza pojedinih komponentama.



Slika 5. Arhitektura klaster sistema

Klaster zahteva efikasan način uravnoteženja opterećenja između raspoloživih računara [19]. Dodatno, time se obuhvata i zahtev da se klaster sistem može postepeno proširivati. Kada se priključi novi računar u klaster, komponenta za uravnoteženje opterećenja treba automatski da uključi taj računar u raspodelu poslova. Postoji veliki broj algoritama koji se primenjuju radi što boljih rezultata uravnotežavanja, neki od njih su:

- **Nasumice** (eng. Random) – odluka o čvoru koji preuzima pristigle zahteve se donosi na osnovu nasumice dobijenog broja koji predstavlja redni broj čvora.
- **Razigravanje** (eng. Round Robin) – odabira se čvor koji je sledeći u redu za čekanje, pri čemu se stalno kruži u odabiru čvora. Ovaj algoritam radi izuzetno dobro u većini klastera ali samo ukoliko su svi čvorovi u klasteru jednaki po svojoj računarskoj snazi.
- **Razigravanje sa težinskim faktorom** (eng. Weighted Round Robin) – svakom čvoru je dodat težinski faktor koji označava koliko više neki čvor može da obradi zahteva, odnosno da se u okviru kružnog reda čekanja nalazi višestruki broj učešća čvorova sa većim težinskim faktorom.
- **Dinamičko razigravanje** (eng. Dynamic Round Robin) – sličan algoritam kao i razigravanje sa težinskim faktorom sa razlikom da se faktor dinamički određuje, u toku rada i menja se kroz rad. Ovaj algoritam zahteva analizu performansi čvorova u realnom vremenu.
- **Najbrži** (eng. Fastest) – algoritam prosleđuje pristigle zahteve čvoru koji ima najmanje vreme potrebno za odgovor u odnosu na ostale čvorove.
- **Najmanji broj veza** (eng. Least Connections) – pristigli zahtevi su prosleđuju čvoru koji ima najmanji broj trenutnih veza. Slično kao i kod algoritma razigravanja, ovaj algoritam podrazumeva da su svi čvorovi u klasteru jednaki po svojoj računarskoj snazi.
- **Posmatranje** (eng. Observed) – algoritam predstavlja kombinaciju algoritma najmanji broj veza i najbrži algoritam. Čvor se izabira na osnovu broja trenutnih veza i na osnovu vremena potrebno za odgovor.
- **Predskazujući** (eng. Predictive) – sličan algoritmu posmatranja sa razlikom da se u analizu uključuje i istorija rangiranja čvorova kako bi se posmatralo da li su performanse čvora opale ili porasle tokom rada, i kako bi se ta informacija iskoristila pri odlučivanju čvora.

Upravljanje otkazima u klaster sistemima je veoma bitan faktor kako bi klaster pravilno funkcionisao. Moguća su dva pristupa u slučaju otkaza: klasteri visoke raspoloživosti i klasteri

otporni na greške. Klasteri visoke raspoloživosti nude veliku verovatnoću da će svi resursi raditi. Ako se otkaz, kao što je prestanak rada računara ili gubitka diska, ipak desi tada se upiti koje su bili u toku izvršavanja gube. Ako se izgubljeni upit ponovi, opslužiće ga drugi računar iz klastera. Međutim takav sistem ne garantuje šta će se desiti sa delimično obavljenim transakcijama. U tom slučaju programska podrška preuzima upravljanje nepotpunih transakcija. Sa druge strane, klasteri otporni na greške obezbeđuju da su svi resursi uvek raspoloživi. To se postiže upotrebom redundantnih deljenih diskova i mehanizmom kojim se prave rezervne kopije neokončanih transakcija i predaja samo potpuno okončane transakcije.

2.3.2 Primeri klastera

Windows klaster poslužilac je klaster bez deljenjem resursa, u kome svaki čvor sadrži sopstveni disk i ostale resurse [20]. Windows klaster poslužilac se opisuje na sledećim pojmovima:

- **Uslužni programi klastera** – programske podrške koje se nalaze u svakom čvoru i koje služe za upravljanje svim aktivnostima u klasteru;
- **Resurs** – stavka kojom uslužni programi klastera upravljaju. Resursi su objekti koji predstavljaju stvarne resurse u sistemu, uključujući fizičku arhitekturu kao što su diskovi i mrežne kartice ali i logičke stavke kao što su particije diska, baza podataka, Transmission Control Protocol/Internet Protocol (TCP/IP) adrese itd;
- **Status aktivnosti** – za neke resurse se kažu da imaju aktivan status u nekom čvoru kada u okviru tog čvora resurs pruža određene usluge;
- **Grupa** – kolekcija resursa kojom se upravlja kao jedinstvenom celinom. Obično grupa sadrži sve elemente neophodne određenoj programskoj podršci da bi se pokrenula i klijentu da bi se povezao sa uslugama koje pruža programska podrška.

Pojam grupe je od izuzetne važnosti. Grupa ujedinjuje resurse u veće celine kojima se lakše upravlja u slučajevima predaje i uravnotežavanja opterećenja. Operacije koje se izvršavaju na grupi, kao što je prenošenje grupe u drugi čvor, automatski utiču na sve resurse u toj grupi. Resursi su implementirani kao biblioteke dinamičkih veza (eng. Dynamic Linked Libraries, DLL) i njima upravlja upravljački program za nagledanje resursa. Taj program sarađuje sa klaster servisom preko udaljenih procedura i odgovara na komande klaster servisa tako što konfiguriše i premešta grupe resursa. Upravljački program čvora odgovoran je za članstvo tog čvora u klasteru. Periodično upravljački program prosleđuje poruku ostalim upravljačkim programima o svom prisustvu. Ukoliko neki od upravljačkih programa ustanovi da je poruka od

određenog čvora izostala on obaveštava ostale čvorove koji proveravaju koji sve čvorovi postoje u klasteru. Ukoliko se ustanovi da je više ne postoji taj čvor on se uklanja iz klastera. Upravljački program baze podataka konfiguracije klastera održava bazu podataka o konfiguraciji klastera. Baza podataka sadrži informacije o resursima i grupama kao i ko je vlasnik određene grupe. Upravljač događajima povezuje sve komponente uslužnih programa klastera, rukovodi zajedničkim operacijama i upravlja inicijalizacijom usluga klastera.

Red Hat Cluster Suite predstavlja skup programskih alata kojima se kreira klaster velike dostupnosti i sa raspoređivanjem opterećenja [14]. Kluster se sadrži sledećih funkcionalnosti:

- **Upravljanje klasterom** – sistem koji upravlja svim članovima klastera, vodi računa o njihovoj aktivnosti i prati kvorum klastera. Upravljački sistem je distribuiran na sve čvorove klastera. Praćenjem aktivnosti svih čvorova u klasteru upravljački sistem proverava da li je polovina ili više čvorova aktivna i u tom slučaju kvorum je zadovoljen. U suprotnom, ako je više od pola neaktivno kvorum nije zadovoljen i ceo klaster prestaje sa radom. Upotrebom kvorum mehanizma se izbegava pojava „split-brain“ stanja – pojava kada su pokrenute dve instance istog klastera i pristupaju istim resursima klastera ne znajući da postoji još jedna instanca koja pristupa istim resursima. Svaka instanca sistema za upravljanje klasterom proverava da li se aktivnost nekog čvora promenila ili je došlo do pojave novog čvora i u tom slučaju se sprovode odgovarajuće akcije kao i obaveštavanje ostalih čvorova.
- **Upravljanje resursima** – sistem zadužen za sinhronizaciju pristupa resursima klastera. Sistem za upravljanje resursima je takođe distribuiran na sve čvorove klastera.
- **Ograđivanje** – sistem zadužen za ograđivanje se aktivira kada čvor postane nefunkcionalan. Sistem za ograđivanje se nalazi na svakom čvoru i u slučaju grešaka na čvoru aktivira se metoda za ograđivanje. Moguće je definisati jednu ili više metoda. Kada postoji više metoda one se koriste redom kojim su definisane pri konfiguraciji čvora. Ukoliko prva metoda ne izvrši uspešno proces ograđivanja aktivira se sledeća, i tako redom. Kada drugi čvorovi registruju da čvor nepravilno radi, najpre se aktivira proces ograđivanja a zatim se sprovode dodatne operacije ako su potrebne, kao što je oporavak podatka ili tekuće sesije. Ograđivanje se može vršiti na nekoliko načina, bilo potpunim gašenjem čvora, onemogućavanjem čvora da komunicira ili prekidanjem fizičkih veza. Međutim, svrha ograđivanja je uvek ista a to je da se čvor odstrani iz klastera i onemogućiti mu se pristup resursima klastera.

- **Upravljanje konfiguracijom klastera** – sistem obezbeđuje da su svi čvorovi ažurno konfigurisani. U slučajevima kada klaster administrator promeni konfiguraciju na čvoru A sistem će ažurirati konfiguraciju na svim ostalim čvorovima.

Red Hat Cluster Suite sadrži i komponentu za obezbeđivanje servisa velike dostupnosti (eng. High Availability, HA). Komponenta obezbeđuje izvršavanje servisa dokle god postoji čvor na kome se servis može izvršavati. Komponenta uvodi pojam domena koji predstavlja grupu čvorova koji se koriste za obezbeđivanje HA servisa. Ukoliko otkáže čvor koji izvršava HA servis sledeći čvor iz domena preuzima njegovu ulogu.

2.3.3 Primena klastera u IoT ekosistemu

Kako bi se zadovoljili principi IoT koncepta korišćen je klaster sa deljenjem diska pri čemu je svakom čvoru klastera omogućen pristup svim podacima u bilo kom trenutku. Svi čvorovi u klasteru se ponašaju kao nezavisne jedinice što znači da svaki zahtev klijenta može biti obrađen od strane bilo kog čvora u klasteru. Time se postiže visoka raspoloživost klastera koja je neophodna u IoT ekosistemima. Za potrebe konkretne implementacije poslužioca kao algoritma za uravnotežavanje opterećenja korišćen je algoritam razigravanja sa težinskim faktorom.

2.4 Cloud tehnologije

Postoji mnogo pokušaja definisanja šta je *Cloud* i šta je njegova prava uloga. Jedna od definicija *Cloud*-a data je u [22]: “*Cloud* predstavlja dinamičko okruženje za izvršavanje resursa koje može uključivati više aktera izvršavanja i obezbeđuje ograničene servise podeljene na nivoe prema definisanim kvalitetima usluge (servisa)”. Drugim rečima, *Cloud* predstavlja platformu ili infrastrukturu koja omogućuje izvršavanje programske podrške u dinamičkom, prilagodljivom i upravljanom okruženju. Upravljano okruženje se odnosi na sposobnost *Cloud*-a da se na osnovu definisanih parametara osigura pouzdanost sistema, dok se dinamičko okruženje odnosi na sposobnost da se upravlja resursima tako što se njihov rad prilagođava trenutnom opterećenju sistema.

Obezbeđivanjem dinamičkog i upravljano okruženja, *Cloud* je najveću primenu našao u Internet-u. Programske podrške poslužioca su integrisane u okviru *Cloud*-a i kao takav sistem predstavljaju poslužioce koji obezbeđuju razne usluge udaljenim korisnicima u ekosistemu.

Postoje sledeći tipovi primene *Cloud*-a:

- **Infrastructure as a Service (IaaS)** – omogućuje se upotreba upravljanih i skalabilnih resursa kao servisa, odnosno obezbeđuju se unapređena virtualna okruženja. Različiti resursi mogu biti ponuđeni kao servisi:
 - **Skladištenje podataka** – primeri su: *Amazon S3*[23], *SQL Azure*[24];
 - **Računarski sistem** - omogućava upotrebu računarskog sistema, kao što su procesori. Primeri su: *Amazon EC2*[25], *Zimory*[26], *ElasticHosts*[27];
- **Platform as a Service (PaaS)** – omogućuje upotrebu računarskih resursa kao platformi za razvoj i izvršavanje aplikacija i servisa. Uobičajeno je da PaaS obezbeđuje sprege čijom upotrebom se može kontrolisati ponašanje izvršnog okruženja prema klijentskim zahtevima. Primeri su: *Force.com*[28], *Google App Engine*[29], *Windows Azure (Platform)*[30].
- **Software as a Service (SaaS)** – omogućuje upotrebu implementiranih specifičnih aplikacija ili servisa koje se izvršavaju u okviru *Cloud* okruženja. Današnji trend je da se standardne aplikacije nude u *Cloud*-u kako bi njihovo korišćenje bilo nezavisno od platforme koju korisnici koristi ili fizičke lokacije. Primeri su: *Google Docs*[31], *Salesforce CRM* [32], *SAP Business by Design* [33].

Takođe, *Cloud* rešenja se mogu razlikovati i po načinu korišćenja, odnosno funkcionalnostima koje se pružaju kroz *Cloud* u skladu sa poslovnim modelom provajdera *Cloud* sistema [34]. Prema takvoj podeli mogu se definisati sledeći oblici *Cloud*-a:

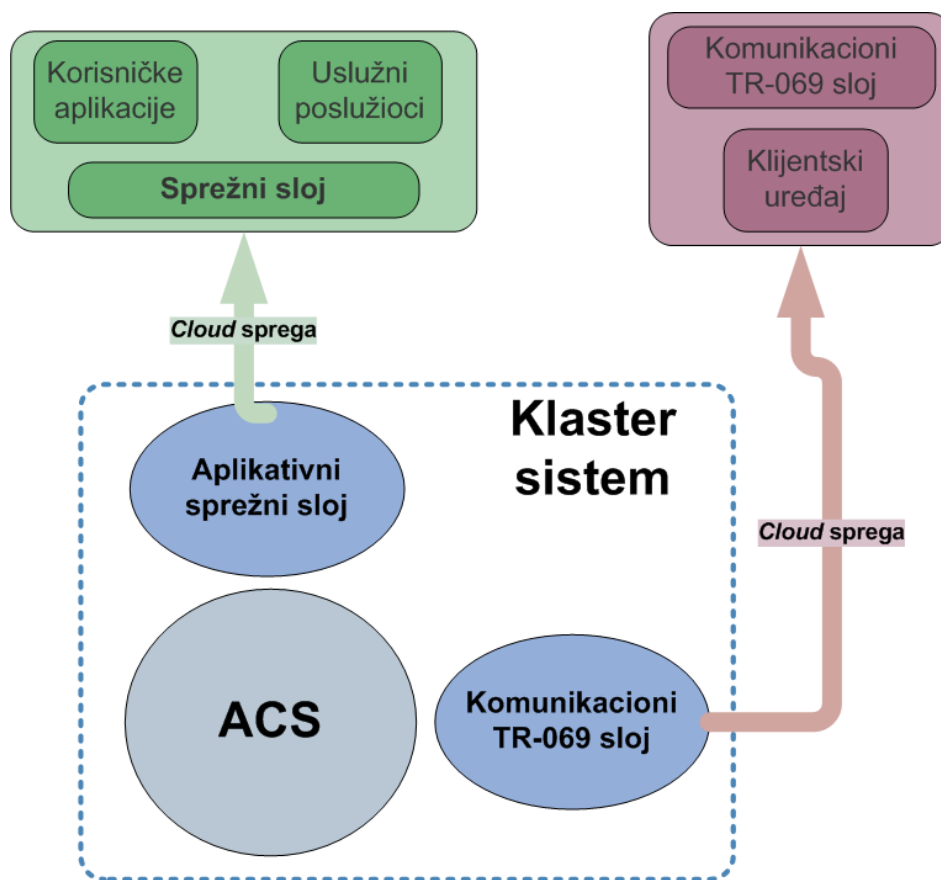
- **Privatni Cloud** – funkcionalnosti nisu direktno izložene korisnicima, mada postoje slučajevi kada su određene funkcionalnosti ponuđene korisnicima. Jasan primer je *eBay*;
- **Javni Cloud** – provajderi kao i drugi korisnici mogu koristiti usluge *Cloud*-a drugih provajdera, ili provajderi omogućavaju korisnicima upotrebu svojih *Cloud*-a usluga. Takav sistem omogućava korisnicima koji poseduju svoje servise da koriste *Cloud* infrastrukturu drugih provajdera i time smanjuju cene i napor kreiranja vlastitog *Cloud*-a. Primeri su: *Amazon*, *Google Apps*, *Windows Azure*;
- **Hibridni Cloud** – upotrebom javnih *Cloud* rešenja, provajderi servisa koji se izvršavaju u okviru *Cloud*-a drugog provajdera imaju ekonomsku dobit ali u tom slučaju ne postoji kontrola nad podacima koji se koriste u okviru servisa. Hibridni *Cloud* je pokušaj da se iskoriste prednosti javnog *Cloud*-a ali i da se obezbedi izvestan nivo sigurnosti nad kritičnim podacima. Praktični primeri primene hibridnih *Cloud*-a u trenutku izrade rada još ne postoje, ali postoje počeci projekata takvog tipa od strane kompanija *IBM* i *Juniper*.

2.4.1 IoT i Cloud

Iako postoje mnoge tvrdnje da je IoT koncept povezan sa *Cloud* tehnologijama stvarna veza između njih ne postoji. Međutim, upotreba *Cloud*-a u okviru IoT ekosistema će svakako doneti mnoga poboljšanja, možda čak i neophodna kako bi se podržali zahtevi IoT ekosistema[35]. Skalabilna i dinamička okruženja koja obezbeđuje *Cloud* su neophodna radi izvršavanja poslužioca čiji je zadatak opsluživanje ekosistema velike gustine. Dalje, Cloud omogućuje i prilagođavanje količinama podataka u IoT ekosistemima, gde količina podataka koju je potrebno uskladištiti i procesirati može dostići izuzetno velike razmere. Kao zaključak se može navesti da ne postoji direktna veza između IoT koncepta i *Cloud* tehnologija, ali da će IoT koncept značajno profitirati upotrebom *Cloud* tehnologija.

3. Analiza problema

Na Slici 6 je prikazan implementirani IoT ekosistem. Centralna tačka prikazanog IoT ekosistema je ACS koji opslužuje klijentske uređaje IoT ekosistema upotrebom TR-069 protokola. ACS takođe obezbeđuje sprege za distribuciju prikupljenih podataka ka korisničkim aplikacijama ili uslužnim poslužiocima.



Slika 6. Arhitektura najvišeg nivoa implementiranog IoT ekosistema

U poglavlju su predstavljeni tehnički zahtevi poslužioca, kao i analiza postizanja skalabilnosti poslužioca. Tehnički zahtevi se prevashodno odnose na implementaciju TR-069 protokola kao i sprega kojima se izlažu sve funkcionalnosti poslužioca za upotrebu od strane spoljnih modula. Skalabilnost poslužioca je postignuta optimizacijom rešenja ali i kreiranjem klaster sistema i integracijom rešenja u takav sistem.

3.1 Analiza tehničkih zahteva

Komunikacija sa TR-069 klijentskim uređajima – standardizovana komunikacija upotrebom TR-069 protokola između CPE-ova i ACS-a predstavlja osnovni zahtev pri implementaciji poslužioca. Zahtev predstavlja implementaciju komunikacionog dela TR-069 specifikacije. Na ovom zahtevu počivaju svi ostali tehnički zahtevi poslužioca.

Nadgledanje rada uređaja - TR-069 protokol propisuje način dostavljanja podataka sa CPE klijentskih uređaja na ACS čime se omogućuje da ACS prati rad uređaja ili grupe uređaja i vrši analizu rada nad njima kao i da izvršava razne statističke obrade nad prikupljenim podacima. Na taj način ACS je u mogućnosti da isporuči rezultate kao što su; mapa kvaliteta signala kod uređaja digitalne televizije (eng. Set-Top Box, STB), mapa izgubljenih paketa uređaja za mrežni prolaz (eng. Gateway), istorija potrošnje električne energije ili temperatura korisničkih uređaja ili čak statistiku gledanosti televizijskog programa.

Konfigurisanje uređaje - TR-069 protokol omogućava i izvršavanje konfiguracije uređaja postavljanjem vrednosti parametara iz modela podataka. Na taj način ACS je u mogućnosti da automatski izvrši konfiguraciju uređaja bilo prilikom prijavljivanja CPE-a – inicijalna konfiguracija, bilo prilikom detektovanja problema. Pored toga ACS treba da omogući i vršenje konfiguracije na zahtev, izvršavanje ponovnog pokretanja uređaja sa ili bez vraćanja u inicijalno stanje.

Razmena datoteka sa uređajima - Otpremanje datoteka sa CPE-a je funkcionalnost koju ACS može iskoristiti radi preuzimanja datoteka koje sadrže detaljnije informacije o radu uređaja. Takođe ACS je u mogućnosti da zahteva da CPE preuzme datoteku sa zahtevane lokacije što može da se iskoristi u svrhu ažuriranja programske podrške klijentskog uređaja.

Aplikativne sprege - Nabrojani primeri primene TR-069 protokola, kao i podaci prikupljeni sa klijentskih uređaja, potrebno je distribuirati i omogućiti njihovu upotrebu u spoljnim modulima. Takva funkcionalnost se može obezbediti upotrebom aplikativne programske sprege (eng. Application Programming Interface, API). Dijagnostičke i statističke obrade nad grupom uređaja se mogu biti izvršene u okviru samog ACS-a ili u potpuno zasebnoj

komponenti. Upotrebom API-ja sve funkcionalnosti ACS-a se mogu izložiti ka spoljnim komponentama i time se ACS oslobađa dela posla i smanjuje se opterećenje ACS-a.

Raspoređivanje opterećenja ACS-a upotrebom TR-069 funkcionalnosti - TR-069 protokol je osmišljen sa konceptom da funkcioniše u velikim mrežama sa velikim brojem klijentskim uređaja. Uobičajena je situacija u kojoj se CPE periodično prijavljuje ACS-u pri čemu se perioda prijavljivanja može dinamički menjati. Perioda prijavljivanja može biti izuzetno retka kao što je jednom dnevno, ili veoma česta – jednom u minutu. Perioda zavisi od prirode uređaja ali i od veličine mreže gde je potrebno uskladiti potrebu za čestim prijavljivanjem uređaja – ažurno nadgledanje rada uređaja, sa veličinom mreže, odnosno brojem uređaja koji se nadgledaju u ekosistemu. Jedan od uobičajenih načina nadgledanja uređaja u realnom ekosistemu je sa periodom prijavljivanja od dva puta u danu, odnosno na svakih 12 sati, pri čemu se prosleđuju podaci koji su definisani sa aktivnom politikom praćenja. U okviru modela podataka mogu postojati parametri čija se vrednost može vremenom proširivati ili akumulirati čime se može smanjiti opterećenje na mrežu i na ACS.

Broj uređaja koje jedan ACS poslužilac treba da opslužuje u realnom ekosistemu se meri u desetinama ili stotinama hiljada uređaja. Potrebno je napomenuti da broj uređaja svakako zavisi i od performansi fizičke arhitekture poslužioca. Prema tome u slučaju jačih personalnih računara (eng. Personal Computer, PC) broj uređaja koje podržava takav ACS trebalo bi da bude približno 100 000 uređaja sa periodom prijavljivanja od 12 sati.

Usled broja uređaja u IoT ekosistemima koji može dostići i brojke i do milion uređaja jasno je da se sa jednom računarskom jedinicom ne može postići opsluživanje celog ekosistema. Iz tog razloga je potrebno implementirati ACS koji je sposoban za rad u klasteru. I dodatno, potrebno je stvoriti klaster okruženje u kojem bi se tako implementirani ACS izvršavao pravilno i čime bi se postigao visok nivo skalabilnosti ACS-a.

3.2 Analiza problema optimizacije poslužioca

Optimizacija poslužioca, ne samo u slučaju ACS implementacije, je od izuzetnog značaja kako bi se postigle maksimalne performanse fizičke arhitekture poslužioca i kako bi se se opslužio maksimalan broj klijenata. Usled zahteva da poslužilac opslužuje veliki broj uređaja potrebno je izvršiti analizu arhitekture poslužioca i ustanoviti koji moduli i koji scenariji upotrebe poslužioca iziskuju najviše vremena i najviše resursa za obradu. Poslužioc-klijent arhitektura se zasniva na principu upit/odgovor (eng. Request/Response), odnosno poslužilac prima upit od klijenta, obrađuje zahtev i prosleđuje odgovor. Prema tome poslužilac u ostalim trenucima, kada ne obrađuje zahtev, treba da je u stanju mirovanja. Specijalni su slučajevi kada

poslužilac ima potrebu za eventualnom periodičnom obradom podataka, ali bilo kakvo opterećivanje poslužioca van obrade pristiglih upita je potrebno izbegavati. Smanjivanjem dodatne obrade kao i brzine obrade upita predstavlja primarni zadatak u optimizaciji poslužioca.

Svakako najveći problem pri realizaciji poslužioca predstavlja upotreba memorije slučajnog pristupa (eng. Random Access Memory, RAM). Iz tog razloga je potrebno ustanoviti koji podaci se mogu upisati u trajnu memoriju, odnosno koji podaci se trebaju sačuvati a koji nisu od koristi. Radi trajnog čuvanja podataka u poslužiocima najčešće se koriste relacione baze podataka. U najvećem broju poslužilaca, upravo baza podataka predstavlja usko grlo sistema zbog sporosti operacija upisa i čitanja podataka iz stalne memorije. Ukoliko se baza podataka distribuira na više računarskih jedinica, operacije nad bazom podataka se usporavaju i tako pogoršavaju performanse poslužioca. Zbog toga je ključno smanjiti broj operacija nad bazom podataka na što manji broj kao i smanjiti količinu podataka koje se prosleđuju bazi podataka.

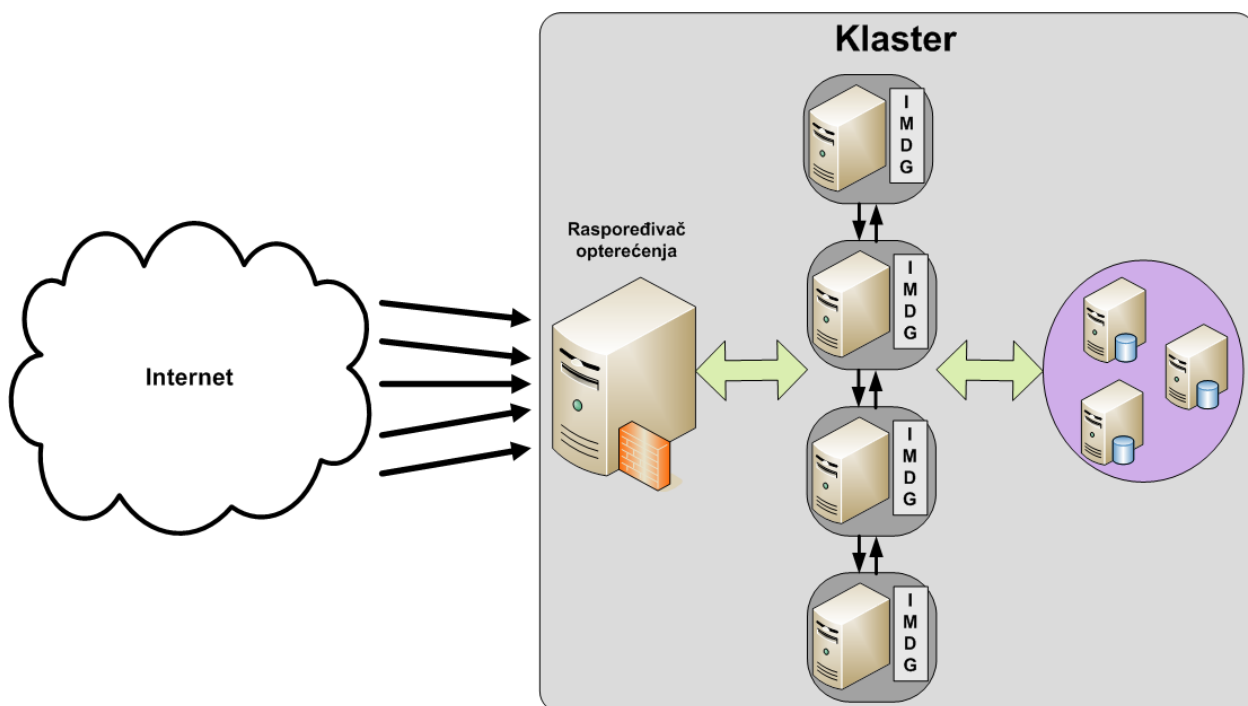
Upotreba mreže podataka koja se čuva u memoriji (eng. In-Memory Data Grid, IMDG) je sve češća praksa u implementaciji poslužioca. IMDG je struktura podataka koja se u potpunosti smešta u RAM memoriju i može se distribuirati između više poslužilaca. IMDG je namenjena za rukovanje velikom količinom podacima (eng. Big Data) i obezbeđuje brzinu, varijabilnost i obim rukovanja velikom količinom podataka. IMDG podržava hiljade operacija u sekundi nad podacima, i podaci mogu biti grupisani i smanjeni u toj meri da se mogu podržavati velike količine podataka. Međutim, upotreba IMDG može biti doneti i pogoršanja, jer usled nepravilne upotrebe može doći do prevelike i nepotrebne iskorišćenosti RAM memorije što može dovesti do potpunog prestanka rada poslužioca. Iz tog razloga je potrebno posebno analizirati upotrebu i rad IMDG struktura podataka kako bi se izbegao nepravilan rad poslužioca.

Nakon detaljne analize rada poslužioca ustanovljeno je da treba izvršiti sledeće tehnike optimizacije:

- **Optimizovati upotrebu IMDG struktura podataka** – potrebno je smanjiti količinu podataka koja se skladišti u njima i ukloniti sve redundantne podatke. Takođe primenom ugrađenih mehanizama IMDG struktura može se znatno smanjiti opterećenje jezgra ACS-a i ograničiti upotreba RAM memorije.
- **Optimizovati upotrebu baze podataka** – takođe je potrebno smanjiti količinu podataka kao i ukloniti redundantne podatke. Pored toga neophodno je smanjiti broj transakcija na bazom podataka primenom kompleksnih upita.

3.3 Analiza problema kreiranja klastera

Upotreba distribuirane obrade je neophodna u današnjim poslužiocima usled obima mreže koje je potrebno opsluživati. I pored toga što distribuirana obrada unosi kašnjenja u obradi zahteva usled međusobne komunikacije između čvorova upotreba klastera je veoma privlačna jer stvara iluziju postojanja jedne računarske jedinice dok njihov broj daleko veći. Klaster naravno omogućava i poboljšavanje performansi, odnosno omogućava obradu i skladištenje većih količina podataka. Međutim, implementacija klastera nije jednostavna jer je da bi se postigao pun kapacitet klastera potrebno distribuirati rad svih komponenti od kojih se poslužilac sastoji. Na Slici 7 su prikazane komponente klastera prikazanog u ovome radu. Broj i organizacija komponenti klastera mogu varirati u zavisnosti od potrebe poslužioca.



Slika 7. Komponente implementiranog klastera

Prva komponenta klastera je *Raspoređivač opterećenja* čiji je zadatak da pristigle zahteve ravnomerno prosleđuje čvorovima u klasteru. Jasno je da raspoređivač predstavlja kritičnu tačku u sistemu jer ukoliko dođe do njegovog otkaza klaster neće biti vidljiv u ekosistemu. Zato je potrebno postojanje pozadinskog raspoređivača koji će periodično proveravati stanje primarnog raspoređivača i ukoliko dođe do njegovog otkaza preuzeti ulogu primarnog raspoređivača opterećenja. Programsku podršku poslužioca koja predstavlja implementaciju poslužioca je takođe potrebno distribuirati i usaglasiti njen rad. Pored toga što je svaki poslužilac je sposoban za samostalni rad, podatke koji se interno čuvaju na njemu je potrebno distribuirati kako bi svaki poslužilac u klasteru mogao da ima pristup svim podacima. Iz tog razloga je potrebno da se IMDG distribuira, kao i baza podataka. Međutim kao što se sa Slika 6 i vidi, baza podataka je

izmeštena kao posebna komponenta klastera kojoj poslužiocu udaljeno pristupaju da bi se smanjilo opterećenje poslužioca. Baza podataka je takođe distribuirana kako ne bi predstavljala tačku otkaza u klasteru i kako bi se povećao memorijski kapacitet.

Da bi programska podrška poslužioca radila pravilno u klasteru potrebno je poštovati pravila definisana za biznis logiku poslužioca poznatu kao **Enterprise JavaBeans** (EJB) ograničenja. Ograničenja se prevashodno odnose na nezavisnost programske podrške od platforme računarskog sistema, a neke od najvažnijih su:

- Čitanje i pisanje statičke promenljive koje nemaju konačnu vrednost.
- Kreiranje grafičkih komponenti u Java okruženju.
- Preusmeravanje ulaznih, izlaznih tokova ili tokova greške.
- Kreiranje i izmene niti.
- Upotreba primitiva sinhronizacije niti kako bi se sinhronizovao pristup EJB objektima.
- Zaustavljanje Java virtualne mašine.
- Direktno čitanje ili pisanje deskriptora datoteka.
- Kreiranje, izmena ili brisanje datoteka iz sistema datoteka.

Poštovanje EJB ograničenja je neophodno kako bi se obezbedio pravilan rad klastera i izbeglo nepredviđeno ponašanje biznis logike poslužioca. Promene neophodne da se izvrše u okviru postojećeg rešenja kako bi se omogućio njegov rad u klasteru su sledeća:

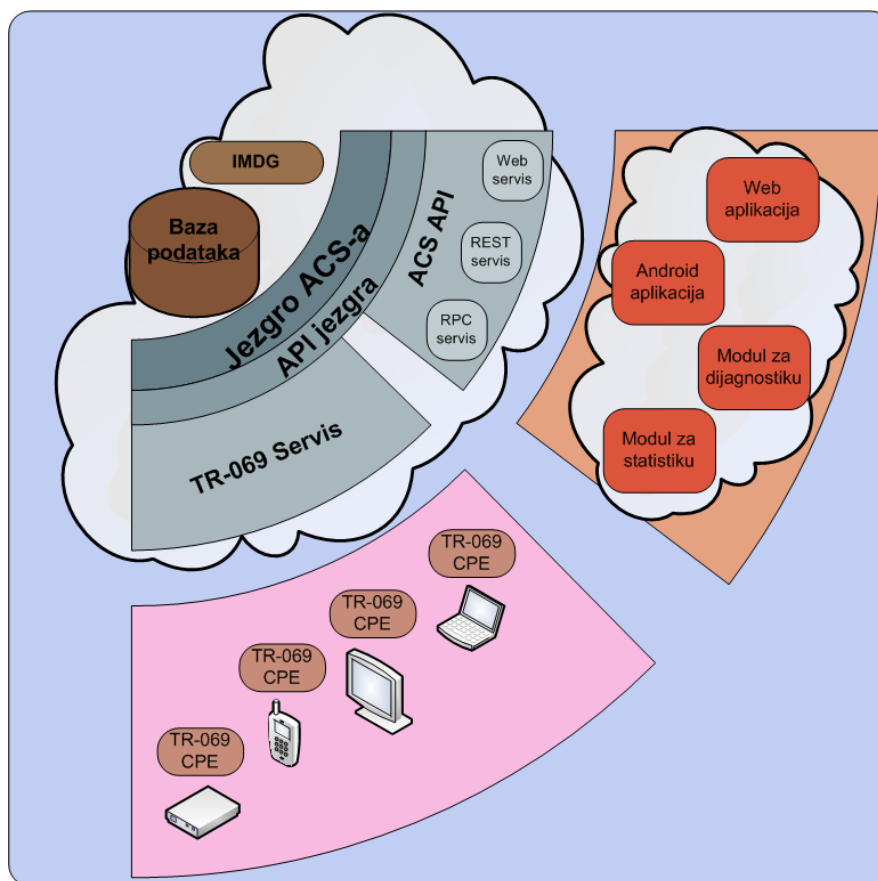
1. Upotreba sistema datoteka.
2. Upotreba EJB vremenskih okidača.
3. Upotreba statičkih promenljivih koje nemaju konačnu vrednost.

4. Koncept rešenja

U narednom poglavlju predstavljena je implementacija poslužioca koji predstavlja realizaciju TR-069 protokola i koji je sposoban za rad u klasteru. Pored toga prikazani su i načini optimizacije poslužioca zajedno sa načinom stvaranja klaster okruženja u kojem je implementirani poslužilac sposoban da funkcioniše na pravilan način. Optimizacija poslužioca je usmerena na upotrebu eksternih resursa kao što su IMDG strukture podataka i baze podataka. Okruženje koje se sastoji od više eksternih resursa je potrebno klasterizovati na svakom nivou, odnosno potrebno je klasterizovati svaki resurs ali na takav način da svi resursi budu sinhronizovani međusobno ali i sa programskom podrškom poslužioca. Takođe, potrebno je programsku podršku prilagoditi da pravilno radi u takvom klaster okruženju.

4.1 Implementacija TR-069 protokola

Funkcionalnosti koje omogućava TR-069 protokol i koje su opisane u prethodnom poglavlju predstavljaju prvi i osnovni zahtev u implementaciji poslužioc. Takođe, poslužilac izlaže sve funkcionalnosti koje su implementirane upotrebom sprega. Poslužilac sadrži i mehanizme kojima je poslužilac u mogućnosti da detektuje potencijalne probleme i načine njihovih rešavanja. Osnovni moduli od kojih se sastoji implementirano rešenje ACS-a koje je u *Cloud* okruženju prikazano je na Slici 8.



Slika 8. Arhitektura najvišeg nivoa implementiranog ACS-a

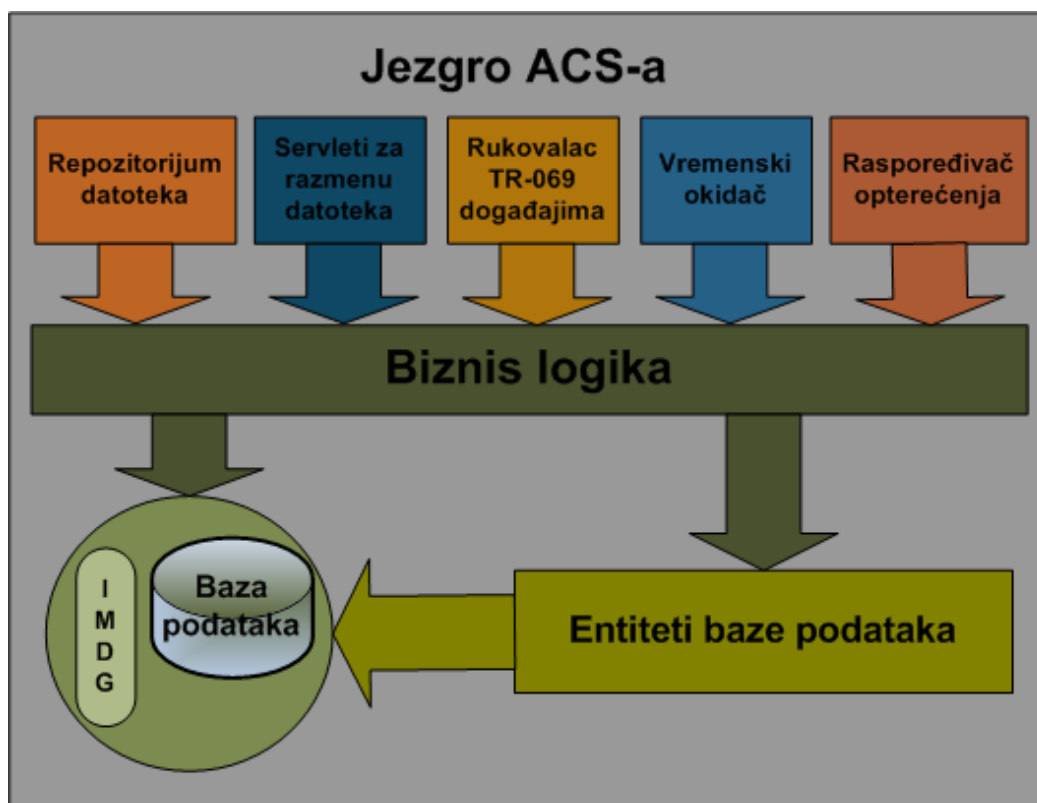
Zahtevi prikazani u okviru poglavlja 3.1 su implementirani u različitim modulima ACS-a. U narednom tekstu su opisani moduli i njihova primena u tehničkim zahtevima ACS-a.

Komunikacija sa TR-069 klijentskim uređajima - *TR-069 servis* predstavlja komunikacioni sloj kojim se obezbeđuje standardizovana komunikacija sa CPE klijentima. Servis sadrži SOAP parser koji služi za parsiranje pristiglih HTTP zahteva prema XSD šemi propisane u okviru TR-106 specifikacije.

Aplikativne sprege - *ACS API* modul se sastoji od više API servisa koji su implementirani kao Web servis – Web Service Description Language (WSDL), kao Representational State Transfer servis (REST) ili kao servis udaljenih proceduralnih poziva (eng. Remote Procedure Call, RPC). REST servis koristi REST zasnovanu arhitekturu kojom se prosleđuju JavaScript Object Notation (JSON) poruke. *ACS API* modul omogućava da se funkcionalnosti jezgra ACS-a mogu izložiti na korišćenje eksternim modulima ili aplikacijama. Posebna pažnja je usmerena ka obezbeđivanju sigurnosti podataka. Svaki od servisa iz *ACS API*-ja koristi poruke šifrovane upotrebom HTTPS protokola i svaka komunikacija je sprovedena upotrebom potvrde identiteta i kreiranjem sesija sa ograničenim vremenom upotrebe. Potvrdom identiteta se omogućava kontrolisanje prava pristupa podacima i time je ACS u mogućnosti da

kontrolira distribuciju podataka, odnosno da ograniči koji korisnici mogu da pristupaju kojim podacima. Nabrojanim merama sigurnosti ACS omogućuje da se eksterni moduli mogu smestiti u sopstvena *Cloud* okruženja.

Ostali tehnički zahtevi implementirani su u okviru *jezgra ACS-a* koji je prikazan na Slici 9. U daljem tekstu su ukratko objašnjene funkcionalnosti svakog modula sa naglašenim tehničkim zahtevima koje određeni modul ispunjava.



Slika 9. Arhitektura jezgra ACS-a

Jezgro ACS-a predstavlja najvažniji i najopširniji modul u kojem je implementirana biznis logika ACS-a. *Biznis logika* obezbeđuje rukovanje nad bazom podataka i IMDG strukturama podataka. Takođe, modul obezbeđuje i automatsku **konfiguraciju** uređaja koja zavisi od tipa uređaja, ali i **nagledanje** i detektovanje problema pri radu CPE uređaja.

Repozitorijum datoteka je modul koji je implementiran upotrebom Java Content Repository (JCR) sprege. Modul omogućava funkcionalnosti skladištenja meta podataka, podataka kao i sadržaja datoteka unutar repozitorijuma. Sadržaji datoteka se mogu čuvati u sistemu datoteka ili u okviru baze podataka.

Servleti za razmenu datoteka obezbeđuje **razmenu datoteka** između ACS-a i CPE-a. ACS je u mogućnosti da, upotrebom TR-069 RPC metoda, prosleđuje zahteve za preuzimanje ili otpremanje datoteka CPE-ovima sa navedenim jednoobraznim lokatorom resursa (eng Uniform Resource Locator, URL) koji definiše izvorišnu ili odredišnu lokaciju datoteke.

Modul za rukovanje događajima rukuje TR-069 događajima kao što su pokretanje CPE-a sa ili bez vraćanja u inicijalno stanje, završen prenos datoteke, registrovanje potencijalnih problema ili CPE prosledi odgovori sa greškom poruke. Modul je u mogućnosti da detektuje takve događaje i sprovede odgovarajuće akcije u zavisnosti od tipa događaja

Periodične funkcionalnosti jezgra ACS-a su izvršene u okviru modula *Vremenskog okidača*. Primeri periodičnih funkcionalnosti su provera statusa CPE-ova, zahtevanje datoteka koje sadrže detaljnije informacije o radu uređaja ili izvršavanja raznih statističkih procesa i analize rada uređaja.

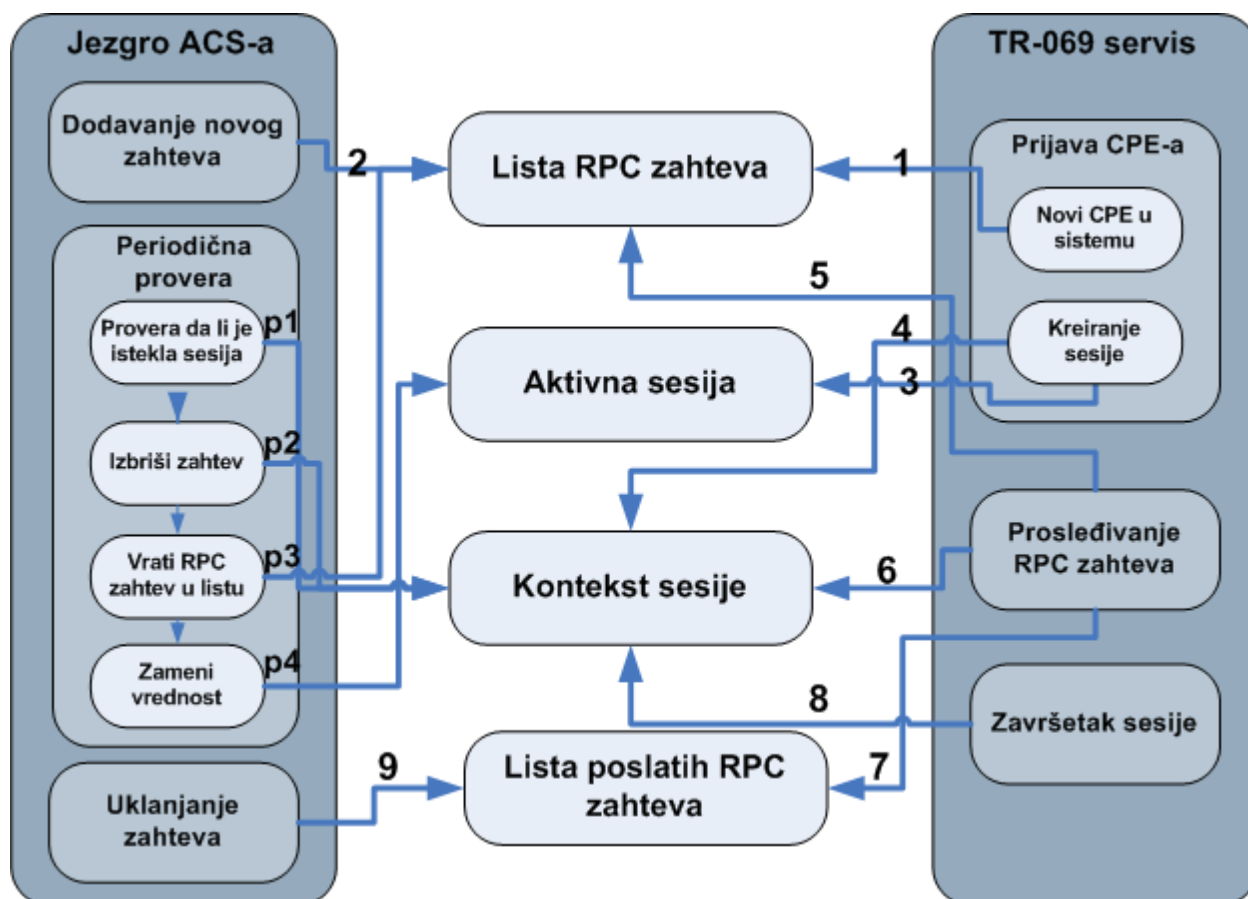
Raspoređivač opterećenja je modul koji prati opterećenje ACS-a kako bi mogao da se prilagodi broju CPE uređaja. Modul obezbeđuje **raspoređivanje opterećenja** upotrebom periode prijavljivanja CPE-ova čiju vrednost smanjuje ili povećava u zavisnosti od opterećenja ACS-a.

4.2 Optimizacija implementiranog poslužioca

Kao što je pomenuto u poglavlju o analizi problema, prilikom optimizacije najbitnije je da se prvo uradi detaljna analiza kako bi se ustanovilo koji scenariji upotrebe poslužioca iziskuju najviše vremena i najviše resursa za obradu. Analizom postojećeg rešenja zaključeno je da se najviše računarskog resursa troši prilikom učestalog prijavljivanja CPE-ova na ACS, odnosno što su učestalije prijave to je potrošnja resursa veća. Jedan od načina poboljšavanja upotrebe resursa jeste i smanjivanje broja prijava što je moguće povećavanjem perioda prijavljivanja na CPE-ovima. Drugi način je da se optimizuje ACS. Daljom analizom, najviše praćenjem korišćenja memorije, ustanovljeno je da postojeća upotreba IMDG-a zahteva mnogo resursa.

4.2.1 Optimizacija upotrebe IMDG struktura podataka

Kako je implementirani način upotrebe IMDG struktura pokazan kao veoma neoptimalan bilo je potrebno da se osmisli novi dizajn upotrebe IMDG-a. Na Slici 10 je prikazani neoptimalni dizajn upotrebe IMDG strukture podataka.



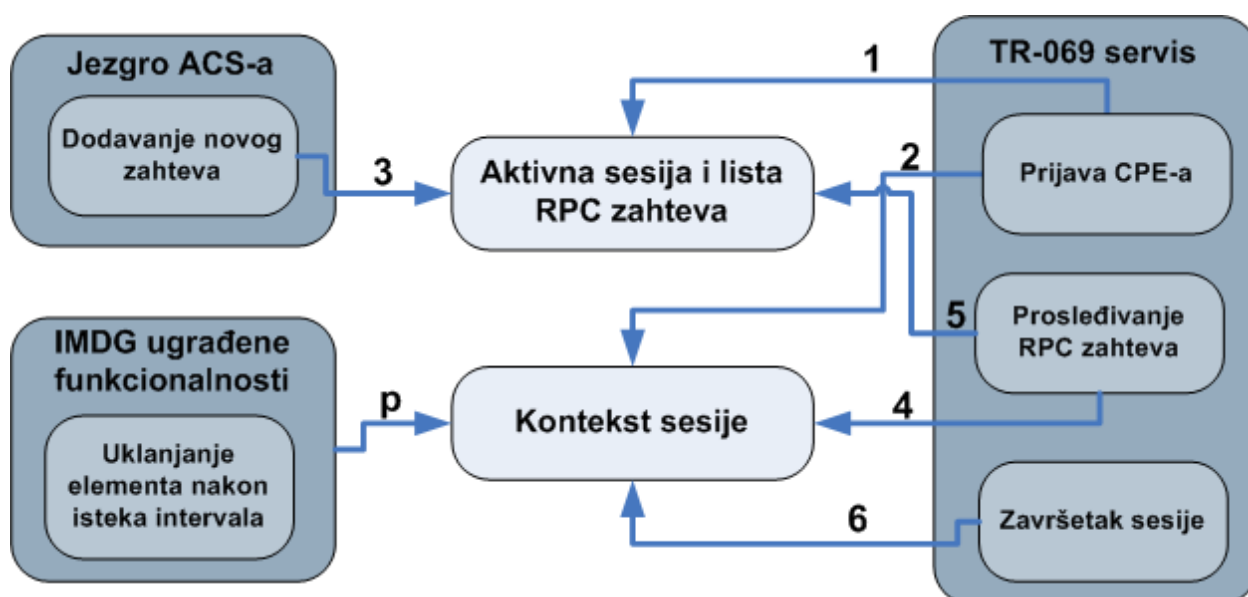
Slika 10. Neoptimalni dizajn upotrebe IMDG struktura 1 - Dodaje se element 2 - Dodaju se elementi 3 – Dodaje se element 4 - Dodaje se element 5 - Čita se i uklanja element 6 - Promena elementa 7 - Dodaje se element 8 - Uklanja se 9 - Uklanja se element p1 – Čitaju se elementi p2 – Uklanjaju se elementi p3 – Dodaje se elementi p4 – Menjaju se elementi

Scenario neoptimalne upotrebe IMDG struktura prikazan na slici se sastoji iz periodične provere u koracima p1-p2-p3-p4 prikazanim na slici ali i iz sledećih koraka:

1. Pri prijavljivanju novog CPE uređaja u ekosistem potrebno dodati novi element koji sadrži listu RPC zahteva za taj CPE.
2. Pri automatskoj ili manualnoj konfiguraciji koju izvršava *Jezgro ACS-a* RPC zahtevi se dodaju u listu RPC zahteva za određeni CPE.
3. Pri svakoj prijavi CPE-a na ACS dodaje se element koji sadrži informaciju koja je trenutna sesija.
4. Takođe se pri svakoj prijavi CPE-a dodaje i novi element sa informacijama o trenutnoj sesiji, kao što je poslednje prosleđeni RPC zahtev ili vreme kada je CPE prosledio poslednju poruku.
5. Kada je sesija započeta vrši se provera da li postoji RPC zahtev za određeni CPE i ukoliko postoji uklanja se prvi iz liste RPC zahteva.
6. Zatim se pročitani RPC zahtev unosi u kontekst sesije kao poslednja poslata poruka ka CPE-u.

7. Zatim se isti element dodaje u okviru liste poslatih RPC zahteva koji se koristi pri obradi CPE odgovora na taj RPC zahtev od strane *Jezgra ACS-a*.
8. Kada se završi trenutna sesija njen kontekst se uklanja iz IMDG strukture.
9. Nakon obrade CPE odgovora na RPC zahtev, zahtev uklanja se iz strukture.

Najveća mana prikazane upotrebe IMDG tehnologije je postojanje redundantnih podataka i zanemarivanje ugrađenih funkcionalnosti IMDG mehanizma čijom upotrebom se mogu postići značajna smanjenja potrošnje resursa. Na Slici 11 prikazan je novi, optimizovan dizajn upotrebe IMDG tehnologije.



Slika 11. Optimizovan dizajn upotrebe IMDG struktura 1 – Dodaje se ili menja element 2 – Dodaje se element 3 – Dodaje se element 4 – Proverava se element 5 – Proverava se ili uklanja element 6 – Uklanja se element p – Proveravaju se i uklanjaju elementi

Pored upotrebe IMDG ugrađene funkcionalnost koja se koristi radi provere da li je neki element predugo u stanju mirovanja, scenario optimizovane upotrebe IMDG struktura se sastoji iz sledećih koraka:

1. Pri prijavljivanju CPE uređaja potrebno dodati novi element ili promeniti vrednost postojećeg elementa koji sadrži listu RPC zahteva za taj CPE kao i identifikacionu reč trenutne sesije.
2. Pri svakoj prijavi CPE-a na ACS dodaje se element sa identifikacionom reči te sesije.
3. Pri automatskoj ili manuelnoj konfiguraciji koju izvršava *Jezgro ACS-a* RPC zahtevi se dodaju u listu RPC zahteva za određeni CPE.

4. Proverava se da li je pristigli odgovor od CPE-a u toku validne sesije.
5. Kada je sesija započeta vrši se provera da li postoji RPC zahtev za određeni CPE i ukoliko postoji prosleđuje se prvi iz liste RPC zahteva. Ukoliko je u toku sesije CPE prosledio odgovor na RPC zahtev u tom slučaju se najpre ukloni prvi iz liste RPC zahteva a zatim, ukoliko postoji, prosledi naredni RPC zahtev iz liste.
6. Na kraju sesije se uklanja kontekst te sesije.

Optimizacija je postignuta smanjenjem količine podataka koja se čuva u IMDG-u kao i upotrebom ugrađenih funkcionalnosti kao što su uklanjanje elementa iz IMDG-a nakon isteka zadatog vremenskog intervala ili ograničavanjem broja elemenata koji su čuvaju memoriji. Broj upita koji se vrše nad IMDG-u je takođe smanjen. Posebna pažnja je usmerena ka vrstama upita. Pošto je IMDG organizovana kao ključ/vrednost struktura podataka, dizajn je osmišljen na taj način da se vrše samo upiti na osnovu ključeva kako bi se izbegli upiti za dobavljanjem cele IMDG strukture. Takva vrsta upita je i predstavljala najveći problem u starom dizajnu upotrebe IMDG.

Usled pojave specifičnih funkcionalnosti u okviru implementacije ACS-a stvorila se potreba za vršenjem upita nad IMDG koji nisu preporučljivi za upotrebu ili nisu podržani od strane korišćene IMDG tehnologije. Međutim, kako je potreba za takvim funkcionalnostima neophodna i njihova upotreba nije česta, što dalje znači da se poslužilac ne opterećuje u primetnijoj meri, za njihovu implementaciju korišćeni su posebni mehanizmi. Jedan od takvih mehanizama je i MapReduce mehanizam [36]. MapReduce mehanizam je programski model za procesiranje i generisanje velikih količina podataka u okviru paralelnih ili distribuiranih sistema. Mehanizam propisuje dve najvažnije funkcionalnosti:

- **Map** – obezbeđuje procesiranje ključ/vrednost parova podataka i generisanje novih ključ/vrednost parova.
- **Reduce** – obezbeđuje operacije sažimanja podataka, odnosno spaja sve nove vrednosti pridružene istim novo generisanim ključevima.

4.2.2 Optimizacija upotrebe baze podataka

Baza podataka i njeno rukovanje predstavlja najveći zadatak sa kojima se suočavaju svi tipovi poslužioca. Baza podataka se čuva u trajnoj memoriji, u kojoj operacija upisa traje vremenski duže od operacije čitanja. Broj operacija nad bazom podataka je potrebno smanjiti koliko god je moguće. Dodatni problem u implementaciji ACS-a predstavlja čest upis podataka u bazu, dok je čitanje podataka manje zastupljeno u ovom slučaju. I pored postojanja mehanizama

koji se ponašaju kao brza memorija baze podataka, odnosno mehanizama gde se deo podataka koji se često čitaju iz baze podataka čuvaju u RAM memoriji, takvi mehanizmi su neupotrebljivi u implementaciji ACS-a. Problem predstavlja upravo čest upis podataka u bazu podataka. U slučaju opisanih mehanizama potrebno je ažurirati podatke i u brznoj memoriji i u bazi podataka, što uvodi dodatno kašnjenje.

Broj transakcija nad bazom podataka je neophodno smanjiti. Provere nad tabelom baze podataka se često vrše sekvencijalno. Takav pristup podrazumeva prolazak kroz svaki red tabele i obavljanje transakcije nad svakim redom ponaosob. Ovo je loša praksa zbog opterećenja baze podataka, te se preporučuje da se sve provere smeste u okviru jedne transakcije. Za izvršavanje takvih transakcija potrebna je upotreba kompleksnih Structured Query Language (SQL) upita. Kompleksni SQL upiti mogu da sadrže druge upite, povezuju se upiti nad više različitih tabela, ali i da se koriste petlje i uslovne grane. Time se omogućava identična funkcionalnost kao kod programskih jezika. Njihova upotreba je neophodna kako bi se smanjio broj transakcija nad bazom podataka. Dodatno, potrebno je obratiti pažnju na podatke koji su suvišni, kao i na prepisivanje već postojećih podataka. Upotrebom mehanizma indeksiranja baze podataka se postiže ubrzavanje izvršavanja operacija nad bazom podataka. Označavanjem kolona tabela baze podataka upotrebom indeksa, baza podataka grupiše redove tabela prema njima i time se postižu ubrzanja pri pretrazi tabela.

4.3 Implementacija klastera

Implementacija klaster sistema se sastoji iz dva dela, gde se prvi deo odnosi na klasterovanje eksternih resursa koje programska podrška poslužioca koristi a drugi deo na programsku podršku na kojoj je potrebno izvršiti određene izmene kako bi pravilno radila u klaster okruženju. U eksterne resurse koji stvaraju klaster okruženje spadaju baza podataka, IMDG, aplikativni poslužilac i raspoređivač opterećenja.

4.3.1 Klasterovanje eksternih resursa

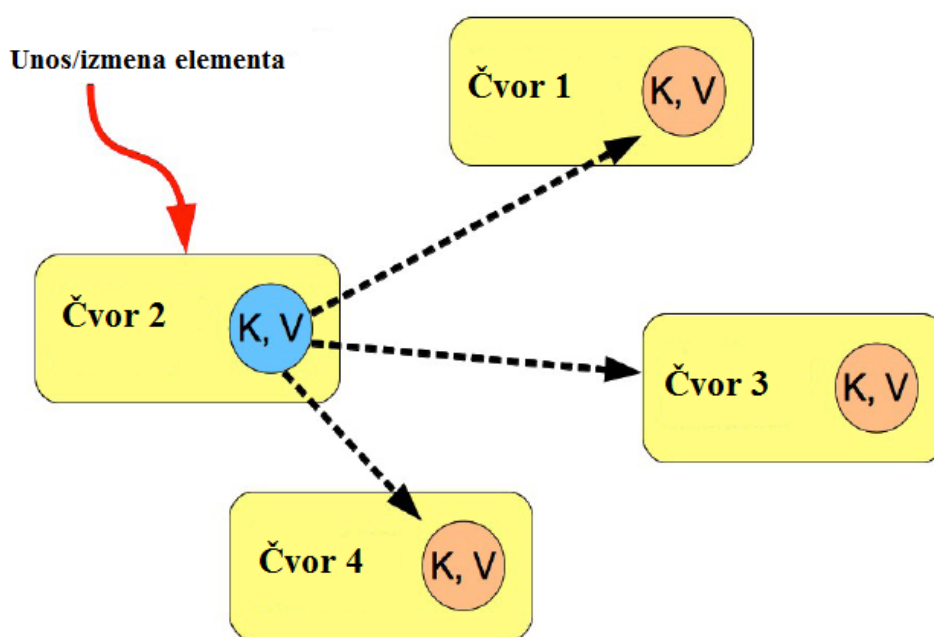
Bazu podataka je potrebno klasterovati kako bi se obezbedili veći memorijski kapaciteti ali i kako baza podataka na jednom čvoru ne bi predstavljala tačku otkaza celog sistema. Pri implementaciji ACS-a korišćen je poslužilac baze podataka PostgreSQL [37]. Međutim, kako je klasterovanje PostgreSQL izuzetno teško ili gotovo nemoguće jer ne postoji zvaničan i validan mehanizam, klasterovanje baze podataka je zanemareno u ovoj implementaciji. Kako bi se omogućio pravilan rad klastera koristi se baza podataka na posebnoj radnoj stanici koju dele svi čvorovi u sistemu. Takva realizacija unosi ograničenja primene klastera zbog čega će dalja

unapređenja rada najpre odnositi na analizu postojećih rešenja baza podataka koja bi donela poboljšanja ali i omogućili klasterizaciju baze podataka kako ne bi predstavljala usko grlo sistema.

4.3.1.1 Klasterovanje Infinispan-a

Kao IMDG u implementiranom ACS-u korišćen je Infinispan [38]. Rad Infinispan-a je takođe potrebno klasterizovati. Postoji više režima u kojima IMDG može raditi u klasteru:

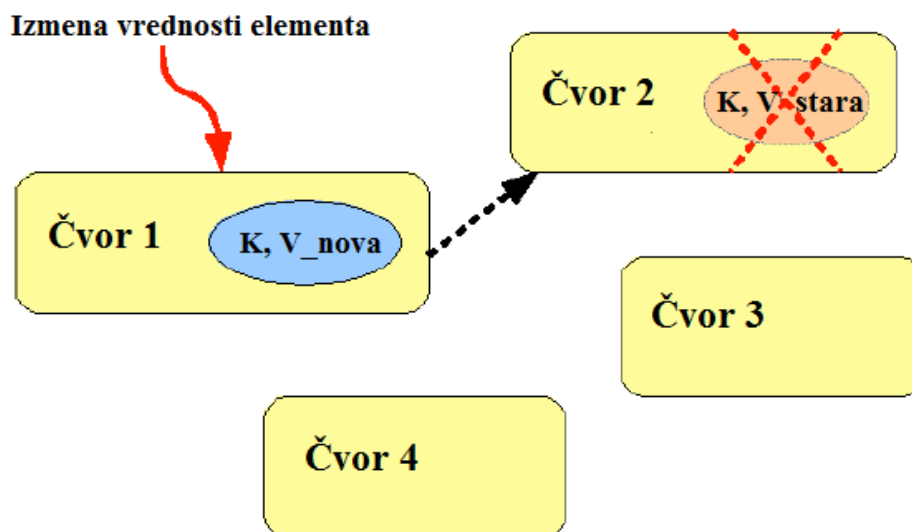
- **Režim sa kopiranjem sadržaja** – svaki čvor u okviru klastera IMDG razotkriva susedne čvorove i sa njima zajedno formira klaster. Kada se doda element u IMDG ostali čvorovi se obaveštavaju o promeni i sami primenjuju promenu, kao što se može videti na Slici 12. Potom svaki čvor može lokalno pristupiti svakom elementu IMDG-a. Režim sa kopiranjem sadržaja obezbeđuje brz i efikasan način za deljenje stanja u okviru klastera, međutim u većim klaster sistemima ovaj režim nije prihvatljiv usled povećavanja broja poruka koji je potrebno razmeniti kako bi se usaglasilo stanje svih čvorova.



Slika 12. Režim sa kopiranjem sadržaja u Infinispan-u

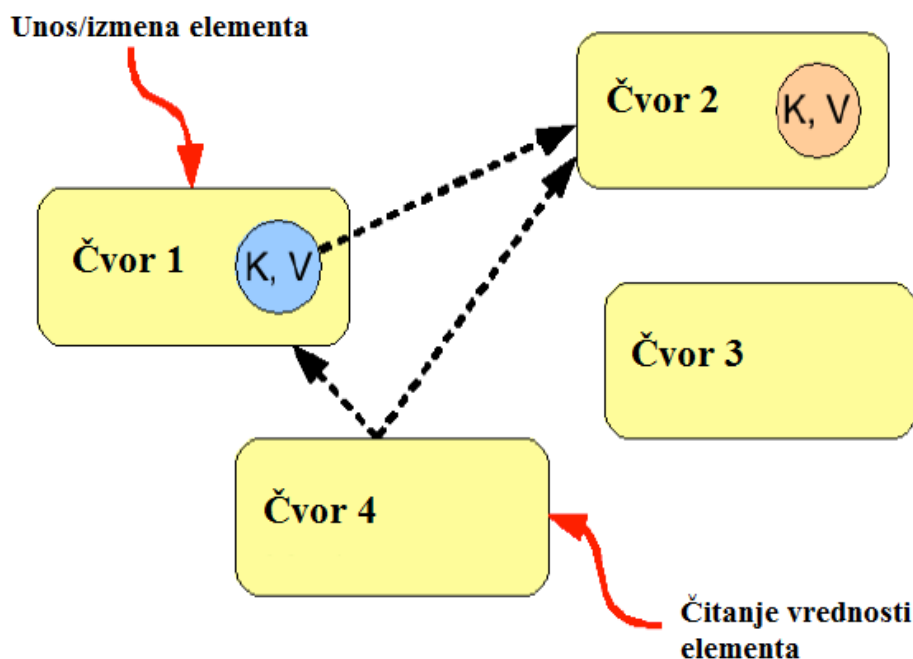
- **Režim sa brisanjem sadržaja** – u ovom režimu čvorovi IMDG-a ne dele ništa već se podaci brišu ukoliko nisu ažurni kao što je prikazano na Slici 13. Režim sa brisanjem sadržaja je poželjan za upotrebu ukoliko postoji dodatni mehanizam za čuvanje podataka dok se IMDG u ovom režimu koristi samo kao metoda optimizacije, odnosno kao brza memorija u kojoj se podaci smeštaju privremeno. U

ovom režimu saobraćaj između čvorova je smanjen u velikoj meri u odnosu na režim sa kopiranjem podataka.



Slika 13. Režim sa brisanjem sadržaja u Infinispan-u

- **Režim distribuiranja sadržaja** – u ovom režimu podaci se mogu distribuirati na proizvoljan broj čvorova kao što je prikazano na Slici 14. Prema tome se isti podaci čuvaju na nekoliko čvorova ali ne na svim čvorovima. Time se postiže da se podaci ne izgube u slučaju otkaza čvora koji sadrži te podatke, dok se saobraćaj u okviru klastera se smanjuje u odnosu na režim sa kopiranjem sadržaja. Ukoliko se zatraži vrednost elementa koji se ne nalazi na lokalnoj verziji čvor kontaktira ostale čvorove u klasteru radi pronalaženja zatraženog elementa. Određivanje koji podaci se čuvaju na kojim čvorima se postiže upotrebom *Hash* algoritama.



Slika 14. Režim sa distribuiranjem sadržaja u Infinispan-u

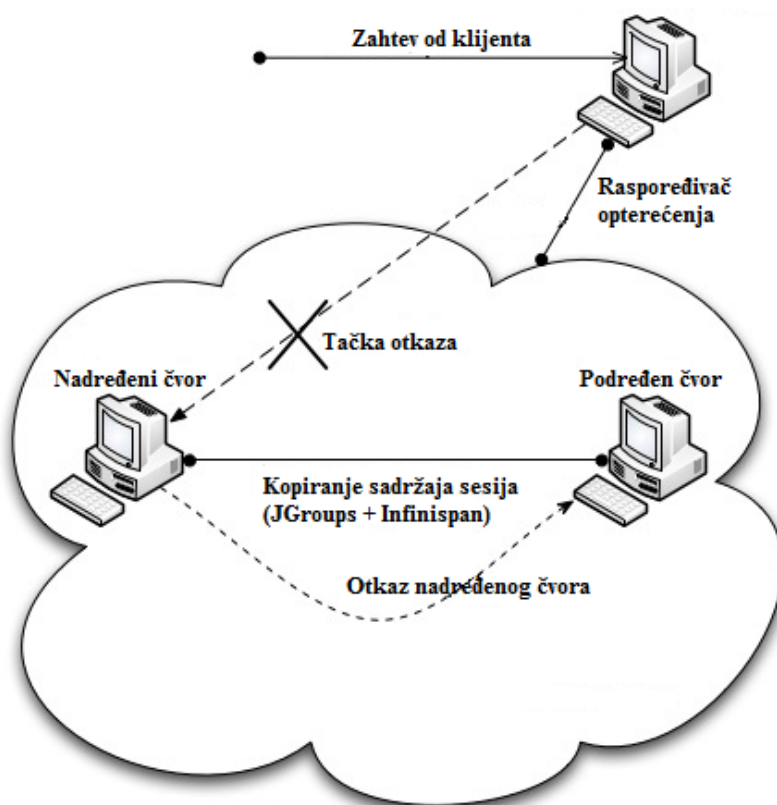
Svaki od nabrojanih režima deljenja podataka u IMDG-u mogu da rade u sinhronom ili asinhronom režimu. U sinhronom režimu operacije promene sadržaja IMDG-a blokiraju korisnika do trenutka kada je primena promena sigurno izvršena na svim čvorima. U asinhronom režimu operacije ne blokiraju korisnika već se sve promene smeštaju u red čekanja i kasnije upotrebom IMDG asinhronih niti bivaju primenjene na IMDG strukture. Za potrebe implementacije ACS-a korišćen je asinhroni režim distribuiranja sadržaja. Razlog za upotrebu ovog režima je iz razloga što se podaci čuvaju jedino u IMDG-u a upotreba režima sa kopiranjem sadržaja unosi preveliko opterećenje u klasteru usled čestog menjanja podataka u IMDG-u. Jedina mana ovog režima je u mogućnosti pojave nekonzistentnih podataka u kratkom vremenskom intervalu dok se promene iz reda čekanja ne primene na sve potrebne čvorove. Međutim, kako kritičnost tih podataka nije velika takav scenario ne pravi veliku štetu u poslužiocu i može se zanemariti.

4.3.1.2 Klasterovanje JBoss aplikativnog poslužioca

Aplikativni poslužilac obezbeđuje standardne funkcionalnosti poslužioca na kome se mogu izvršavati razvojne aplikacije. Programska podrška implementiranog ACS-a se pokreće u okviru jednog takvog aplikativnog poslužioca. Kao aplikativni poslužilac se koristi JavaBeans Open Source Software Application Server 7.1 (JBoss AS) poslužilac [39]. JBoss u klasteru radi po principu nadređen/podređen mehanizma, gde se nadređen čvor ponaša kao rukovodeći čvor. Ukoliko dođe do otkaza nadređenog čvora njegovu ulogu preuzima jedan od podređenih čvorova

što se može videti na Slici 15. JBoss koristi JGroups podsistem [40] koji obezbeđuje podršku za komunikaciju u okviru grupe za HA servise, odnosno dostupnosti svim čvorovima u klasteru u formi Jgroups kanala. JBoss koristi i Infinispan kao IMDG podsistem za skladištenje podataka vezanih za HA servise čime se postižu visoke performanse i mogućnost oporavka od grešaka u slučaju prestanka rada nekog čvora. Potrebno je da se JBOSS pokrene u režimu klastera kako bi se omogućile sledeće funkcionalnosti:

- Automatsko registrovanje čvorova u klasteru.
- Kopiranje Java Naming and Directory Interface (JNDI) konteksta unutar klastera.
- Oporavak od grešaka i raspoređivanje opterećenja za JNDI i sve EJB objekte.
- Kopiranje sadržaja HTTP sesija.
- Velika dostupnost Java Message Service (JMS) mehanizma koji se koristi za razmenu poruka između Java Enterprise Edition (Java EE) aplikacija.
- Mogućnost prosleđivanja konfiguracije i otpremljenih razvojnih aplikacija sa jednog čvora u klasteru na sve ostale.



Slika 15. Preuzimanje uloge rukovodećeg čvora u klaster režimu JBoss-a

4.3.1.3 Klasterovanje NGINX raspoređivača

Raspoređivač opterećenja služi kako bi pristigle zahteve ravnomerno raspoređivao svim čvorovima u klasteru. Pored toga raspoređivač stvara iluziju postojanja jednog računara jer se svi čvorovi klastera nalaze pod njegovom kontrolom i korisnici klastera poznaju samo adresu raspoređivača. U implementiranom sistemu kao raspoređivač opterećenja korišćen je NGINX poslužilac [41]. NGINX omogućuje da se definiše skup pozadinskih poslužioca kojima se prosleđuju zahtevi. NGINX omogućava korišćenje tri metode raspoređivanja opterećenja:

- Razigravanje sa težinskim faktorom;
- Najmanji broj veza;
- *IP_Hash* metoda - specifična metoda koja određuje pozadinskog poslužioca na osnovu prva tri okteta IP adrese pristiglog zahteva kod IPv4 adresa ili koristi celu adresu kod IPv6 adresa. Metoda je pogodna u slučajevima kada je potrebno zahteve istog klijenta prosleđivati istom pozadinskom poslužiocu.

NGINX takođe obezbeđuje aktivnu i pasivnu proveru stanja pozadinskih poslužilaca. Aktivna provera podrazumeva periodičnu proveru svakog čvora u klasteru i ukoliko se definisan kriterijum ispuni čvor se smatra neaktivnim ili ponovo aktivnim. Definisani kriterijumi su broj neuspešnih odnosno uspešnih provera. Kod pasivne provere se prilikom prosleđivanja zahteva čvoru prati broj neuspešno obrađenih zahteva ili vremenski interval potreban čvoru da prosledi nazad odgovor. Ukoliko su te vrednosti veće od definisanih vrednosti čvor će se označiti kao neaktivan.

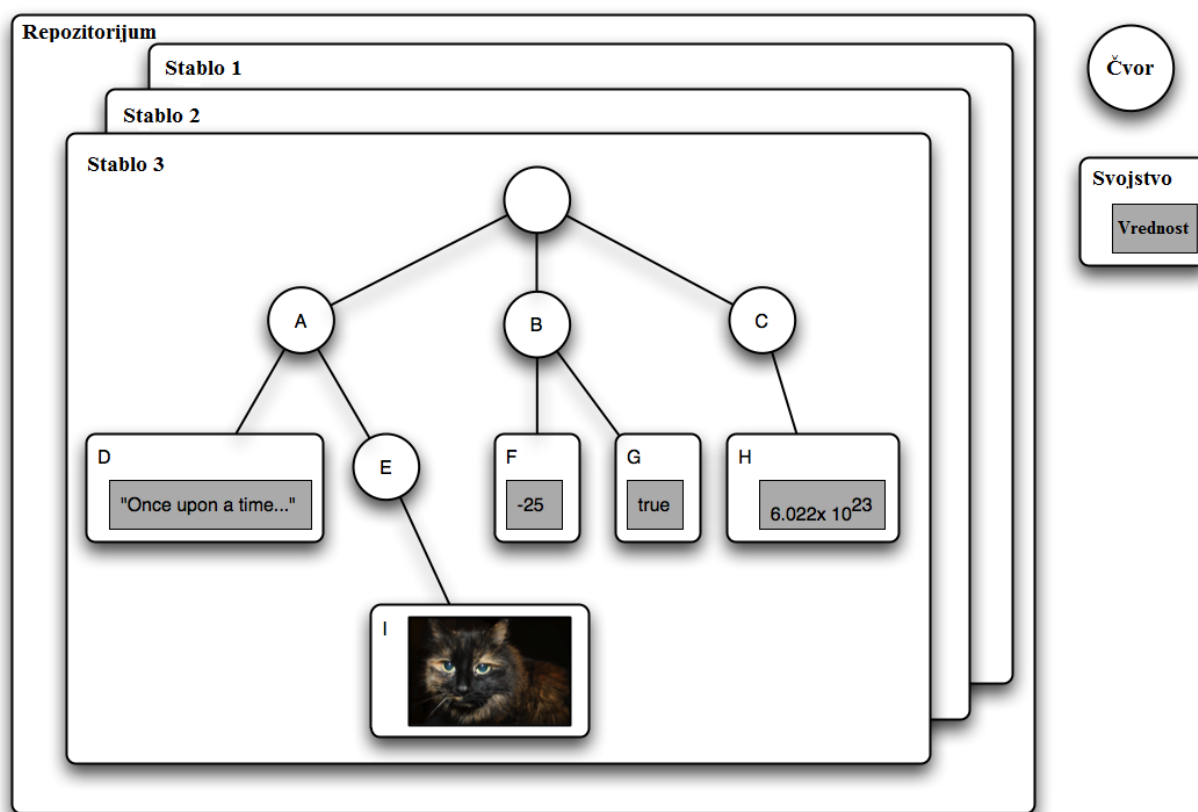
4.3.2 Prilagođenje implementiranog poslužioca za rad u klasteru

Implementirani ACS je potrebno izmeniti da bi njegovo funkcionisanje u klasteru bilo moguće i da bi se pravilno izvršavao. Izmene koje su potrebne su u skladu sa propisanim EJB ograničenjima. Kao što je pomenuto na kraju poglavlja 3.3, upotreba statičkih promenljivih, EJB vremenskih okidača kao i aktivno korišćenje sistema datoteka je neprimereno u okviru klastera i iz tog razloga je njihovu upotrebu potrebno ukloniti ili promeniti način njihove upotrebe.

4.3.2.1 Upotreba sistema datoteka

Upotrebom Java repozitorijuma sadržaja (eng. Content Repository API for Java, JCR) prevaziđen je problem korišćenja sistema datoteka. JCR je specifikacija koja propisuje Java spregu koja obezbeđuje pristup i rukovanje repozitorijumom na jedinstven način. Specifikacija

predstavlja zvanično proširenje Java programskog jezika predstavljeno u okviru JSR-170 - prvu verziju specifikacije [42] i JSR-283 - druga verzija [43]. JCR omogućava smeštanje datoteka i drugih binarnih objekata zajedno sa meta podacima vezanim za te objekte. JCR organizuje sve podatke kao strukturu podataka tipa stabla kao što je prikazano na Slici 16. Čvorovi stabla mogu pored svog naziva i potomaka sadržati i dodatna svojstva među kojima mogu biti i sadržaji datoteka. JCR dalje omogućava vršenje upita i pretrage nad stablom.



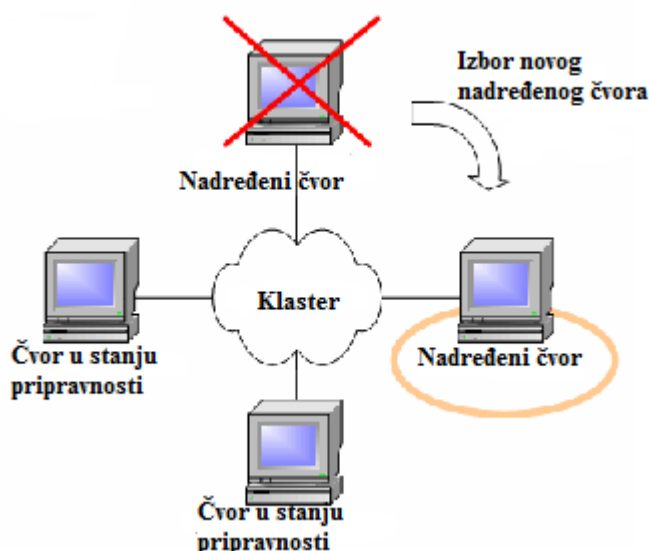
Slika 16. Primer JCR strukture podataka tipa stabla

Kako JCR samo propisuje standardizovan API u pozadini je potreban dodatni mehanizam koji funkcionalnosti API-ja sprovodi u stvarna dela. Pri realizaciji prvo je isprobana upotreba Apache JackRabbit rešenja [44]. Međutim, usled slabe podrške i nemogućnosti za brisanjem datoteka u okviru JCR-a JackRabbit rešenje nije upotrebljeno već je iskorišćeno ModeShape rešenje [45] implementirano od strane RedHat korporacije čiji je proizvod i korišćeni JBoss aplikativni poslužioc. Upravo to je razlog za postojanje dobre dokumentacije i mogućnosti za integrisanjem ModeShape rešenje u JBoss poslužioc. Svaka implementacija JCR standarda omogućava podršku da se datoteke čuvaju u sistemu datoteka ili u bazi podataka. Korišćenje sistema datoteka u ovom slučaju je moguće jer programska podrška poslužioca ne vrši operacije direktno nad sistemom datoteka i svaka operacija je standardizovana i nezavisna je od platforme. ModeShape, kao i JackRabbit, podržava čuvanje datoteka u okviru baze podataka ili sistema

datoteka. Prilikom implementacije ACS-a korišćeno je čuvanje u bazu podataka koja je jedinstvena za sve čvorove klastera. ModeShape je potrebno konfigurisati da radi u klaster modu kako bi svaki čvor imao pristup svim podacima u repozitorijumu.

4.3.2.2 Upotreba EJB vremenskih okidača

Upotrebu standardnih EJB vremenskih okidača u klasteru je potrebno posebno istražiti. Ukoliko zadaci koji se izvršavaju u okviru vremenskih okidača su vezani za globalne podatke, odnosno utiču na ponašanje klastera kao jedinstvenog sistema, njihova primena je tada nemoguća. Razlog tome je što će svaki čvor raditi isti posao nad istim ili možda već promenjenim podacima i posledice takvog ponašanja su nesagledive. Jedno rešenje je da programska podrška na jednom čvoru se razlikuje u odnosu na druge čvorove u tom smislu da će samo taj čvor sadržati vremenske okidače. Međutim takav čvor će predstavljati tačku otkaza u sistemu, jer ukoliko čvor postane neaktivan postaju neaktivni i vremenski okidači. Drugo rešenje je upotreba JBoss klasterizovanih *Singleton* servisa, poznatih kao i *HA Singleton*. *Singleton* servisi omogućavaju da svaki čvor sadrži servis ali da je aktivan samo na jednom čvoru koji se naziva nadređenog. Ukoliko nadređenog čvor postane neaktivan tada sledeći čvor preuzima ulogu nadređenog čvora i izvršava definisani *Singleton* servis kao što je prikazano na Slici 17. Dakle, osim u intervalu kada je jedan nadređenog postao neaktivan a drugi čvor još nije preuzeo ulogu nadređenog čvora, *HA Singleton* je uvek aktivan u klasteru. Na slici je prikazan takav princip rada *Singleton* servisa.



Slika 17. Rukovanje otkazima pri izvršavanju *HA Singleton* servisa

4.3.2.3 Upotreba statičkih promenljivih koje nemaju konačnu vrednost

Poslednji problem prilikom prilagođavanja implementacije ACS-a na rad u klasteru jeste upotreba statičkih promenljivih koja nemaju konačnu vrednost. Statičke promenljive su namenjene upotrebi u slučajevima kada je potrebno da se promenljivoj može pristupiti iz celog sistema programske podrške. Međutim u uslovima klastera te promenljive treba da budu pristupačne programskim podrškama na svim čvorovima. Usled toga i njihova primena u klasteru je nemoguća. Iz tog razloga je potrebno sve statičke promenljive ukloniti ili ih čuvati u nekom eksternom resursu kao što je IMDG ili baza podataka, kako bi svi čvorovi imali pristup svim statičkim promenljivim u sistemu. U ovoj realizaciji je korišćeno smeštanje statičkih promenljivih u bazu podataka iz razloga boljeg obezbeđivanja konzistentnosti i dostupnosti podataka u bazama podataka.

5. Programsko rešenje

Projektni zadatak je realizovan upotrebom Eclipse Java EE razvojnog okruženju na Microsoft Windows i Linux Ubuntu platformama. Referentna baza podataka korišćena u implementaciji jeste PostgreSQL baza podataka. Pored nabrojanih Java EE tehnologija kao što su JBOSS, Infinispan, NGINX, EJB, potrebno je istaći i upotrebu Hibernate tehnologije [46]. Hibernate predstavlja implementaciju sprega propisanih Java specifikacijom JPA (eng. Java Persistence API) koji obezbeđuje upravljanje relacionim podacima.

5.1 Klasterovanje eksternih resursa

U narednim poglavljima objašnjeni su načini konfigurisanja eksternih resursa klastera na način na koji su detaljno objašnjeni u okviru poglavlja 4.3.1.

5.1.1 Konfigurisanje Infinispan-a

Infinispan koji koristi kao IMDG rešenje je potrebno klasterizovati tako da svi podaci budu dostupni svim čvorovima. Infinispan je konfigurisan da radi u asinhronom režimu sa distribuiranjem sadržaja. Infinispan za komunikaciju između čvorova koristi JGroups mehanizam zbog čega je potrebno konfigurisati JGroups kanale na pravilan način. U ovoj implementaciji korišćen je TCP protokol za razmenu poruka unutar JGroups kanala, dok se *User Datagram Protocol* (UDP) koristi za potrebe otkrivanja čvorova. Moguće je koristiti UDP protokol i za razmenu poruka ali nije pogodan za distribuiran režim rada Infinispan-a jer koristi prevelik broj veza. Takođe je preporuka da se UDP koristi u klasterima sa većim brojem čvorova(preko 100 čvorova) a u suprotnom slučaju TCP protokol.

```

<TCP bind_port="7800"
      bind_addr="192.168.237.62" />

<TCPPING timeout="3000"
initial_hosts="${jgroups.tcpping.initial_hosts:192.168.237.62[7800]},192.168.237.112[7800],192.168.237.89[7800]}" port_range="1" num_initial_members="3"/>

```

- *bind_addr* – lokalna IP adresa.
- *bind_port* – port na kojem se otvara JGroups kanal.
- *initial_hosts* – skup čvorova koji predstavljaju početni skup čvorova u klasteru.
- *num_initial_members* – broj čvorova u početnom skupu čvorova u klasteru.

Konfigurisanje Infinispan je moguće obaviti dinamički u okviru programske podrške ili deklarativno kroz XML datoteku. U ovom slučaju korišćena je XML datoteka.

```

<transport>
  <properties>
    <property name="configurationFile" value="jgroups-tcp.xml">
  </property></properties>
</transport>

```

- *configurationFile* – potrebno je navesti putanju datoteke koja sadrži konfiguraciju JGroups kanala.

```

<default>
  <clustering mode="dist">
    <hash numOwners="2" numSegments="100" capacityFactor="2"
      factory="org.infinispan.distribution.ch.SyncConsistentHashFactory"/>
  </clustering>
</default>

```

- *mode* – distribuirani režim rada Infinispan.
- *numOwners* – broj čvorova među kojima se distribuira sadržaj IMDG-a.
- *numSegments* – predstavlja broj segmenata koje svaki čvor kontroliše. Segment predstavlja podelu IMDG ključeva u klasteru.
- *capacityFactor* – predstavlja broj koliko puta će lokalni čvor sadržati više elemenata u odnosu na druge čvorove u klasteru. Ovaj parametar predstavlja sugestiju za *Hash* algoritam ali nema garancije da će čvor sa vrednošću 2 sadržati dvostruko više elementa od čvora sa vrednošću 1.
- *factory* – definiše koji će se *Hash* algoritam koristiti za raspodelu opterećenja.

Definisanje jedne IMDG strukture i Infinispan-u se vrši na sledeći način:

```

<namedCache name="messageQueueCache">
  <eviction maxEntries="200" strategy="LRU"/>
  <persistence passivation="false">
    <singleFile fetchPersistentState="false"
      preload="true" shared="false" purgeOnStartup="false"
      ignoreModifications="false"
      location="data\infinispan\store">
      <async enabled="true"
        modificationQueueSize="100" threadPoolSize="10"/>
    </singleFile>
  </persistence>
</namedCache>

```

- *eviction* – definiše mehanizam koji se koristi kako bi se ograničio broj elemenata IMDG strukture koji se smeštaju u RAM memoriju.

- *maxEntries* – broj elemenata koji se čuvaju u RAM memoriji.
- *strategy* – definiše algoritam koji se koristi za uklanjanje elementa iz RAM memoriji kada se dostigne definisani maksimalni broj elemenata. Poslednje korišćen (eng. Least Recently Used, LRU) algoritam se koristi u ovom slučaju.
- *persistence* – definiše da li se IMDG struktura čuva u trajnoj memoriji.
- *passivation* – predstavlja vezu sa *eviction* mehanizmom. Ukoliko je onemogućen *passivation* to znači da se u trajnoj memoriji nalaze elementi koji su u RAM memoriji ali i elementi koji su upotrebom *eviction* mehanizma uklonjeni iz RAM memorije. Ukoliko je omogućen *passivation* to znači da se u trajnoj memoriji nalaze samo elementi koji su upotrebom *eviction* mehanizma uklonjeni iz RAM memorije, dok se elementi koji su u RAM memoriji samo u njoj čuvaju.
- *singleFile* – definiše naziv i putanju datoteke u kojoj se smešta IMDG struktura kao trajno sačuvana struktura. Dodatno definiše da li se prilikom aktiviranja Infinispan-a podaci iz takve datoteke učitavaju u IMDG strukturu.
- *async* – obezbeđuje aktiviranje asinhronog režima čuvanja izmena u trajnu memoriju.
- *modificationQueueSize* – definiše veličinu reda čekanja u kojoj se smeštaju izmenjeni elementi kasnije upotrebom IMDG niti smeštaju u trajnu memoriju.
- *threadPoolSize* – definiše broj niti koji opslužuje red čekanja i smešta izmene iz reda u trajnu memoriju IMDG strukture.

Bitna funkcionalnost koju Infinispan nudi jeste uklanjanje elemenata iz IMDG strukture posle određenog vremenskog intervala neaktivnosti elementa.

```
<namedCache name="sessionContextCache">
  <expiration maxIdle="30000"/>
</namedCache>
```

- *expiration* – definiše mehanizam koji se vrši uklanjanje elemenata iz IMDG strukture posle definisanog vremenskog intervala neaktivnosti elementa ili životnog veka elementa.
- *maxIdle* – definiše vremenski interval dozvoljene neaktivnosti elementa.

5.1.2 Konfigurisanje JBoss aplikativnog poslužioca

Konfigurisanje aplikativnog poslužioca je neophodno kako bi se omogućile funkcionalnosti koje JBoss pruža u klaster režimu rada. U implementaciji ACS-a najneophodnije stavke su pre svega podrška za već opisani mehanizam HA *Singleton* servis. Takođe pogodna mogućnost je i automatsko otpremanje programske podrške sa nadređenog čvora na podređene čvorove. Konfiguracioni parametri JBoss-a se sa nadređenog čvora prenose na podređene čvorove što takođe predstavlja olakšanje pri kreiranju klastera. Pri konfiguraciji JBoss-a na rad u klasteru postoji režim nadređenog i režim podređenog čvora. U okviru *host.xml* konfiguracione datoteke se navodi u kom režimu se JBoss pokreće.

```
<host name="master" xmlns="urn:jboss:domain:1.2">
```

```
<host name="master" xmlns="urn:jboss:domain:1.2">
```

Dodavanje korisnika u okviru JBoss nadređenog čvora je neophodno kako bi se omogućilo pristupanje upravljačkom alatu JBoss-a. Takođe, potrebno je konfigurisati sprege na nadređenog čvoru kojima se omogućava udaljeno pristupanje programskim podrškama otpremljene u okviru JBoss-a kao i upravljačkim alatima JBoss-a. Pri konfiguraciji je važno navesti pravilnu lokalnu IP adresu. Ova adresa je bitna jer se pri konfiguraciji podređenih čvorova navodi IP adresa nadređenog čvora.

```
<interfaces>
  <interface name="management">
    <inet-address value="{jboss.bind.address.management:192.168.237.62}"/>
  </interface>
  <interface name="public">
    <inet-address value="{jboss.bind.address:192.168.237.62}"/>
  </interface>
  <interface name="unsecure">
    <inet-address value="{jboss.bind.address.unsecure:192.168.237.62}"/>
  </interface>
</interfaces>
```

Pri konfiguraciji podređenih čvorova koriste se isti konfiguracioni parametri. Najbitnije pri konfigurisanju podređenog čvora jeste poznavanje IP adrese nadređenog čvora kao i šifrovane vrednosti potrebne za pristupanje nadređenom čvoru. Šifrovana vrednost se dobija upotrebom sledeće formule:

Base64(password)

- *Base64* – algoritam za šifrovanje podataka;
- *password* – šifra korisnika upravljačkih alata na nadređenom JBoss-u.

Vrednost dobijena primenom navedene formule se koristi pri konfigurisanju podređenog čvora.

```
<security-realms>
  <security-realm name="ManagementRealm">
    <server-identities>
      <secret value="MTIzMTIz="/>
    </server-identities>
    <authentication>
      <properties path="mgmt-users.properties" relative-to="jboss.domain.config.dir"/>
    </authentication>
  </security-realm>
</security-realms>
```

Na kraju potrebno je navesti IP adresu nadređenog čvora i port konfiguracionog alata pri konfigurisanju podređenog čvora.

```
<domain-controller>
  <remote host="{192.168.237.62}" port="{9999}"
    security-realm="ManagementRealm"/>
</domain-controller>
```

5.1.3 Konfigurisanje Nginx raspoređivača opterećenja

Raspoređivač opterećenja predstavlja jako bitnu komponentu u klasteru usled važne funkcije ravnomernog raspoređivanja pristiglih zahteva. Raspoređivač vodi računa o aktivnosti svih čvorova kako bi znao kojim čvorovima da prosleđuje zahteve. U ovoj rešenju korišćen je Nginx poslužilac koji se pokazao kao veoma brz i efikasan sa malim utroškom i zahtevom za memorijom.

```

http {
    server {
        location /insight_acs_tr069/ {
            proxy_pass    http://pool;
        }
    }
    limit_upstream_zone test 10m;
    upstream pool {
        server 192.168.247.48:8080 max_fails=3 fail_timeout=30s;;
        server 192.168.247.47:8080 max_fails=3 fail_timeout=30s;;
        server 192.168.247.53:8080 weight=2 max_fails=3 fail_timeout=30s;;

        health_check interval=10 fails=3 passes=2;
        limit_upstream_conn limit=260 zone=test backlog=100 timeout=180s;
        limit_upstream_log_level error;
    }
}

```

- *server* – predstavlja definisan virtualni poslužioc.
- *location* – definišu se jednoobrazni identifikatori resursa (eng. Uniform Resource Identifier, URI) koje Nginx procesira u okviru virtualnog poslužioca.
- *proxy_pass* – služi za direktno prosleđivanje pristiglih zahteva od klijenta ka definisanom URL-u. Odgovor sa zadatog URL-a se potom prosleđuje nazad klijentu.
- *limit_upstream_zone* – definiše veličinu RAM memorije koja služi za skladištenje podataka.
- *upstream* – definiše se grupa poslužioca kojima se prosleđuju pristigli zahtevi.
- *server* – definiše se URL poslužioca kojem se prosleđuju zahtevi.
- *max_fails* – definiše broj neuspešnih pokušaja obrade poslatih zahteva u definisanom vremenskom intervalu, potrebnih da bi se poslužilac smatrao neaktivnim.
- *fail_timeout* – definiše vremenski interval u okviru kog se određeni broj neuspešnih pokušaja obrade zahteva od strane poslužioca kako bi se poslužilac smatrao neaktivnim.

- *weight* – definiše težinski faktor algoritma razigravanja za raspoređivanje opterećenja. Ukoliko nije drugačije definisano, algoritam razigravanja je podrazumevani algoritam za raspoređivanje opterećenja.
- *health_check* – definiše aktivan način provere aktivnosti definisanih poslužioca.
- *interval* – definiše vremenski interval između dve provere aktivnosti. Podrazumevana vremenska jedinica je sekund.
- *fails* – definiše potreban broj neuspešnih provera da bi se poslužilac smatrao neaktivnim.
- *passes* – definiše potreban broj uspešnih provera da bi se poslužilac smatrao aktivnim.
- *limit_upstream_conn* – omogućava definisanje maksimalnog broja zahteva prosleđenih poslužiocu kao i red čekanja za novo pristigle zahteve.
- *limit* – definiše maksimalnog broja zahteva koji se prosleđuju poslužiocu.
- *backlog* - definiše veličinu reda čekanja za nove zahteve ukoliko je *limit* već ispunjen.
- *timeout* – definiše vremenski interval koliko zahtev može biti prisutan u redu čekanja. Ukoliko istekne definisani interval Nginx vraća grešku klijentu.
- *limit_upstream_log_level* – definiše nivo praćenja rada *limit_upstream_conn* komponente.

5.2 Programsko rešenje implementiranog poslužioca

Programska implementacija ACS-a je detaljno izložena u okviru [47][48]. U ovom poglavlju će se naznačiti delovi implementacije koji su unapređeni ili izmenjeni zbog potrebe optimizacije. Takođe će se prikazati dodati moduli potrebni u cilju prilagođavanja implementacije ACS-a na rad u klasteru. Radi lakšeg praćenja programskog rešenja u okviru svakog poglavlja predstaviće se po jedan modul, nabrojani u poglavlju 4.1, sa kratkim opisom a zatim će se naglasiti izmene i razlozi izmena.

5.2.1 Biznis logika

Pri realizaciji biznis logike korištene su EJB tehnologije, tačnije korišćene su *Stateless SessionBean* objekti. Osnovna karakteristika ovih objekata jeste da se ne čuvaju podaci između sesija klijenata sa objektima. U slučaju paralelnog pristupa biznis logici kreiraju se nove instance *Stateless SessionBean* objekata pri čemu ne postoji međusobna komunikacija između postojećih objekata.

Biznis logika obezbeđuje rukovanje nad bazom podataka i IMDG-om. Najvažniji moduli su prikazani u Tabeli 3.

Naziv modula	Opis
CPEBean	Modul za rukovanje virtualnim objektima CPE uređaja u bazi podataka.
CPEParameterBean	Modul za rukovanje modelom podataka CPE uređaja u bazi podataka.
GroupBean	Modul za rukovanje grupisanjem CPE uređaja u bazi podataka.
OwnerInfoBean	Modul za rukovanje pretplatnika CPE uređaja u bazi podataka.
CPEActivityBean	Modul za rukovanje aktivnostima koje se dešavaju u okviru rada poslužioca i njihovo rukovanje u IMDG strukturi podataka.
CPEIssuesBean	Modul za nadgledanje potencijalnih problema pri radu CPE uređaja i njihovo rukovanje u IMDG strukturi podataka.
ConfigurationPackageBean	Modul za izvršavanje konfiguracionih paketa nad CPE uređajima.
EjbRPC	Modul koji enkapsulira funkcionalnosti biznis logike u jedan modul koji se koristi kao API prema raznim korisnicima modula biznis logike.
EjbTR069API	Modul koji obezbeđuje API prema TR-069 komunikacionom sloju.
EjbTR069Events	Modul za rukovanje TR-069 događajima prosleđeni od strane TR-069 komunikacionom sloju.

Tabela 3. Funkcionalni moduli biznis logike

Radi postizanja optimizacije ACS-a izmene su izvršene u okviru *CPEParameterBean*, *EjbTR069API* i *EjbTR069Events* modula. Izmene su izvršene radi izbegavanja pojava redundantnih podataka i omogućavanja upotrebe kompleksnih SQL izraza. SQL izrazi se dinamički kreiraju.

```
UPDATE cpeparameter SET isApplied = 0,
    value = CASE path
        WHEN 'Device.DeviceSummary' THEN "
        WHEN 'Device.DeviceInfo.HardwareVersion' THEN
        'Hardware Version 2'
        WHEN 'Device.ManagementServer.ParameterKey' THEN
        'bmvfibiapk6la59ntoq39eqmtk'
        WHEN 'Device.LAN.IPAddress' THEN '192.168.1.1'
        ELSE value END
    WHERE cpeid = 0 AND path
    IN
    ('Device.DeviceSummary','Device.DeviceInfo.HardwareVersion','Device.DeviceI
    nfo.SoftwareVersion','Device.ManagementServer.ParameterKey','Device.LAN.IP
    Address');
```

5.2.2 Prilagođenje implementiranog poslužioca

U narednim poglavljima objašnjeni su načini prilagođenja implementiranog poslužioca za rad u klasteru kao što je objašnjeno u okviru poglavlja 4.3.1.

5.2.2.1 Modul za rukovanje sistemom datoteka

Modul za rukovanje sistemom datoteka obezbeđuje rukovanje JCR repozitorijumom. Modul omogućava dodavanje i uklanjanje specifičnih datoteka, kao i preuzimanje njihovog binarnog sadržaja.

Prototip funkcije	addFirmwareIntoRepo(String firmwareName, InputStream is)
Opis funkcije	Dodavanje „firmware“ datoteke u JCR repozitorijum.
Parametri funkcije	firmwareName – naziv datoteke is – binarni sadržaj datoteke
Povratna vrednost funkcije	Logička vrednost – da li je dodavanje uspešno izvršeno ili ne

Prototip funkcije	removeFirmwareFromRepo(String firmwareName)
Opis funkcije	Uklanjanje „firmware“ datoteke iz JCR repozitorijum.
Parametri funkcije	firmwareName – naziv datoteke
Povratna vrednost funkcije	Logička vrednost – da li je uklanjanje uspešno izvršeno ili ne

Prototip funkcije	getFirmwareFromRepo(String firmwareName)
Opis funkcije	Preuzimanje binarnog sadržaja „firmware“ datoteke iz JCR repozitorijuma.
Parametri funkcije	firmwareName – naziv datoteke
Povratna vrednost funkcije	Binarni sadržaj datoteke – ukoliko navedena datoteka postoji u repozitorijumu, odnosno ukoliko ne postoji povratna vrednost je NULL.

Prototip funkcije	addCPELogIntoRepo(String cpeId, InputStream is)
Opis funkcije	Dodavanje „log“ datoteke u JCR repozitorijum.
Parametri funkcije	cpeId – identifikacioni broj CPE uređaja koji je prosledio „log“ datoteku is – binarni sadržaj datoteke

Povratna vrednost funkcije	Logička vrednost – da li je dodavanje uspešno izvršeno ili ne
----------------------------	---

Prototip funkcije	<code>getCPELogFromRepo(String cpeId, int offset, int limit)</code>
Opis funkcije	Čitanje teksta iz binarnog sadržaja „log“ datoteka iz JCR repozitorijum. Tekst se čita u sekvencijalnom redosledu iz datoteka sortiranih u hronološkom redosledu.
Parametri funkcije	cpeId – identifikacioni broj CPE uređaja koji je prosledio „log“ datoteku offset – redni broj početnog reda „log“ datoteke limit - broj redova iz „log“ datoteke
Povratna vrednost funkcije	Lista redova teksta – ukoliko za navedeni CPE uređaj postoje „log“ datoteke vraća se navedeni broj ili manje redova teksta, odnosno ukoliko ne postoji povratna vrednost je prazna lista.

Prototip funkcije	<code>removeAllCPELogsFromRepo(String cpeId)</code>
Opis funkcije	Uklanjanje svih „log“ datoteka iz JCR repozitorijuma za navedeni CPE uređaj.
Parametri funkcije	cpeId – identifikacioni broj CPE uređaja koji je prosledio „log“ datoteku

Prototip funkcije	<code>getAllCPELogs(String cpeId)</code>
Opis funkcije	Preuzimanje binarnih sadržaja svih „log“ sa pridruženim nazivom datoteke iz JCR repozitorijuma.
Parametri funkcije	cpeId – identifikacioni broj CPE uređaja koji je prosledio „log“ datoteku
Povratna vrednost funkcije	Lista parova (naziv datoteke, sadržaj datoteke) - ukoliko za navedeni CPE uređaj postoje „log“ datoteke, odnosno ukoliko ne postoji povratna vrednost je prazna lista.

5.2.3 Modul za distribuirane vremenske okidače

Za potrebe realizacije periodičnih zadataka koji se aktiviraju u određenim vremenskim intervalima korišćeni su HA *Singleton* servisi. Da bi se rad HA *Singleton* servisa omogućio

najpre je potrebno implementirati mehanizam koji će pokrenuti rad servisa. U tu svrhu je potrebno preklopiti funkcionalnost sledeće funkcije:

- *void activate(ServiceActivatorContext context)*

Potom je neophodno implementirati servis i njegove funkcionalnosti definisanjem sledećih funkcija:

- *void start(StartContext arg0)* – funkcija se aktivira pri prvom pokretanju servisa
- *void stop(StopContext arg0)* – funkcija se aktivira prilikom potpunog prekida rada servisa

U okviru navedenih funkcija potrebno je upotrebom JNDI API-ja pronaći sve *Singleton* objekte koji predstavljaju vremenske okidače i pozvati njihovu funkciju pokretanja, odnosno zaustavljanja.

Za definisanje vremenskih okidača upotrebljen je EJB *Timeout* mehanizam koji omogućava definisanje periode izvršavanja navedene funkcije. U okviru te funkcije se definiše zadatak koji se izvršava u okviru tog vremenskog okidača. U Tabeli 4 prikazan je spisak svih vremenskih okidača zajedno sa kratkim opisom funkcionalnosti koji su implementirani.

Naziv vremenskog okidača	Opis funkcionalnosti
CPEStatusChecker	Periodična provera statusa svih prijavljenih CPE uređaja.
UploadLogFile	Periodično zahtevanje otpremanja „log“ datoteka sa svih prijavljenih CPE uređaja.
STUNChecker	Periodična provera statusa STUN poslužioca neophodnog za realizaciju <i>Connection Request</i> mehanizma. Ukoliko se stanje promenilo sprovodi se akcija postavljanja određenih vrednosti parametara na svim CPE uređajima.
CPEActivityValidity	Periodična provera validnosti elemenata koji sadrže ACS aktivnosti u okviru IMDG strukture.
CPEActivityDeletion	Periodična provera da li je potrebno ukloniti neki od elemenata koji sadrže ACS aktivnosti u okviru IMDG strukture.

Tabela 4. Implementirani vremenski okidači

5.2.4 TR-069 komunikacioni sloj

Sloj je detaljno opisan u [48]. Međutim, u cilju optimizacije ovaj modul je najviše izmenjen. Sloj se u velikoj meri oslanja na IMDG strukture. Dizajn upotrebe IMDG struktura je u velikoj meri izmenjen, kao što je opisano u konceptu rešenja, što dovodi do velikih izmena u implementaciji sloja. Struktura modula implementira u okviru TR-069 komunikacionog sloja je ostala nepromenjena. U Tabeli 5 su prikazani moduli sloja zajedno sa opisom funkcionalnosti.

Naziv modula	Opis funkcionalnosti
ACSServlet	Pristupna tačka za CPE uređaje.
TR069SOAPParser	Izvršava pretvaranje ulaznog toka podataka u Java objekte.
TR069MessageHandler	Interpretira pristigle poruke, ažurira stanje sistema i generiše odgovor.
TR069ConnunicationAPI	Prilagodni sloj prema modulu biznis logike ACS-a.

Tabela 5. Funkcionalni moduli u TR-069 komunikacionom sloju [48]

Iako je implementacija funkcija u unutar TR-069 komunikacionog sloja izmenjena njihova funkcionalnost je ostala identična i iz tog razloga se neće navoditi u nastavku rada. Dalje su prikazane funkcije koje su dodate radi prilagođavanja novom dizajnu upotrebe IMDG struktura.

Prototip funkcije	checkSessionID(String sessionID)
Opis funkcije	Proverava da li je prijavljeni CPE uređaj započeo sesiju sa ACS-om i da li je definisana sesija validna trenutna sesija.
Parametri funkcije	sessionId – identifikaciona reč TR-069 sesije
Povratna vrednost funkcije	Identifikacionu struktura virtualnog CPE-a – ukoliko je definisana sesija validna, odnosno ukoliko nije validna povratna vrednost je NULL.

Prototip funkcije	generateSessionID()
Opis funkcije	Služi za generisanje identifikacione reči nove sesije. Potrebno je da identifikaciona reč bude jedinstvena kako ne bi došlo preplitanja sesija.
Povratna vrednost funkcije	Identifikaciona reč sesije.

Prototip funkcije	removeFirstRequest(DeviceIdStruct deviceIdStruct)
Opis funkcije	Uklanja RPC zahtev koji na koji je uspešno stigao odgovor iz liste RPC zahteva.
Parametri funkcije	deviceIdStruct – identifikaciona struktura virtualnog CPE-a
Povratna vrednost funkcije	Objekat koji predstavlja poslati RPC zahtev – ukoliko za definisanu identifikacionu strukturu virtualnog CPE-a postoji lista RPC zahteva sa bar jednim elementom, odnosno u suprotnom slučaju povratna vrednost je NULL.

6. Ispitivanje i rezultati

Ispitivanje implementiranog rešenja je vršeno u laboratorijskim uslovima. Kreiran je IoT ekosistem kao što je prikazano na Slici 18. Korisnički uređaji su simulirani postojećim rešenjem CPE biblioteke [49]. Usled potrebe za velikim brojem korisničkih uređaja, kako bi se verodostojno definisao IoT ekosistem, CPE klijenti su pokretani u velikom broju na računarskim sistemima sa simuliranim radom STB uređaja. Simuliran rad STB uređaja podrazumeva menjanje vrednosti parametara iz TR-135 modela podataka, kao što je npr. trenutno gledani kanal što uzrokuje dodatnu promenu zavisnih parametara, ili kvaliteta antenskog signala. Time se postiže dinamičko ponašanje kreiranog IoT ekosistema.



Slika 18. IoT ekosistem kreiran za ispitivanje rešenja

Ispitivanje tehničkih zahteva implementiranog rešenja je vršeno upotrebom grafičkih aplikacija kao što su Web i Android aplikacija. Grafičke aplikacije koriste API-je ACS-a.

Svrha ispitivanja implementiranog ACS-a jeste da se dokaže funkcionalna ispravnost ACS-a, poboljšanja koje je doprinela optimizacija kao i ispitivanje skalabilnosti ACS-a u okviru klastera. Cilj ispitivanja klastera jeste da se pokaže maksimalan broj klijenata koje ACS u okviru

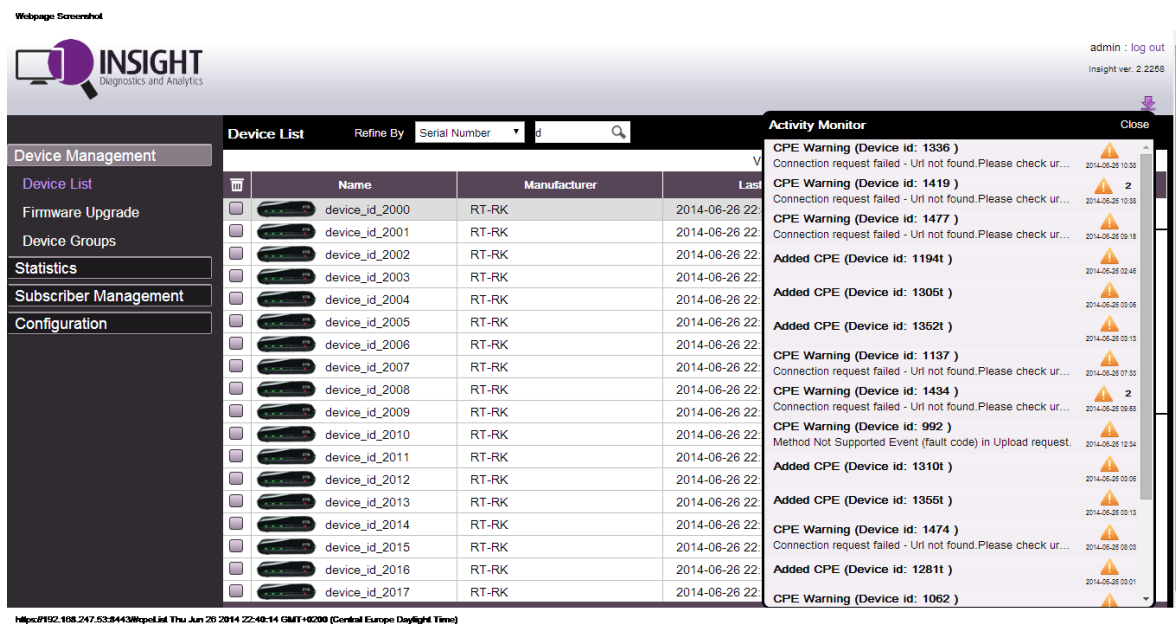
klastera može da opsluži u zavisnosti od fizičkih mogućnosti. Na osnovu takvih rezultata može se dalje predviđati ponašanje klastera u odnosu na poboljšanja, ili pogoršanja fizičkih sposobnosti klastera.

6.1 Ispitivanje tehničkih zahteva

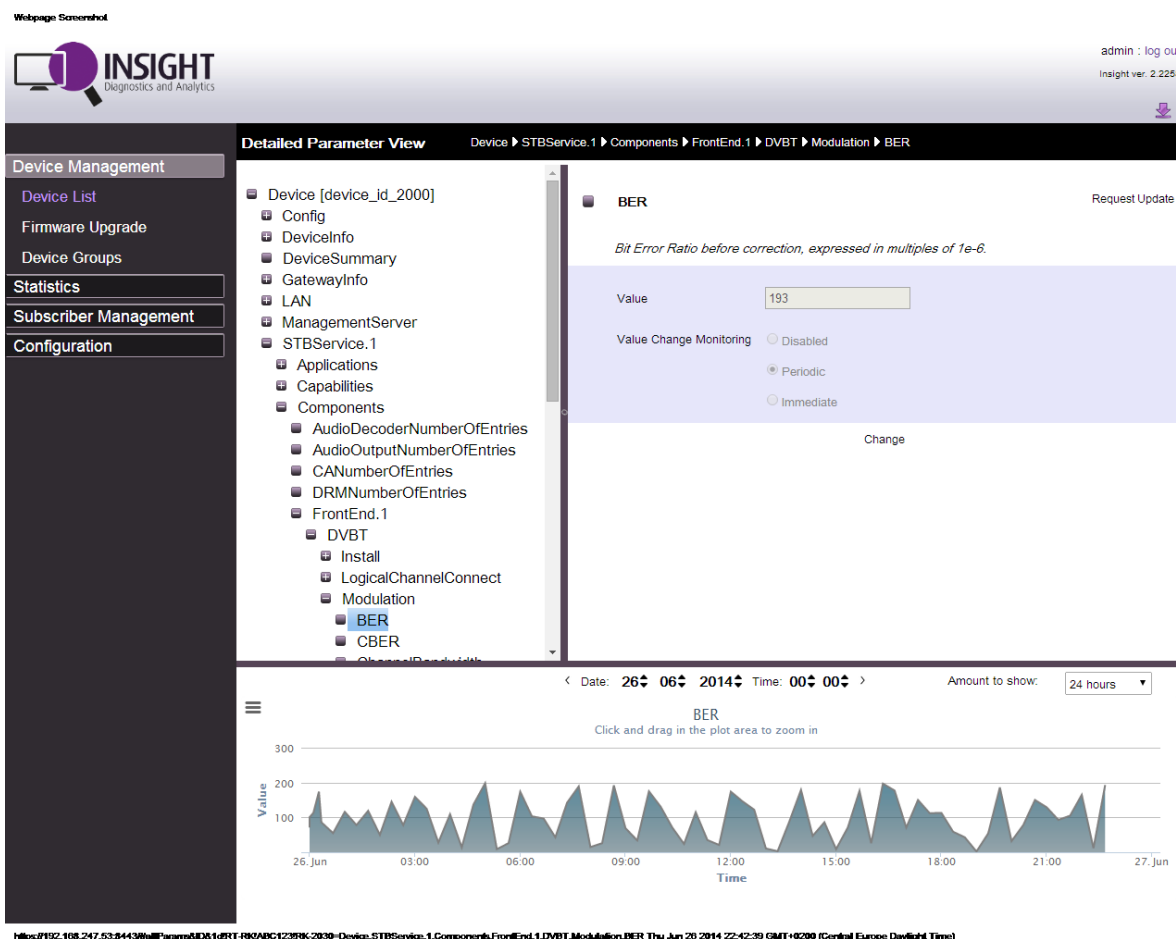
Ispitivanje tehničkih zahteva je izvršeno upotrebom Google Web Toolkit (GWT) aplikaciji [50]. U Tabeli 6 je dat spisak zahteva koja su testirana sa kratkim opisom testa i rezultatima testa.

Tehnički zahtevi	Opis testa
Nadgledanje uređaja	Ispitana je komunikacija sa CPE uređajima kao i praćenje rada uređaja. Na Slici 19 je prikazana lista CPE uređaja sa njihovom aktivnošću.
Konfigurisanje uređaja	Ispitan je zahtev ACS-a da je u mogućnosti da konfiguriše CPE uređaje postavljanjem određenih parametara na željene vrednosti. Na Slici 20 je prikazano stablo modela parametra sa mogućnošću da se izabere parametar i postavi njegova vrednost ili vrednost njegovih atributa.
Obezbeđivanje aplikativnih sprega	Ispitane su sprege kojima se distribuiraju podaci prikupljeni sa CPE uređaja. Na Slici 21 prikazani su vremenski trenuci menjanja kanala STB uređaja, statistika gledanosti kao i kvalitet signala grupe STB uređaja.

Tabela 6. Ispitivanje tehničkih zahteva

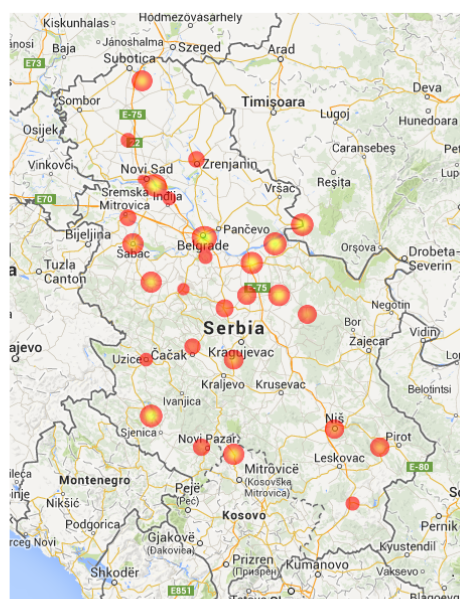
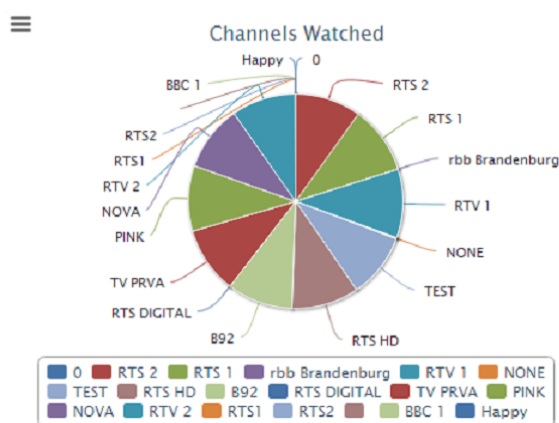


Slika 19. Lista prijavljenih CPE uređaja u GWT aplikaciji



Slika 20. Stablo modela podataka CPE uređaja u GWT aplikaciji

Name of Channel	Time of Channel Zapping
B92	20/06/2014, 01:55:40
NOVA	20/06/2014, 01:56:10
RTV 2	20/06/2014, 01:56:40
RTS 2	20/06/2014, 01:57:10
RTS HD	20/06/2014, 01:57:40
RTV 2	20/06/2014, 01:58:10
RTV 1	20/06/2014, 01:58:40
TEST	20/06/2014, 01:59:10
PINK	20/06/2014, 01:59:40
RTV 2	20/06/2014, 02:00:10



Slika 21. Primeri servisa koji koriste usluge ACS-a

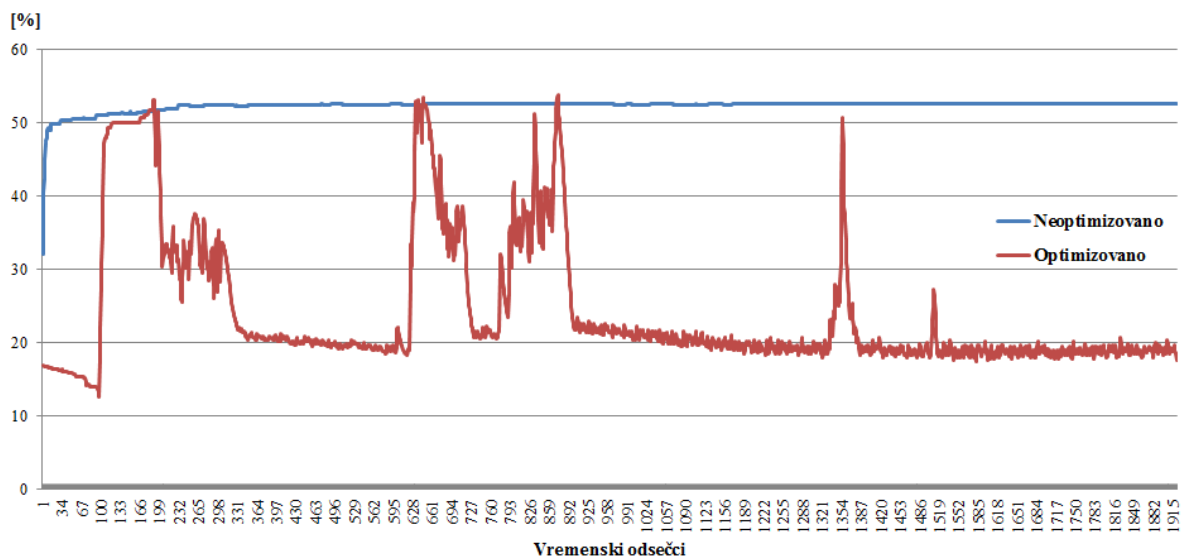
6.2 Ispitivanje optimizacija postojećeg rešenja

Radi ispitivanja korišćeni su simulacioni CPE uređaji jer je potreban broj uređaja za ispitivanje meri u stotinama ili hiljadama uređaja. U okviru ovog ispitivanja vršena su poređenja neoptimizovanog i optimizovanog rešenja. Za potrebe ispitivanja korišćen je računarskih sistem sledećih performansi fizičke arhitekture prikazanim u Tabeli 7.

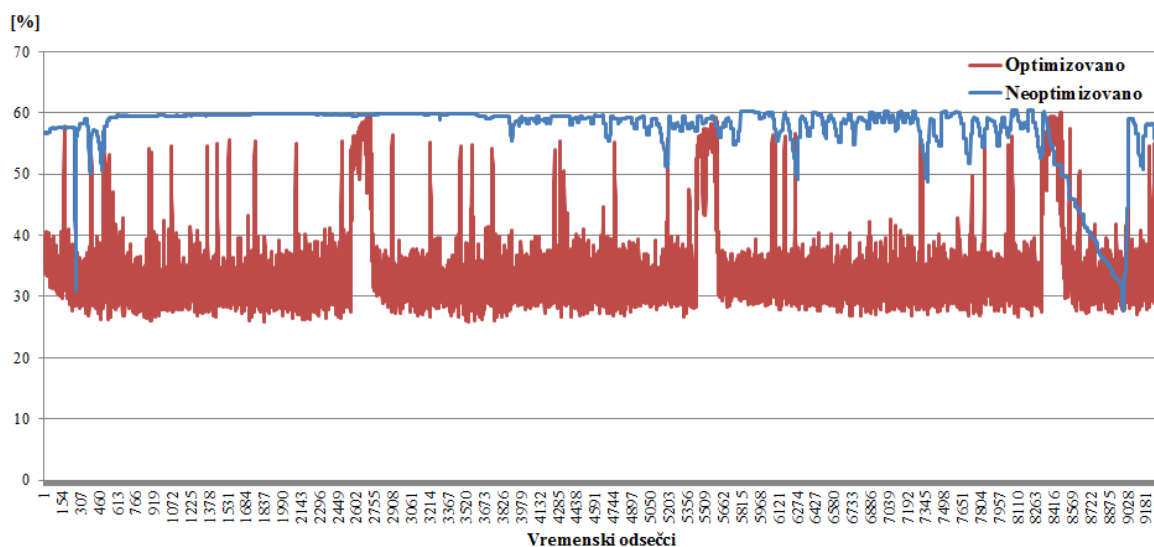
Procesor	Intel i5-2400 sa taktom 3.10GHz
RAM memorija	4GB sa taktom od 1333 MHz
Trajna memorija	500GB sa brzinom obrtanja glave diska 7,200 obrtaja/minuta
Operativni sistem	Linux Ubuntu 12.04.4 LTS x86_64

Tabela 7. Performanse fizičke arhitekture pri ispitivanju optimizacije

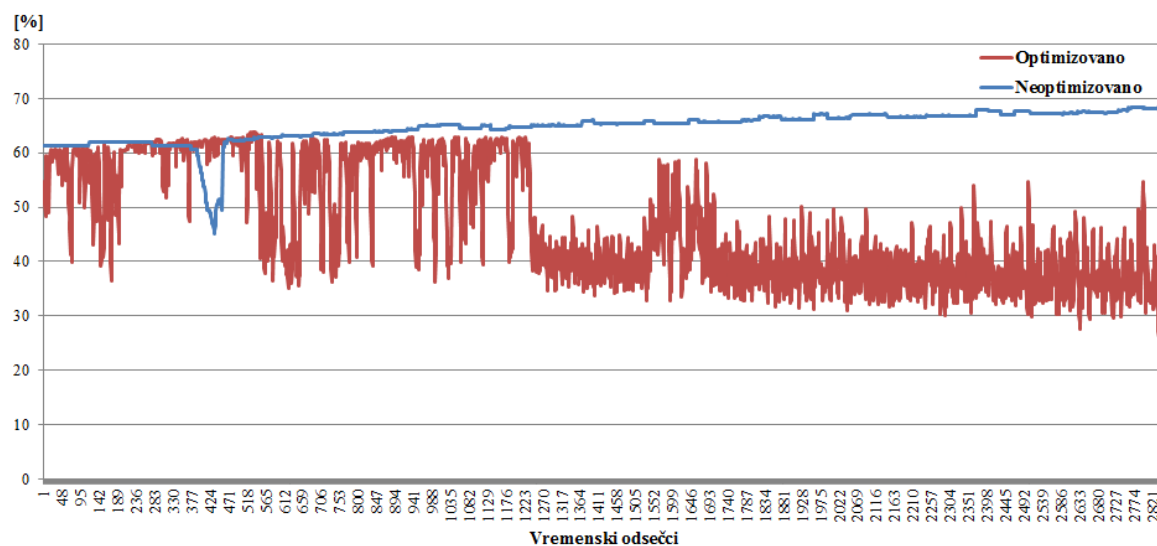
Pri ispitivanju, perioda prijavljivanja CPE klijenata je bila 10 minuta. Na Slikama 22, 23 i 24 je prikazana potrošnja RAM memorije optimizovanog i neoptimizovanog ACS-a u odnosu na broj CPE klijenata. Na Slikama 25, 26 i 27 je prikazano opterećenje procesora optimizovanog i neoptimizovanog ACS-a u odnosu na broj CPE klijenata.



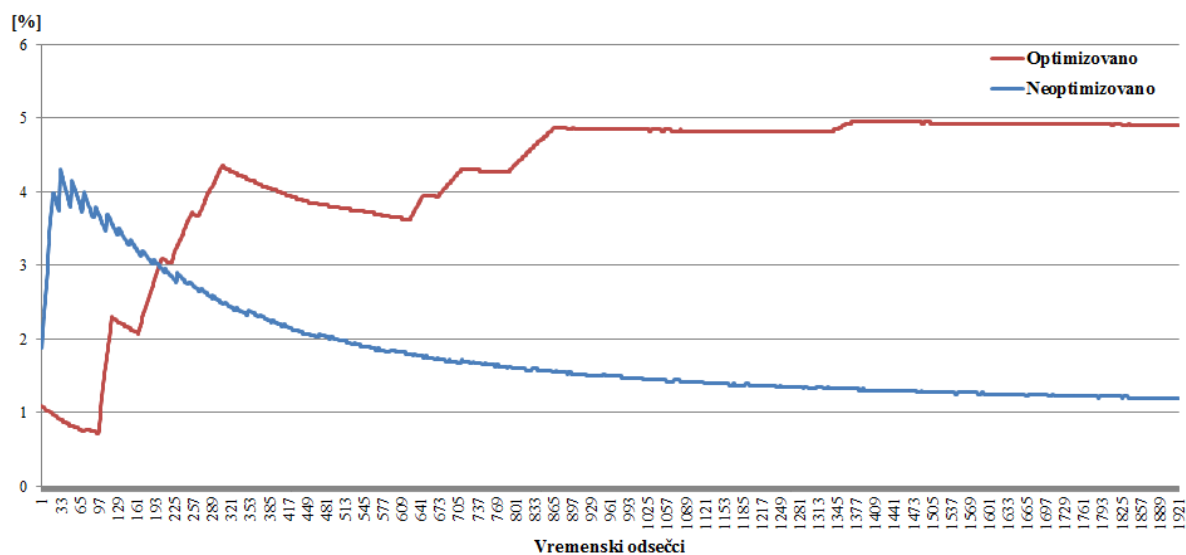
Slika 22. Potrošnja RAM memorije pri opsluživanju 500 CPE klijenata



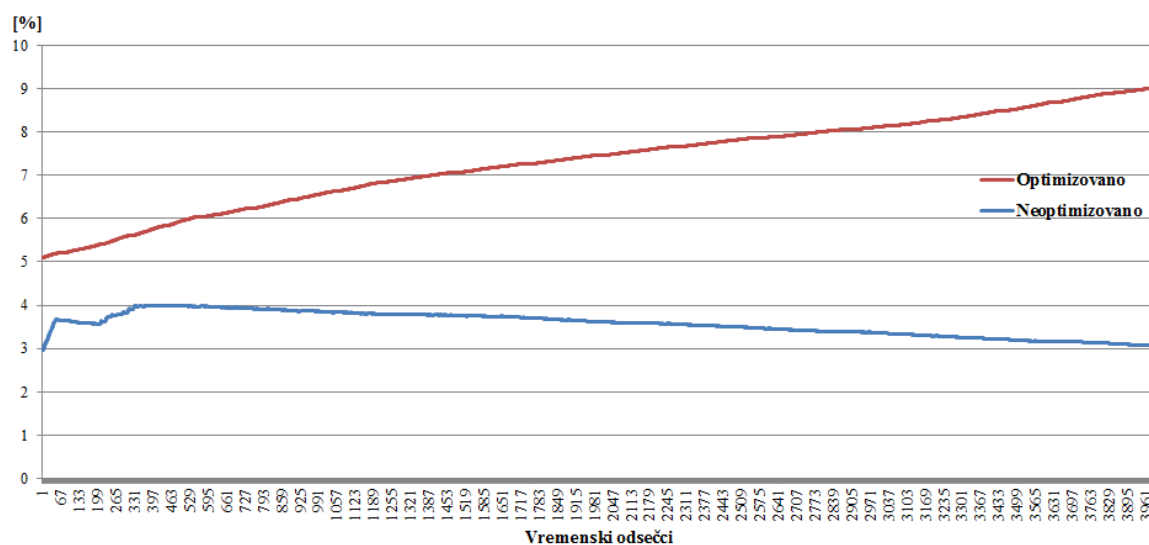
Slika 23. Potrošnja RAM memorije pri opsluživanju 1000 CPE klijenata



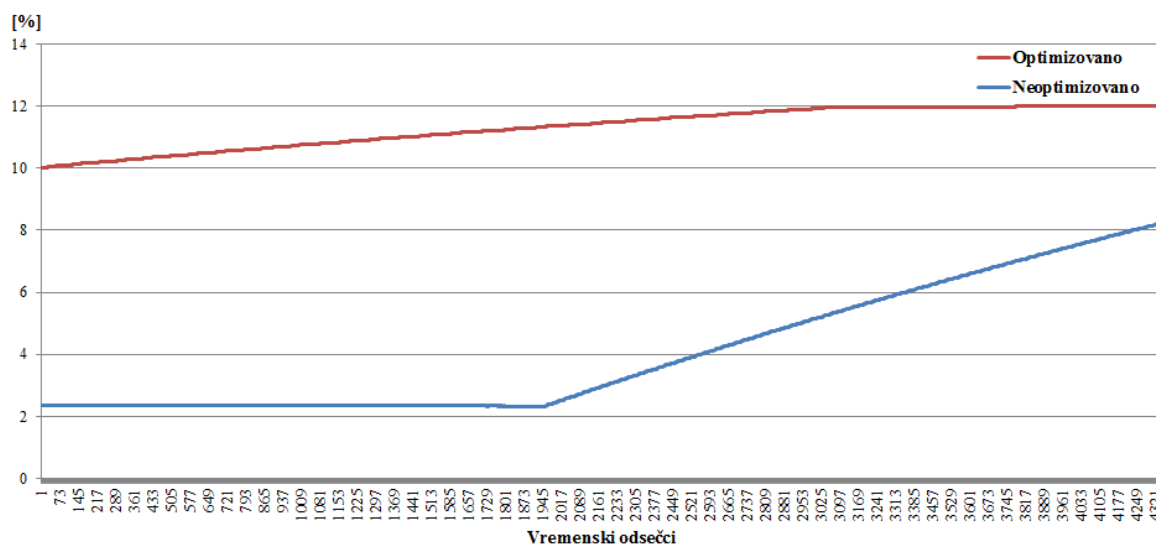
Slika 24. Potrošnja RAM memorije pri opsluživanju 1500 CPE klijenata



Slika 25. Opterećenje procesora pri opsluživanju 500 CPE klijenata



Slika 26. Opterećenje procesora pri opsluživanju 1000 CPE klijenata



Slika 27. Opterećenje procesora pri opsluživanju 1500 CPE klijenata

Iz prikazanih rezultata merenja jasno je da je postignuta optimizacija pri upotrebi RAM memorije dok je opterećenje procesora u slučaju optimizovanog rešenja nešto veća u odnosu na neoptimizovano. Razlog takvom ponašanju jeste veća aktivnost procesora pri nagledanju IMDG struktura podataka koje su smeštene u RAM memoriji. Ugrađeni mehanizmi IMDG strukture služe za izmeštanje elemenata iz RAM memorije nakon što su prekršili definisana ograničenja, kao što su npr. dozvoljeno vreme mirovanja elementa ili broj elemenata strukture u RAM memoriji. Kako je opterećenje procesora zanemarljivo i kod optimizovanog i neoptimizovanog rešenja, maksimalna vrednost oko 12%, sa obzirom da je u potrošnji memorije napravljen značajan pomak, optimizovano rešenje je prihvatljivo.

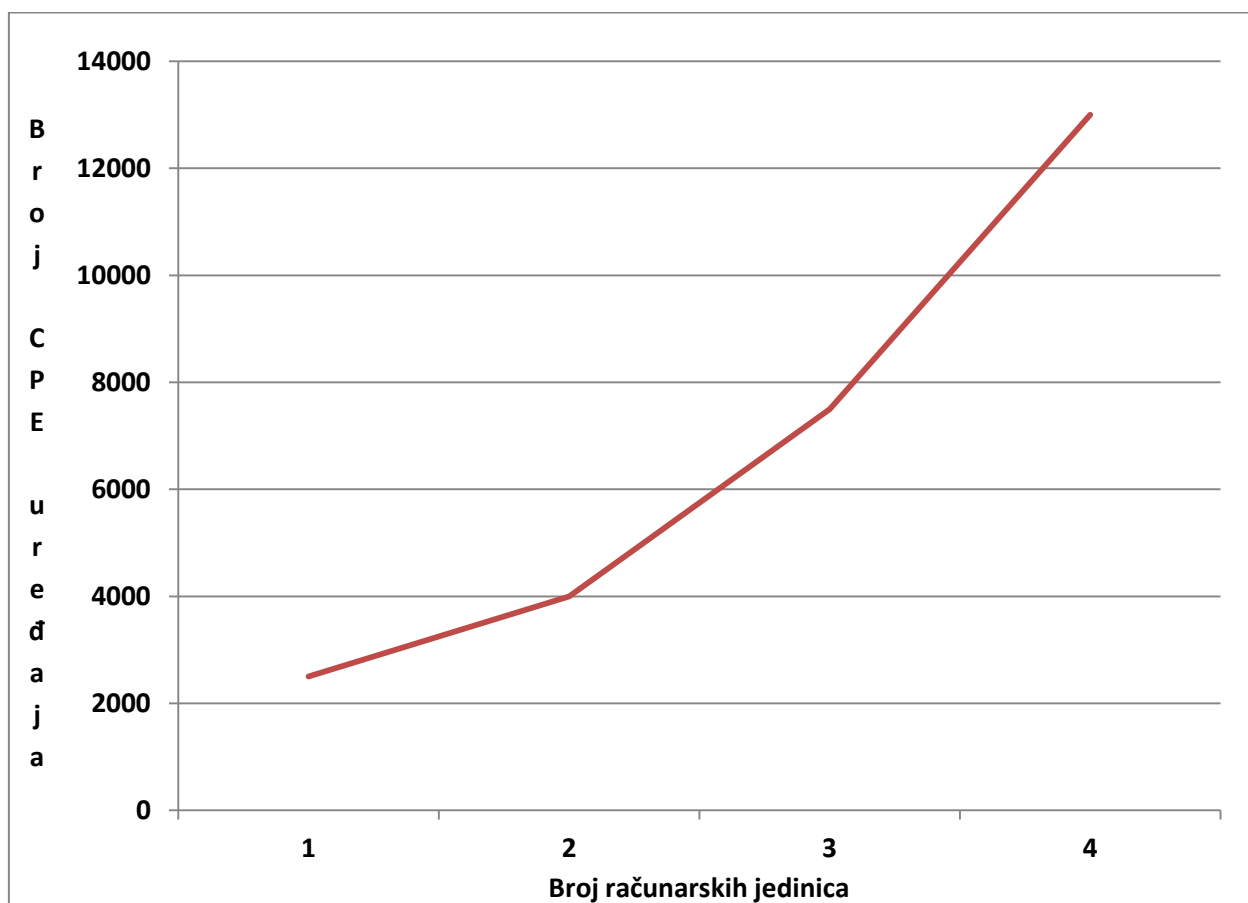
6.3 Ispitivanje skalabilnost klastera

U okviru ovog ispitivanja izvršeno je poređenje broja CPE klijenata odnosu na broj računarskih jedinica od kojih se sastoji klaster. Za potrebe ispitivanja korišćen je računarskih sistem sledećih performansi fizičke arhitekture prikazanim u Tabeli 8.

Procesor	Jedno jezgro procesora Intel Core i7-4770 sa taktom 3.40GHz
RAM memorija	2GB + 1GB memorije za razmenu (eng. Swap) DDR3 sa taktom od 1600 MHz
Trajna memorija	30GB sa brzinom obrtanja glave diska 7,200 obrtaja/minuta
Operativni sistem	Virtualni računarski sistem – VirtualBox sa Linux Ubuntu 12.04.4 LTS x86_64

Tabela 8. Performanse fizičke arhitekture pri ispitivanju skalabilnosti

Pri ispitivanju, perioda prijavljivanja CPE klijenata je bila 20 minuta. Na Slici 28 je prikazan odnos broja CPE uređaja i broja računarskih jedinica od koji se sastoji klaster.



Slika 28. Grafik skalabilnost implementiranog klastera

6.4 Ukupni rezultati

Na osnovu izvršenih ispitivanja donosi se zaključak da implementirani ACS pruža sve funkcionalnosti koje se otvaraju upotrebom TR-069 protokola. Takođe, pri ispitivanju tehničkih

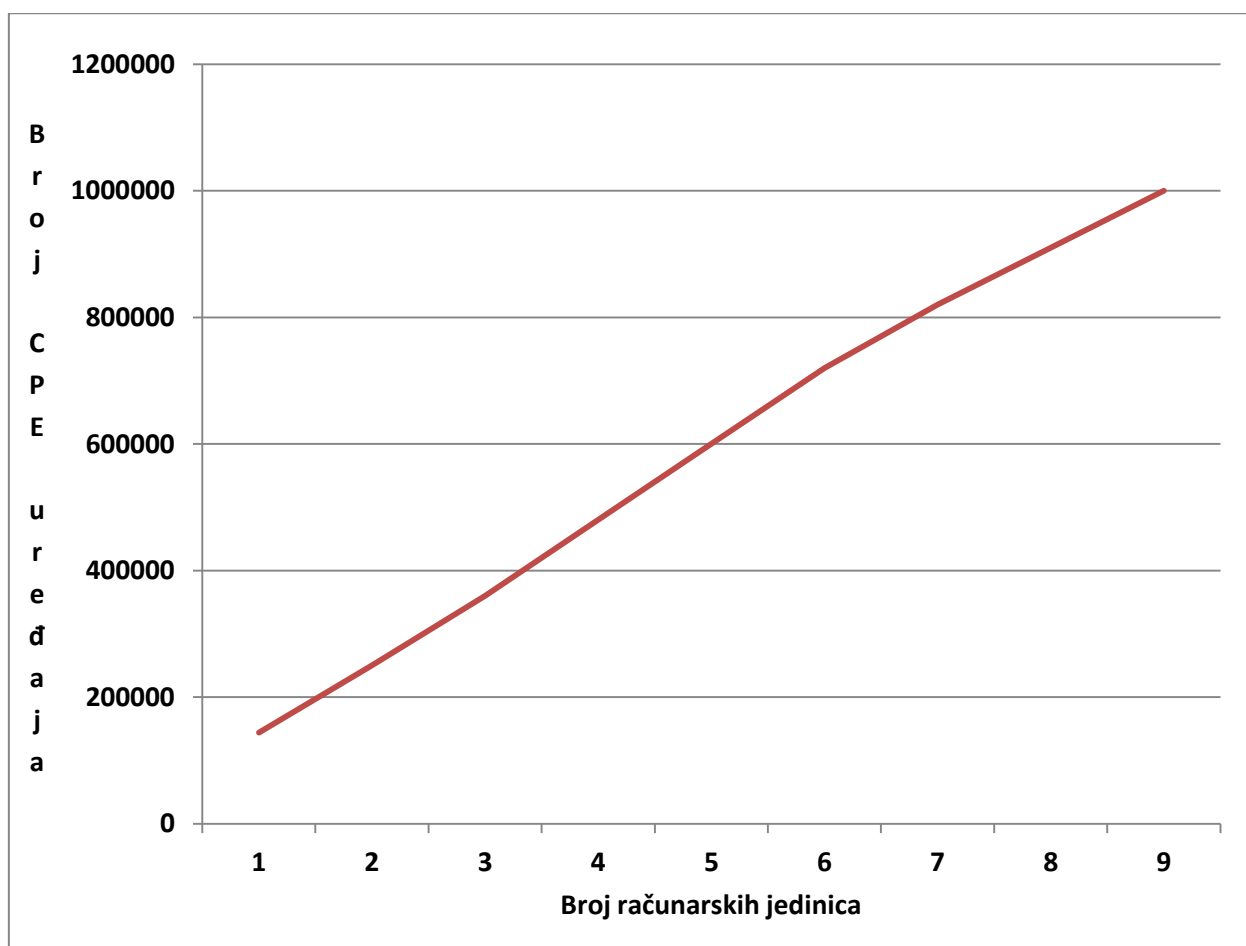
zahteva prikazani su primeri upotrebe implementiranog poslužioca koji predstavljaju mali podskup mogućnosti njegove primene. Implementacijom funkcionalnosti TR-069 protokola, stvoren je poslužilac koji pokriva osnovne principe IoT ekosistema i kao takav je primenljiv u njima.

Daljim unapređivanjem poslužioca postignut je određeni nivo skalabilnosti poslužioca u nameri da se zadovolji i najteži zahtev IoT ekosistema – opsluživanje izuzetno velikog broja krajnjih uređaja. Pri ispitivanju i poređenju neoptimizovanog i optimizovanog rešenja dokazano je da je optimizacijom postignut značajan napredak u performansama poslužioca, što će i predstavljati dalji fokus razvoja ovog projekta. Ispitivanjem klastera pokazane su krajnje mogućnosti poslužioca u datim uslovima. Na Slici 29 je prikazana teorijska skalabilnost implementiranog rešenja pri dostizanju milionskog broja CPE uređaja kada je perioda prijavljivanja dva puta nad dan (12 sati) i kada ne postoje ograničenja u korišćenju baze podataka.

TR-069 protokol u IoT ekosistemu predstavlja jedan novi izazov i biće predmet mnogih diskusija u narednim godinama. U ovom radu je dokazano da TR-069 je primenljiv u IoT ekosistemu zbog sledećih mogućnosti:

- Komunikacija poslužilaca sa korisničkim uređajima svakog tipa.
- Definisana politika praćenja rada uređaja upotrebom atributa parametara.
- Sposobnost da poslužilac dinamički menja periodu prijavljivanja uređaja i time vrši uravnoteženju opterećenja.
- Mehanizama za uspostavu veze kojim se eksplicitno aktivira komunikacija sa uređajem.

Sve nabrojane funkcionalnosti TR-069 protokola doprinose razvoju IoT ekosistema. Kao jedina mana primene protokola u gustim ekosistemima jeste veličina poruka (upotreba XML i SOAP protokola) koje se razmenjuje između poslužioca i klijentskih uređaja što unosi kašnjenje u komunikaciji i opterećenje mreže. Međutim, pitanje je da li bi sve nabrojane funkcionalnosti TR-069 protokola bile moguće da nije upotrebljen takav način kreiranja komunikacionih poruka.



Slika 29. Teorijska skalabilnost implementiranog poslužioca

7. Zaključak

U radu je prikazan skalabilan poslužilac za upravljanje i nadgledanje klijenata u IoT ekosistemu zasnovan na TR-069 protokolu. Osnovni cilj zadatka je bio implementiranje skalabilnog poslužioca koji je u stanju da opslužuje veliki broj klijentskih uređaja takođe zasnovanih na TR-069 protokolu. Skalabilnost je postignuta optimizovanjem poslužioca, razvojem klaster sistema i integrisanjem poslužioca u klaster. Funkcionalnosti koje omogućava TR-069 protokola su u potpunosti podržane i izložene su za upotrebu preko API-ja. Mogućnosti koja se otvaraju upotrebom API-ja su mnogobrojne jer se nad uređajima mogu obavljati dijagnostičke i statističke obrade, ili obavljati njihova konfiguracija. Podskup rezultata ovog rada je objavljen na skupu TELFOR 2013 [51].

U IoT ekosistemima, primena TR-069 protokola je vrlo primamljiva usled mogućnosti nagledanja uređaja nezavisno od njegove fizičke arhitekture. TR-069 protokol ostavlja mogućnost za definisanje politike praćenja i nivoa detaljnosti praćenja rada CPE uređaja. Takođe, mogućnošću da se definiše perioda prijavljivanja i dodatnog postojanja mehanizma za uspostavu veze unosi se znatno rasterećenje za mrežu. Postizanjem visokog nivoa skalabilnosti poslužioca, primenom klastera, dolazi se do zaključka da je takav poslužilac u stanju da opsluži izuzetno velike IoT ekosisteme.

Fokus daljeg razvoja je povećanje nivoa skalabilnosti, kako daljom optimizacijom programske podrške tako i proširenjem klaster sistema. Pored konfigurisanja eksternih resursa za bolji rad klastera potrebno je uzeti u obzir i upotrebu novijih i savremenijih tehnologija koja će doneti poboljšanja performansi klastera. Tako npr. upotreba *Apache Cassandra* baze podataka [52], koja ima izuzetnu linearnu skalabilnost, malo kašnjenje operacija kao i oporavak od grešaka, može uneti velika poboljšanja u klasteru.

8. Literatura

- [1] From the Internet of Computers to the Internet of Things, Friedemann Mattern, Christian Floerkemeier, 2010.
- [2] UPnP Device Architecture 1.0, <http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf>, 2014.
- [3] An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, D. Harrington et al, IETF, 2002.
- [4] TR-069 protocol Amendment 3, http://www.broadband-forum.org/technical/download/TR-069_Amendment-3.pdf, 2011.
- [5] SOAP Specification, www.w3.org/TR/soap/, 2014.
- [6] Vision and Challenges for Realising the Internet of Things, H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, 2010.
- [7] The Computer for the 21st Century, Weiser Mark, Scientific American 265(9):66–75, 1991.
- [8] The Networked Physical World. TR MIT-AUTOIDWH-001, MIT Auto-ID Center, S. Sarma, D.L. Brock, K. Ashton, 2000.
- [9] Internet of Things (IoT): A vision, architectural elements, and future directions, J. Gubbi et al, Science Direct Magazine, 2013.
- [10] IoT Startegic Research Roadmap, O.Vermesan, P. Friess, Internet of Things - Global Technological and Societal Trends, 2011.
- [11] TR-111 Applying TR-069 to Remote Management of Home Networking Devices, <http://www.broadband-forum.org/technical/download/TR-111.pdf>, 2005.
- [12] TR-069 protocol Amendment 5, http://www.broadband-forum.org/technical/download/TR-069_Amendment-5.pdf, 2013,

-
- [13] TR-106 Data Model Template for TR-069-Enabled Devices Amendment 6, http://www.broadband-forum.org/technical/download/TR-106_Amendment-6.pdf, 2011.
 - [14] TR-135 Data Model for a TR-069 Enabled STB, <http://www.broadband-forum.org/technical/download/TR-135.pdf>, 2009.
 - [15] TR-098 Data Model for a Internet Gateway Device, <http://www.broadband-forum.org/technical/download/TR-098.pdf>, 2009.
 - [16] TR-140 Data Model for Storage Service Enabled Devices, http://www.broadband-forum.org/technical/download/TR-140_Amendment-1.pdf, 2011.
 - [17] Cluster computing, R. Buyyaa, H. Jinb, T. Cortesc, Future Generation Computer Systems 18 v–viii, 2002.
 - [18] Operating Systems: Internals and Design Principles, Seventh Edition, William Stallings, 2012.
 - [19] Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers, Chee Shin Yeo, Rajkumar Buyya, 2006
 - [20] Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP and Windows 2000, M. Russinovich, D. Solomon, 2005.
 - [21] Red Hat Cluster Suite, https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Cluster_Suite_Overview/s1-rhcs-intro-CSO.html, 2014.
 - [22] The Future of Cloud Computing, Keith Jeffery, Burkhard Neidecker-Lutz, 2010.
 - [23] Amazon S3, <http://aws.amazon.com/s3/>
 - [24] SQL Azure, <http://azure.microsoft.com/en-us/services/sql-database/>
 - [25] Amazon EC2, <http://aws.amazon.com/ec2/>
 - [26] Zimory, <http://www.zimory.com/>
 - [27] ElasticHosts, <http://www.elastichosts.com/>
 - [28] Force, <http://www.salesforce.com/platform/overview/?d=70130000000lan8>
 - [29] Google App Engine, <https://cloud.google.com/products/app-engine/>
 - [30] Windows Azure, <http://azure.microsoft.com/en-us/>
 - [31] Google Docs, drive.google.com
 - [32] Salesforces CRM, <http://www.salesforce.com/eu/>
 - [33] SAP Business ByDesign, <http://www.sap.com/pc/tech/cloud/software/business-management-bydesign/overview/index.html>
 - [34] Comparing Public, Private, and Hybrid Cloud Computing Options, Judith Hurwity et al, Cloud Computing For Dummies, 2013.

- [35] Shaping Future Service Environments with the Cloud and Internet of Things: Networking Challenges and Service Evolution, Gyu Myoung Lee, Noel Crespi, Leveraging Applications of Formal Methods, Verification, and Validation Lecture Notes in Computer Science Volume 6415, pp 399-410, 2010.
- [36] MapReduce: Simplified Data Processing on Large Clusters, J. Dean, S. Ghemawat, Google Inc., 2004.
- [37] PostgreSQL, Open source object-relational database system, <http://www.postgresql.org/>
- [38] Infinispan, Open source In-Memory Data Grid, http://infinispan.org/docs/6.0.x/user_guide/user_guide.html
- [39] JBOSS AS, JavaBeans Open Source Software Application Server, <http://www.jboss.org/overview/>
- [40] JGroups, <http://www.jgroups.org/>
- [41] NGINX, <http://nginx.org/en/>
- [42] JSR 170: Content Repository for Java technology API, <https://www.jcp.org/en/jsr/detail?id=170>
- [43] JSR 283: Content Repository for Java Technology API Version 2.0, <https://jcp.org/en/jsr/detail?id=283>
- [44] Apache JackRabbit, Open source JCR solution, <http://jackrabbit.apache.org/>
- [45] ModeShape, Open source JCR solution, <http://modeshape.jboss.org/>
- [46] Java Persistence with Hibernate, Christian Bauer, Gavin King, 2006.
- [47] Realizacija podsistema za upravljanje klijentima za konfiguracioni poslužilac zasnovan na TR-069 protokolu, Mića Četković, BSc rad, izdavač Fakultet tehničkih nauka, Univerzitet u Novom Sadu, 2013.
- [48] Realizacija komunikacionog sloja za konfiguracioni poslužilac zasnovan na TR-069 protokolu, Norbet Nemet, BSc rad, izdavač Fakultet tehničkih nauka, Univerzitet u Novom Sadu, 2013.
- [49] Diagnostic CWMP client for set-top box devices, MZ Bjelica, I Papp, N Teslić, 2014.
- [50] Realizacija GWT klijent aplikacije za pristup poslužiocu za konfigurisanje STB korišćenjem MVP projektnog uzorka, Daniel Knežević, BSc rad, izdavač Fakultet tehničkih nauka, Univerzitet u Novom Sadu, 2013.
- [51] Realizacija servera za automatsku konfiguraciju i nadzor velikog broja uređaja potrošačke elektronike, M. Četković, N. Ikonić, V. Jelovac, M. Atanasievski, 2013.
- [52] Apache Cassandra, <http://cassandra.apache.org/>