



UNIVERSITETI I EVROPËS JUGLINDORE
УНИВЕРЗИТЕТ НА ЈУГОИСТОЧНА ЕВРОПА
SOUTH EAST EUROPEAN UNIVERSITY

FAKULTETI I SHKENCAVE DHE TEKNOLOGJIVE BASHKËKOHORE

STUDIME POSTDIPLOMIKE – CIKLI I DYTË

DREJTIMI: MENAXHIMI I BAZAVE TË TË DHËNAVE

Punim masteri

NoSQL - MODELIMI I TË DHËNAVE NË NJË RRETHINË TË DISTRIBUAR DHE PARALELE

KANDIDATI:

Enver Selmani

MENTORI:

Prof. Dr. Jaumin Ajdari

Tetovë, 2019

Deklaratë

Unë, Enver Selmani, deklaroj se ky punim është rezultat i hulumtimit tim origjinal si në aspektin teorik ashtu dhe në atë praktik.

Punimi në fjalë i përmbush të gjitha kërkesat sipas kriterëve të punimeve të nivelit Master, duke përfshirë: pjesën tekstuale, figurat, tabelat, bibliografinë, shtojcën si dhe referencat e duhura.

Enver Selmani



Lektoroi:

Doc.dr.Luljeta Adili-Çeliku

Abstrakt

Zhvillimi teknologjik po ndryshon çdo ditë e më shumë në secilën sferë të jetës sonë. Ky zhvillim ka ndikim në masë të madhe në shumicën e aspekteve të cilat përbën, si: zhvillimet financiare, aspektin edukativ, ekonomik, rrjeteve sociale dhe të tjera. Të gjitha aspektet të cilat gjenerojnë një numër shumë të madh të të dhënave të procesuara në kohë reale në sisteme të shpërndara dhe paralele. Ky volum i madh i të dhënave paraqet një aspekt të rëndësishëm por edhe një sfidë në ndryshimin e teknologjisë aktuale e cila merret me menaxhimin e të dhënave, e njohur si bazat e të dhënave. Pasi këto baza të të dhënave tradicionale në disa aspekte nuk janë të afta për të menaxhuar këtë fluks të madh të të dhënave, janë paraqitur alternativa të reja të cilat mundësojnë menaxhimin në mënyrë më efikase dhe më të shpejtë të të dhënave.

Këto teknologji alternative njihen si bazat e të dhënave NoSQL të cilat përdorin metodologji të ndryshme për të menaxhuar rrjedhjen e madhe të të dhënave. Qëllimi i NoSQL bazave të të dhënave është që të sigurojnë vetitë e përshkallëzueshmërisë, disponueshmërisë si dhe konsistencën e të dhënave, sidomos në një rrethinë të shpërndarë të kompjuterëve. Në mënyrë që të realizojmë qëllimet e bazave të të dhënave NoSQL gjatë punimit janë përdorur metoda të ndryshme teorike dhe praktike të cilat trajtojnë modele të ndryshme. Ndër pyetjet kryesore kur kemi të bëjmë me gjenerim shumë të madh të të dhënave është se si përdoruesit dhe aplikacionet e ndryshme softuerike mund të kenë qasje të shpejtë dhe të sigurtë të të dhënave në çdo kohë dhe vend. Rezultatet tona nga puna hulumtuese dhe eksperimentale përfshijnë procesimin e operacioneve kryesore për menaxhimin me bazat e të dhënave NoSQL. Në fund konkludohet se procesimi i të dhënave përmes NoSQL siguron një menaxhim më efikas dhe më të shpejtë bazuar në një arkitekturë të shpërndar dhe paralele.

Abstract

Technological development is changing every day more and more in every sphere of our lives. This development greatly affects many aspects of financial, educational, economic, social and other aspects of life. All of these issues generate a large number of processed data in real time in distributed and parallel systems. This large volume of data represents an important aspect but also a challenge in changing the current technology that deals with data management, known as databases. Since these traditional databases in some aspects are not capable of managing this large flow of data, new alternatives have been introduced which enable more efficient and faster data management.

These alternative technologies are known as NoSQL databases which use different methodologies to manage large data flows. The purpose of NoSQL databases is to provide the features of scalability, availability and consistency of data, especially in a distributed environment of computers. In order to accomplish the goals of NoSQL databases, during the work have been used various types of teoretical and practical approaches that deal with different models. Among the key questions when dealing with the huge amount of data is how users and various software applications can have quick and secure access to data at any time and place. Our research and experimental work outcomes include the processing of key management operations with NoSQL databases. In the end we will conclude that data processing through NoSQL provides a more efficient and faster management based on a parallell and distributed architecture.

Përmbajtja

1.	HYRJE.....	8
1.1.	Zhvillimi i bazave të të dhënave NoSQL.....	9
1.1.1.	Teorema CAP.....	10
1.1.2.	Përshkallëzueshmëria	15
1.1.3.	Disponueshmëria	16
1.1.4.	Bazat e të dhënave të cilat veprojnë në një rrethinë të shpërndarë.....	16
1.2.	Llojet e bazave të të dhënave NoSQL	18
1.2.1.	Bazat e të dhënave të llojit të bazuara në çelës / vlerë	18
	1.2.1.1. Thjeshtësia	19
	1.2.1.2. Shpejtësia	19
1.2.2.	Bazat e të dhënave të llojit të bazuar në dokumente.....	22
	1.2.2.1 MongoDB	22
1.2.3.	Bazat e të dhënave të llojit të bazuar në kolona.....	24
	1.2.3.1. Cassandra	26
1.2.4.	Bazat e të dhënave të llojit të bazuar në grafe	27
1.3.	Qëllimi i punimit.....	27
1.4.	Hipotezat.....	27
1.5.	Pyetjet kërkimore.....	28
1.6.	Metodologjia e hulumtimit	28
2.	GJENDJA MOMENTALE E PERFORMANCAVE TË BAZAVE NOSQL NË MJEDIS TE DISTRIBUARA ...	30
3.	TRANSAKSIONET OLTP	34
3.1	Transaksionet OLTP për bazat e të dhënave NoSQL.....	35
3.2	Transaksionet OLTP të cilat veprojnë në sistemet e shpërndara	37
3.3.	Transaksionet OLTP për llojet e bazave të të dhënave NoSQL	41
	3.3.1 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në Kolona	41
	3.3.2 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në Dokumente	42
	3.3.3 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në çelës/vlerë (key/value)	43
	3.3.4 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në graf	44
4.	PUNA EKSPERIMENTALE	46
4.1.	Platforma HADOOP për ndërtimin e një rrethine të shpërndarë	46

4.2.	MongoDB	48
4.3.	Metodologjia e hulumtimit.....	50
4.4.	Analiza dhe interpretimi i rezultateve	51
5.	KONKLuzionET	55
	BIBLIOGRAFIA.....	57
	Shtojca. PROCESI DOKUMENTUES PËR INSTALIMIN E PLATFORMËS	62

Lista e tabelave

Tab 1. Bazat e të dhënave të bazuar në NoSQL dhe SQL në raport me kërkesat e ndryshme (Burimi: [33]).....	37
Tab 2. Të dhënat rreth operacionit për insertim	51
Tab 3. Të dhënat rreth operacionit për lexim.....	51
Tab 4. Të dhënat rreth operacionit për përditësim	52
Tab 5. Të dhënat rreth operacionit për fshirje	53

Lista e figurave

Fig 1 Vetitë e teoremës CAP.	11
Fig 2. Teknologjitë përbërëse në sistemin e shpërndarë.....	17
Fig 3.Struktura e operimit të transaksioneve OLTP bazuar në modelin çelës/vlerë të bazave të të dhënave NoSQL (Burimi: [5]).....	20
Fig 4 Menaxhimi i “cache” memorjes bazuar në algoritmin LRU, fshirja e të dhënave në “cache” të cilat më së paku janë përdorur në kohën e fundit (Burimi: [5]).	20
Fig 5 Një koleksion i tetë serverëve të vendosur në një strukturë unazore (Burimi: [5]).	21
Fig 6.Një koleksion i tetë serverëve të vendosur në një strukturë unazore me një sistem të replikimit të katër kopjeve të serverëve (Burimi: [5]).	22
Fig 7. Strukturimi i bazave të të dhënave të llojit të bazuar në kolona (Burimi: [1]).	25
Fig 8. Strukturimi sipas bazës së të dhënave – Cassandra (Burimi: [1]).	26
Fig 9. Menaxhimi i transaksioneve në një sistem të shpërndarë (Burimi: [12]).....	38
Fig 10. Procesi i lëshimit të kërkesës transaksionale nga klienti (Burimi: [12]).....	39
Fig 11. Procesi i lëshimit të kërkesës transaksionale nga klienti (Burimi: [12]).....	40
Fig 12. Procesi i lëshimit të kërkesës transaksionale nga klienti (Burimi: [12]).....	40
Fig 13. Struktura e procesimit të transaksionit bazuar në llojin dokument (Burimi: [44])	43
Fig 14. Leximi i të dhënave duke përdorur “Hash” funksionin (Burimi: [1]).....	44
Fig 15. Të dhënat rreth klasterit Hadoop.....	47
Fig 16. Procesi i Replikimit në bazë të ndërtimi të klasterit	49
Fig 17. Grafikoni – Insertimi i të dhënave	51
Fig 18. Grafikoni – Leximi i të dhënave	52
Fig 19. Grafikoni – Përditësimi i të dhënave	52
Fig 20. Grafikoni – Leximi i të dhënave	53
Fig 21. Katër makinat virtuale me sistem operativ Ubuntu 14.04 brenda VirtualBox	62
Fig 22. Rregullimi i rrjetit brenda makinës virtuale	63
Fig 23. Gjenerimi i SSH çelësit (ang. ssh key).....	64
Fig 24. Kopjimi i çelësit publik në nyjën përkatëse	64
Fig 25. Qasja nga Master në nyjën e njohur si ‘test3’	65
Fig 26. Deaktivizimi i IPv6 protokolit.....	65

Fig 27. Java platforma e instaluar	66
Fig 28. Konfigurimi i variablave për Java.....	66
Fig 29. Instalimi dhe konfigurimi i Java platformës në njëjën ‘test4’	66
Fig 30. Instalimi i platformës Hadoop	67
Fig 31. Autorizimi i përdoruesit ‘test1’ për të pasur qasje brenda fajllave Hadoop	67
Fig 32. Konfigurimi i variablave mjedisore për Hadoop dhe Java.....	67
Fig 33. Vendosja e lokacionit për Java të instaluar brenda skriptës Hadoop	68
Fig 34. Konfigurimi dhe versioni i Hadoop-ve të instaluar	68
Fig 35. Konfigurimi i hdfs-site.xml për sistemin e file-ve të Hadoop-it (ang. HDFS).....	69
Fig 36. Konfigurimi për file-in core-site.xml.....	69
Fig 37. Formatimi i NameNode sistemit të file-ve Hadoop (ang. HDFS).....	70
Fig 38. Shënimet për nyjet punuese (ang. slave) në Master	70
Fig 39. Konfigurimi i hdfs-core.xml për sistemin e file-ve të Hadoop -së (ang. HDFS) për njëjën punuese (test3)	71
Fig 40. Konfigurimi për file-in core-site.xml në njëjën punuese ‘test3’	71
Fig 41. Lëshimi i shell skriptës kryesor për ndërtimin e klasterit Hadoop.....	72
Fig 42. Testimi i të gjitha shërbimeve të lëshuara përmes ‘jps’ komandës.....	72
Fig 43. Testimi i të gjitha shërbimeve të lëshuara përmes ‘jps’ komandës në njëjën ‘test2’ ...	73
Fig 44. Përmbledhje për të dhënat për klasterin Hadoop e ngritur	73
Fig 45. Të dhënat për sistemin e file-ve të Hadoop-së dhe nyjet punuese	74
Fig 46. Nyjet punuese (ang. slave) brenda Hadoop klasterit.....	75

1. HYRJE

“We’re here to put a dent in the universe” - Steve Jobs

“Të lësh gjurmë në univers” – Steve Jobs

Shtirja e internetit në volum të madh gjithandej ka bërë rritjen e kërkesave për të menaxhuar sasi të mëdha të të dhënave, të gjeneruar nga përdoruesit, sistemet e ndryshme, të ofruara nga kompani të ndryshme, ku bazat e të dhënave relacionale hasin në vështirësi në trajtimin e këtyre të dhënave. Si për shembull hasin probleme në modelimin e të dhënave dhe kufizimet e përshkallëzueshmërisë horizontale. Kërkesa të shumta për menaxhimin dhe ruajtjen e aplikacioneve siç janë ato për analizën e BigData-së (shumë të dhënave), rrjeteve sociale si dhe aplikacionet e fushës së intelegjencës së biznesit që gjenerojnë sasi shumë të mëdha të të dhënave (të rendit ang. peta-byte), ka çuar në zhvillimin e një lloji tjetër të bazave të të dhënave, të shpërndara, lehtë të përshkallëzuara në mënyrë horizontale, të quajtura **bazat e të dhënave NoSQL**.

Si krahasim, Bazat e të dhënave relacionale ofronjë një model të normalizuar të të dhënave dhe një zbatim strikt të vetive ACID (ang. atomicity, consistency, isolation and durability) të cilat ofrojnë konsistencë të të dhënave. Bazat e të dhënave relacionale gjithashtu, kërkojnë një definim paraprak të skemave para ruajtjes së të dhënave në bazë. Ridefinimi i një skeme në rast të paraqitjes së kërkesave të reja në të ardhmën, pas futjes së të dhënave në bazën e dhënave do të jetë shumë i vështirë për t’u kryer dhe do të shkaktojë prishjen e skemës aktuale [1]. Ndërsa, siç edhe cekëm më lart, në ditët e sotme kur kemi të bëjmë me kërkesa për procesimin e grupeve të mëdha të të dhënave, paraqitet nevoja e paqetësueshme për shtimin e tipeve të ndryshme të të dhënave për të bërë aplikacionet më efikase dhe më fleksibile [1].

Bazat e të dhënave NoSQL nuk janë shfaqur si opozitë apo kundërshtim të bazave relacionale të bazuara në modelin relational të cilat si zakonisht quhen SQL bazave të të dhënave. Këto bazat e të dhënave NoSQL lindën si nevojë e menaxhimit të grupeve (klasterimeve) të mëdha të të dhënave jo të strukturuar apo gjysmë të strukturuar, ku bazat e të dhënave relacionale vështirë ose dhe gati aspak nuk janë në gjendje për menaxhim të të dhënave të këtilla. Gjatë një konference të mbajtur në San Francisco në vitin 2009, një nga bloggerët e njohur, Eric Evans, kishte përshkruar qëllimin e lëvizjes NoSQL se duhet gjetur dhe hulumtuar alternativa të tjera që të mund të zgjidhim disa kërkesa që bazat e të dhënave relacionale nuk janë në gjendje për të zgjidhur atë. Gjithashtu, gjatë kësaj konference të mbajtur në San Francisco, zhvilluesit e NoSQL-së, duke parë që bazat e të dhënave relacionale mund të kenë kosto më të lartë dhe të jenë më të ngadalshme për të menaxhuar disa aplikacione, ata favorizonin një mënyrë më efikase dhe më të shpejtë në menaxhimin e të dhënave, të njohur si NoSQL. Qëllimi ishte që teknologjia NoSQL të zhvillojë një model që përfshinë thjeshtësinë e dizajnit dhe menaxhimit të të dhënave bazuar në: përshkallëzueshmërinë horizontale, disponueshmërinë, dhe shpejtësinë e ofrimit

të të dhënave. NoSQL-ja qëndron nga fjala angleze. “Not Only SQL”, që në përkthim kemi “Jo vetëm SQL” që do të thotë se këto baza të të dhënash janë një sistem plotësues komplementar në fushën e bazave të të dhënave, me qëllim që të përmbushin disa kërkesa të cilat modelet relacionale vështirë ose nuk mund t’i përmbushin. Bazat e të dhënave NoSQL janë kryesisht të shpërndara, të projektuar për ruajtjen e të dhënave në sasi të mëdha, duke ofruar procesimin e të dhënave në mënyrë paralele nëpër një numër të madh të serverëve [2]. Këto sisteme përdorin mekanizma dhe mjete të ndryshme kryesisht jo-SQL për të manipuluar të dhënat në bazë të tipeve të ndryshme të bazave të të dhënave jo-relacionale. Sistemet e bazave të të dhënave NoSQL u ngritën dhe u përdorën fillimisht nga kompanitë e mëdha dhe kryesore të internetit, si: Google, Facebook dhe Amazon, të cilat kishin vështirësi në menaxhimin e sasive të mëdha të të dhënave në bazat e të dhënave relacionale, (ang. SQL) paraqitnin vështirësi në këtë drejtim. Për dallim nga bazat e të dhënave relacionale, NoSQL-ja bazat e të dhënave nuk kërkojnë definimin paraprak të skemave relacionale për ruajtjen dhe menaxhimin e të dhënave dhe gjithashtu nuk mbështesin operacionin për manipulimin e të dhënave si “JOIN” (lidhja ndërmjet dy e më shumë tabelash të dhënash). Sipas një hulumtimi të bërë tri nga karakteristikat më të rëndësishme të bazave të të dhënave janë:

Përshkallëzueshmëria horizontale – që mundëson disponueshmërinë e të dhënave në nivel më të lartë në një sistem të shpërndarë, duke lejuar shpërndarjen e të dhënave nëpër shumë serverë të ndryshëm të të dhënave. Fluksi i madh i të dhënave që arrijnë nëpër serverët e ndryshëm mund të çojë në shtimin e serverëve të rinj për të shpërndarë dhe balancuar ngarkesat e të dhënave. Shumë baza të të dhënave NoSQL mundësojnë shpërndarjen e të dhënave në mënyrë automatike në serverë të ndryshëm [1].

Fleksibiliteti – Në aspektin e strukturimit të të dhënave, definimi i skemave relacionale për ruajtjen e të dhënave nuk është i nevojshëm dhe bazat e të dhënave NoSQL nuk kërkojnë një definim të tillë. Kjo lejon përdoruesit që të ruajnë të dhëna në formate dhe struktura të ndryshme brenda të njejtës tabelë të bazës së të dhënave [1].

Replikimi i të dhënave – Në procesin e replikimit, një kopje e të dhënave është shpërndarë në serverë të ndryshëm, për të arritur një disponueshmëri të lartë të të dhënave, si dhe një shpërndarje grupeve të mëdha të të dhënave në shumë serverë [1].

1.1. Zhvillimi i bazave të të dhënave NoSQL

Ndër ndodhitë kryesore që nxiti zhvillimin e teknologjise NoSQL ishte paraqitja e platformës së Ueb-it 2.0, e cila përqëllim kishte transformimin e Ueb-it 1.0 në menaxhimin e të dhënave si ato në përmbajtje e ueb-it apo të tjera në mënyrë më dinamike dhe më të shpejtë nga ato statike siç kishin qenë [3]. Këtu fillojnë të dalin në skenë krijimi i faqeve ueb në të cilat përdoruesit mund të gjenerojnë përmbajtje të ndryshme që lejojnë bashkëpunimin ndërmjet përdoruesve të ndryshëm, shkëmbimin e informacioneve të

shërbimeve të ndryshme ndërmjet tyre duke gjeneruar grupe të mëdha të të dhënave që bazat e të dhënave relacionale kanë vështirësi për t'i menaxhuar [3]. Përdorimi i skemave relacionale nuk është i përshtatshëm për aplikacionet e ueb-it që përmbajnë lloje të ndryshme të attributeve ,si: përmbajta tekstuale, fotografi, apo video dhe informacione të tjera që lehtëson performancën në qasjen e këtyre të dhënave [2]. Për shkak të rritjes shumë të madhe dhe të shpejtë të përdoruesve dhe të dhënave, ka lindur nevoja për një sistem më të aftë dhe efikas për të trajtuar këtë trend, i cili është gjithnjë e më shumë është duke u rritur [4]. Në fillim si shembull mund të marrim paraqitjen e rrjeteve sociale, si: Facebook-u, Twitter-i si dhe ato komerciale, si Amazon-i që gjeneron miliona të dhëna brenda orëve. Rrjeti sociale i lartëpërmendura, Facebook-u, gjeneron mbi 500 TB (ang. tera bytes) të të dhënave çdo ditë [4]. Gjithashtu, një rrjet tjetër social shumë i popullarizuar siç është Twitter-i, gjeneron 5,700 komente (cicërima – ang. tweets) për sekond sipas statistikave të vitit 2013 [4]. Të gjitha këto faqe interneti dinamike kërkojnë një përshkallëzueshmëri horizontale të të dhënave dhe ofrimin sa më të shpejtë të informacionit si dhe shërbimeve për përdoruesit. Shtrirja e madhe e internetit si dhe kërkesat për shërbime të ndryshme, paraqitja e platformave ueb më dinamike ka bërë që të kemi rritje dhe shpërndarje të madhe të informacionit në grupe (klasterime) të mëdha. Bazat e të dhënave NoSQL ofrojnë një mundësi shtesë në përmbushjen e këtyre kërkesave në mënyrë sa më efikase dhe të shpejtë.

Teknologjia NoSQL garanton përshkallëzueshmërinë horizontale përmes sharding (copëzimit) automatik të të dhënave në nyje të bazave të të dhënave, duke nënkuptuar shpërndarjen e të dhënave ndërmjet shumë serverëve, duke përdorur qasjen shared-nothing (asgjë e përbashkët). Kjo veti garanton përmirësimin e performancës duke shtuar serverë të tjerë në pëmbushjen e kërkesave shtesë [4] [1].

Por, duhet pasur parasysh se bazat e të dhënave në një sistem të shpërndarë paraqesin vështirësi në menaxhimin e bazave të të dhënave, në kuptimin e ruajtjes së disponueshmërisë dhe konsistencës së të dhënave. Kjo pasqyrohet edhe te menaxhimi i transaksioneve, se manipulimi i të dhënave në një sistem të shpërndarë është më i vështirë se në një sistem të centralizuar. Për të lehtësuar këtë proces sa i përket teknologjisë NoSQL, është rekomanduar përdorimi i një modeli të programimit që trajton grupe të mëdha të të dhënave në një sistem të shpërndarë dhe paralel i quajtur MapReduce i cili lejon përshkallëzueshmërinë ndërmjet qindra apo mijëra serverëve. Ndarja e të dhënave në serverët e tjerë bëhet në mënyrë automatike. Përparësia qëndron në atë se në rast të dështimit të ndonjë serveri duke kryer ndonjë nga operacionet për marrjen dhe përditësimin e të dhënave, kemi kopjet në dispozicion nga serverët e tjerë që mundësojnë një ruajtje të konsistencës së të dhënave.

1.1.1. Teorema CAP

Teorema CAP, për herë të parë e përmendur nga Eric Brewer, do të thotë se në një

sistem me bazat e të dhënave të shpërndara është e pa mundur që njëkohësisht të ketë qëndrueshmëri (konsistencë), disponueshmëri dhe tolerim gjatë ndarjes (ang. partition tolerance) [3].

Teorema CAP garanton kryerjen e transaksioneve OLTP në mënyrë efikase dhe të shpejtë, duke ofruar përshkallëzueshmërinë kur kemi grupe të mëdha të të dhënave. Teorema CAP mundëson ruajtjen e konsistencës apo ruajtjen e të dhënave në një rrethinë të shpërndarë. Duke pasur parasysh kopjimin e të dhënave (replikimin) në më shumë serverë të të dhënave, që në rast deshtimi të ndonjë serveri të caktuar, ai ruan disponueshmërinë dhe qëndrueshmërinë (ang. availability and durability) e sistemit [3].

Teorema CAP garanton disponueshmërinë e lartë (ang. high availability) duke përdorur protokole të ndryshme të replikimit dhe sinkronizimit të të dhënave në serverë të ndryshëm. Kjo mundëson krijimin e infrastrukturës së shpërndarë me qasje më efikase dhe fleksibilitet në menaxhimin e grupeve të mëdha të të dhënave.

Teorema CAP erdhi si nevojë për sigurimin e disponueshmërisë së lartë (ang. high availability) në sistemet e shpërndara. Akronimi CAP qëndron

- **Consistency** – të gjithë përdoruesit shohin të njëjtin verzion të të dhënave, në të gjithë serverët e shpërndara, ruatja e kopjeve konsistente të të dhënave në server të ndryshëm [2].
- **Availability** – të gjithë klientët mundet të gjejnë gjithmonë të paktën një kopje të të dhënave të kërkuara, madje edhe nëse disa nga serverët brenda koleksionit dështojnë për arsye të ndryshme [2]. Do të thotë që ekziston gjithmonë një përgjigje për ndonjë përzgjedhës (ang. query).
- **Partition-tolerance** - ka kuptimin që nëse një rrjet që lidhë dy ose më shumë serverë të bazës së të dhënave dështon, pjesa tjetër e serverëve do të jenë në dispozicion me të dhënat konsistente.

Teorema CAP kërkon që minimum dy nga tri kërkesat e dhëna të jenë të arritur plotësisht në të njëjtën kohë [2].

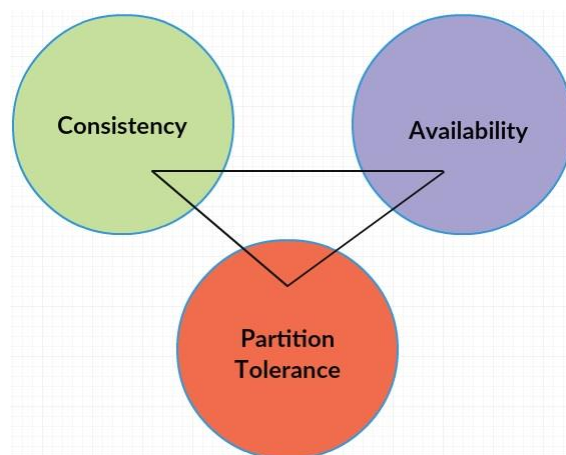


Fig 1 Vetitë e teoremës CAP.

Një nga vetitë e teoremës CAP e shkruar edhe në akronomin e saj, tolerimi i ndarjes (ang. partition tolerance) apo particionimit, ka të bëjë me tolerancën ndaj ndarjes së rrjetit apo shkeputjes së komunikimit ndaj dy serverëve të ndryshëm të të dhënave. Kjo do të thotë se në rast se dështon komunikimi ndërmjet serverëve gjatë shkëmbimit të mesazheve të ndryshme ndërmjet tyre, sistemi dhe grupet e mëdha të të dhënave vazhdojnë operimin. K duke pasur parasysh edhe shtimin e nyjeve të reja të të dhënave në sistem apo fshirjen e njërës aktuale. Nga tri kërkesat e teoremës CAP kuptojmë se ndërtuesit e aplikacioneve të bazës së të dhënave kanë për t'u marrë më shumë me një kompromis ndërmjet konsistencës dhe disponueshmërisë se sa me çështjet e tolerimit të rrjetit. Nganjëherë kur dëshirojmë të arrijmë koeksistencën përmes një protokollit të quajtur “two-phase commit” mund të humbim disponueshmërinë e të dhënave. “Two-phase commit” protokollit kryen dy hapa, atë të ruajtjes së të dhënave në serverin kryesor dhe ruajtjen e një kopje në një server rezervë të të dhënave, që në rast se serveri kryesor dështon për ndonjë arsye atëherë serveri rezervë vihet në dispozicion. Shkaku pse “two-phase commit” protokollit e dëmton vetinë e disponueshmërisë është se përderisa ky protokoll është duke u ekzekutuar, përdoruesit e tjerë që kërkojnë të dhëna dhe lëshojnë një pyetje (ang. query) janë bllokuar.

Të dhënat e përditësuar (ang. updated) nuk janë në dispozicion deri sa “two-phase commit” protokollit të përfundojë në kryerjen e operacioneve të tij. Për dallim nga bazat e të dhënave relacionale të cilat përmbajnë vetitë e rëndësishme të ruajtjes së integritetit të të dhënave dhe konsistencën e transaksioneve, të quajtur ACID, bazat e të dhënave NoSQL përmbajnë disa veti të cilat pak a shumë na mundësojnë një kompromis dhe bashkëveprim ndërmjet disponueshmërisë dhe koeksistencës së të dhënave.

Pra, këta paraqiten përmes akronimit BASE (ang. Basically Available, Soft State, Eventually Consistent)

- **Basically Available** – ka kuptimin që nëse ndodh ndonjë dështim në disa pjesë të sistemit të shpërndarë (në disa serverë), pjesa tjetër e sistemit vazhdon funksionimin. Pra, pavarësisht dështimit të disa serverëve brenda sistemit të shpërndarë, sistemi është i qasshëm në çdo kohë për të procesuar kërkesat e përdoruesve. Kjo veti arrihet duke ofruar nga NoSQL më tepër kopje të të dhënave në serverë të ndryshëm [5].
- **Soft State** – do të thotë se të dhënat që qëndronjë për një kohë të gjatë në serverë, mund të jenë mbishkruar me të dhënat më të fundit [5].
- **Eventually Consistent** – ndonjëherë bazat e të dhënave mund të jenë në një gjendje jo konsistente (të qëndrueshme), ku shumë kopje të të dhënave të ruajtura në shumë server mund të mos jenë konsistente për një periudhë të shkurtër kohore. Kjo do të thotë se kur një përdorues përditëson një kopje të të dhënave, kopjet e tjera (të vendosur në serverë të ndryshëm) vazhdojnë për të pasur versionin e vjetër e të dhënave.

Raste të tilla kur kemi jo konsistencë të të dhënave në shume serverë, ndodhin në bazë të disa faktorëve, siç janë: ngarkesa e të dhënave në sistem dhe shpejtësia e rrjetit për të përditësuar të dhënat. Si shembull mund të shqyrtojmë një bazë të të dhënave që ruan tri kopje të të dhënave në serverë të ndryshëm. Një përdorues dëshiron për të përditësuar (ang. updated) adresën e tij të vendbanimit në një serverë. Sistemi i menaxhimit i bazave të të dhënave NoSQL automatikisht përditëson dy kopjet e tjera. Supozojmë që një nga kopjet tjera për t'u përditësuar është në një server në të njëjtin rrjet lokale (ang. local area network), kështu që përditësimi i të dhënave ndodh shpejt. Ndërsa një serverë tjetër i tretë ndodhet në një qendër të të dhënave (ang. data center), me mijëra kilometra larg nga serverët kryesor, dhe këtu realisht mund të ketë një vonesë kohore deri në përditësimin e kopjes së tretë. Përderisa ky server është duke u përditësuar, imagjinojmë se një përdorues dëshiron të marrë të dhëna nga serveri i tretë (dhe lëshon një pyetje), atëherë ai mund të marrë adresën e vjetër të dhënë. Ndërsa dikush tjetër që kërkon të njëjtën të dhënë nga serveri i parë (kryesor) do të marrë si përgjigje adresën e përditësuar më parë. Pra, nga kjo kuptojmë se kemi një humbje të konsistencës së të dhënave në një sistem të shpërndarë, që mund ta bëjë sistemin jo efikas dhe të papërdorshëm për përdoruesit [5].

Teorema CAP në një sistem të shpërndarë dhe paralel të procesimit të të dhënave, në grupe të mëdha ofron përshkallëzueshmëri të lartë duke lejuar serverë të ndryshëm të të dhënave të veprojnë në vende të ndryshme në vend të përshkallëzueshmërisë vertikale. Përshkallëzueshmëria në nivelin më të lartë ka kuptimin e krijimit të një sistemi sa më efikas dhe të shpejtë, përmes shtimit të serverëve të tjerë brenda koleksionit. Kjo nënkupton shpërndarjen e grupeve të mëdha të të dhënave në mënyrë të balancuar në serverët e ndryshëm. Gjithashtu, mund të krijohen kopje të të dhënave për secilin server për të siguruar disponueshmërinë e lartë në të dhënat. Përshkallëzueshmëria horizontale si pjesë e NoSQL-së mundëson reduktimin e kostos operacionale për të kryer sa më shpejt dhe në mënyrë sa më efikase transaksionet OLTP.

Teorema CAP përveç përshkallëzueshmërisë horizontale, garanton një sistem të qëndrueshëm, efikas dhe të shpejtë të menaxhimit dhe ofrimit të të dhënave bazuar në një rrethinë të shpërndarë dhe paralele në grupe të mëdha të të dhënave. Varësisht nga kërkesat për ndërtimin e modelit të të dhënave, mund të zgjedhim se cilat nga shërbimet e teoremës CAP të përdorim.

Modelimi i të dhënave NoSQL përdor particionimin (ndarjen) e të dhënave në serverë të ndryshëm duke mundësuar kopjimin e të njëjtave të dhëna në më shumë serverë. Kjo siguron disponueshmërinë e të dhënave në çdo rrethanë pavarësisht dështimeve në sistem.

Modelimi i të dhënave NoSQL ofron mënyra të ndryshme në qasjen e të dhënave. Varësisht nga përdorimi i modelit të bazuar në NoSQL, si psh. modeli i bazuar në çelës/vlerë përdorë thajse të njëjtën qasje të të dhënave sikur bazat e të dhënave tradicionale SQL (ang. structured query language). Gjithashtu, kemi metodat e tjera të transaksioneve OLTP

të cilat veprojnë në mënyrë të shpërndarë, që u qasën një grupi të serverëve të caktuar. Rezultatet nga të gjithë serverët e kërkuar kthehen si rezultat i vetëm për përdoruesin e caktuar.

Një mënyrë tjetër për qasjen e të dhënave paraqet edhe modeli i pemës B+ (ang. B+ tree). Pema B+ përmban një nyje kryesore (ang. root) si dhe nyjet e tjera të brendshme të cilat janë të lidhur me njëri – tjetrin në nivelet me të ulëta (në formë piramidale). Secila nyje (e ashtuquajtur prindër) përmban një numër identifikues për nyjën tjetër në nivel më të ulët (në nyjet fëmijë). Kështu që për të gjetur nyjën e cila përmban të dhënat e kërkuara, duhet të kalojmë përmes secilës nyje brenda pemës B+. Si model tjetër për pyetësorët (ang. querying) vlen të përmendet edhe modeli i pemës B+ (ang. B+tree) i shpërndarë. Përmes një nyjeje kryesore të pemës së quajtur si rrënja e pemës (ang. root) e cila përmban numrin identifikues të nyjes fëmijë të saj në një nivel më poshtë, dhe kështu me rradhë ku secila nyje përmban një identifikues të tillë për nyjën tjetër në nivel më të ulët në pemë, shkojmë deri tek gjetja e nyjës së gjethit e cila përmban informacionin e kërkuar nga përdoruesi që ndeshet me pyetësin (ang. query) e caktuar.

Bazat e të dhënave NoSQL nuk zëvendësojnë bazat e të dhënave SQL relacionale, por vetëm përmbushin me lloje të ndryshme të nevojave bazuar në rritjen e kërkesave tjera, të cilat bazat e të dhënave relacionale nuk i plotësojnë ato, prandaj nevojitet një qasje si kjo e bazave të të dhënave NoSQL . Për të arritur që t'i përmbushim këto nevoja, si: përshkallëzueshmëri, fleksibilitet, kontroll të qasjes dhe disponueshmëri të të dhënave, është e nevojshme të bëhet dizajnimi (projektimi) i një sistemi të menaxhimit të të dhënave që funksionon në shumë serverë të ndryshëm, pra një sistem i shpërndar.

Konzistenca si një veti e rëndësishme brenda bazave të të dhënave jo relacionale përdoret për të përshkruar gjendjen e kopjeve të të dhënave në sistemet e shpërndara. Për shembull, nëse kemi dy serverë brenda një sistemi dhe secili mban kopjet e të dhënave rreth një produkti të caktuar, atëherë themi se sistemi është konsistent nëse këta serverë kanë kopje të njëjta të të dhënave (do të thotë se duhet pasur përputhshmëri të të dhënave) [5]. Rëndësia e kësaj qëndron në faktin se, nëse dy përdorues lexojnë të dhëna të ndryshme, për kërkesën e njëjtë, nga dy serverët që mbajnë të njëjtën kopje të të dhënave, atëherë kemi humbje të vetisë së koeksistencës, çka e bën sistemin jo efikas dhe të mos jetë konsistent. Bazat e të dhënave NoSQL , për të garantuar vetinë e koeksistencës, shpesh përdorin konceptin e quajtur “Quorums” kur veprojnë me operacionet për leximin (ang. read) dhe shkrimin (ang. write) e të dhënave. Një “Quorum” paraqet numrin, respektivisht bashkësinë e serverëve që duhet përgjigjur për një kërkesë për lexim (ang. read) apo shkrim (ang. write). Kjo duhet të na sigurojë se të gjithë serverët përgjegjës duhet të kenë një sinkrozinim të operacioneve, për shembull në rast se një përdorues ‘A’ shkruan (ang. write) disa të dhëna në një serverë të caktuar, serverët tjerë përgjegjës për lexim (ang. read) dhe shkrim (ang. write) që mbajnë kopjet e atyre të dhënave nuk duhet të pranojnë kërkesa nga

përdoruesi tjetër B për të shkruar tek ato të dhëna deri sa përdoruesi 'A' të mbarojë operacionin me sukses.

Durability (qëndrueshmëria) – vetia e ruajtjes së një kopje korrekte të të dhënave për një periudhë të gjatë të kohës. Një operacion për përditësimin e të dhënave (ang. write) konsiderohet të përfundojë kur një numër minimal i kopjeve ndodhet në ruajtjen e përhershme (persistente) [5].

Bazat e të dhënave NoSQL mund të veprojnë në grupe të ndryshme të serverëve të cilat njihen si sistemet e shpërndara, ku aplikacionet mund të veprojnë në më shumë serverë duke koordinuar operacionet ndërmjet tyre [5]. Këto sisteme të shpërndara ndihmojnë në përmirësimin e vetive të shkallëzueshmërisë dhe disponueshmërisë, por e kanë më të vështirë për të siguruar konzistencën e të dhënave ndërmjet shumë serverëve. Kjo ndodh përshkak të dështimeve në rrjetin e komunikimit apo në anomalitë tjera që rrjedhin nga qasja e të dhënave në këta serverë.

1.1.2. Përshkallëzueshmëria

-Faqet e internetit e nivelit më të lartë dallohen nga përshkallëzueshmëria e madhe, gjendje latente më të ulët, aftësia për të rritur kapacitetin e bazës së të dhënave në bazë të kërkesave si dhe një model i thjeshtë i programimit [6]. Përshkallëzueshmëria paraqet aftësinë e sistemit për të shtuar më shumë resurse kompjuterike për të përballuar një fluks të madh të të dhënave, kërkesat në të dhënat si dhe rritjen e përdoruesve të ndryshëm. Në menaxhimin me të dhënat kemi dy lloje të përshkallëzueshmërisë dhe atë: përshkallëzueshmëria vertikale (ang. scale up) dhe përshkallëzueshmëria horizontale (ang. scale out). Përshkallëzimi vertikal do të thotë se ne mund të ndërrojmë një server që nuk përmbush kërkesat aktuale, me një server tjetër me efikas dhe më të shpejtë në menaxhimin e bazës së të dhënave [3]. Serveri i ndërruar do të përmban paisjet harduerike me kapacitet dhe shpejtësi më të madhe se sa ai paraprak. Gjithashtu, përshkallëzueshmëria vertikale përfshin jo vetëm ndërrimin e një serveri me një server tjetër më "efikas", por edhe ndërrimin e pjesëve kompjuterike të serverit aktual si memorja, hard-disku dhe paisje të tjera më efikase, të shpejta dhe kapacitet më të madh në menaxhimin e të dhënave në masë të madhe.

Ndërsa përshkallëzimi horizontal paraqet shtimin apo heqjen e serverëve të tjerë në bazë të kërkesave të nevojshme. Përshkallëzimi horizontal është më fleksibil se përshkallëzimi vertikal [5] për arsye se përshkallëzimi vertikal kërkon bartjen e tërë bazës së të dhënave nga një server te një tjetër, nëse e ndërrojmë serverin me një më efikas. Gjithashtu, mangësia tjetër qëndron se nëse do të ngritnim serverin aktual me disa resurse shtesë, do të bënte që baza e të dhënave aktuale të mos jetë në dispozicion për një kohë. Këto arsye do të shkaktonin rënien e disponueshmërisë së sistemit të shpërndarë, për të qenë gjithmonë i gatshëm për t'iu përgjigjur kërkesave. Pra, përshkallëzimi horizontal

kryhet në atë mënyrë që shtimi apo heqja e serverëve të caktuar nga sistemi për të përmbushur kërkesat e caktuara nuk do të pamundësojë disponueshmërinë e sistemit [3]. Përshkallëzimi horizontal i të dhënave na mundëson ndarjen e detyrave të cilat mund të procesohen njëkohësisht në shumë serverë. Kompleksiteti i menaxhimit të të dhënave përmes shpërndarjes horizontale kërkon gjuhë programuese më të specializuara. Në këtë rast, një platformë strukturore si MapReduce mundëson një procesim të të dhënave në mënyrë më efikase në një sistem të shpërndarë dhe të përshkallëzuar [6].

1.1.3. Disponueshmëria

Përdoruesit kërkojnë që faqja apo aplikacionet e internetit të jetë në dispozicion sa herë që ata dëshirojnë të përdorin (të kenë qasje në të). Të dhënat të cilat nuk janë në dispozicion gjatë tërë kohës duke pasur parasysh dështimet e shumta, do të bënin që përdoruesit të neglizhonin të tilla aplikacione. Bazat e të dhënave NoSQL janë projektuar për të pasur në dispozicion shumë server me kosto të ulët. Kështu që kur një server dështon ose është nxjerrë nga përdorimi, atëherë serverët e tjerë në koleksion marrin tërë përgjegjësinë për punën e serverit “të dështuar” ashtu që aplikacioni të jetë në dispozicion çdoherë. Krahasojmë këtë lloj të sapo përmendur më lart për disponueshmërinë (ang. availability) me një rast kur një server kryesor (primar) ka vetëm një kopje të serveri rezervë (ang. backup). Serveri “back-up” mban një kopje rezervë të të dhënave të kopjuara nga serveri kryesor (primar) [5]. Në rast se ndodh që serveri kryesor të mos jetë në dispozicion apo të dështojë, atëherë serveri rezervë do të marrë përgjegjësinë në menaxhimin e të dhënave. Kjo mund të jetë një zgjidhje jo efikase për arsye sepse serveri rezervë (ang. backup) nuk është në dispozicion gjatë kohës së dështimit të serverit aktual, por vihet në dispozicion pas dështimit, e cila do të marrë kohë sa të vihet në dispozicion. Pra, sistemi tradicional i ruajtjes së kopjes rezervë (ang. backup) nuk mundëson disponueshmërinë e të dhënave gjatë gjithë kohës

1.1.4. Bazat e të dhënave të cilat veprojnë në një rrethinë të shpërndarë

Baza të të dhënave të shpërndara përbëjnë një koleksion të shumë bazave të të dhënave të cilat janë të ndërlidhura logjikisht me njëra-tjetrën të shpërndarë mbi një rrjet kompjuterik [7]. Ky sistem i shpërndarë mbështetet në disa teknologji, si: teknologjia e bazave të të dhënave, teknologjia e rrjetit kompjuterik si dhe teknologjia e komunikimit të të dhënave [7].

Bazat e të dhënave të shpërndara janë vendosur në më shumë kompjuterë në të njëjtin lokacion fizik apo mund të vendosen edhe në lokacione të ndryshme fizike (shpesh në distanca të mëdha) të ndërlidhur me një rrjet kompjuterik. Sistemet e shpërndara për dallim

nga ato paralele janë krejtësisht heterogjene si në aspektin harduerik, softuerik dhe të të dhënave [7] [8].

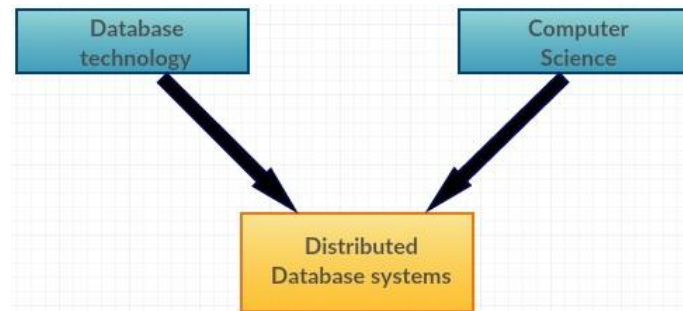


Fig 2. Teknologjitë përbërëse në sistemin e shpërndarë.

Disa nga kriteret që dallojnë bazat e të dhënave të shpërndara janë:

- 1) Bazat e të dhënave të shpërndara duhet të jenë të lidhura mbi një rrjet kompjuterik të komunikimit, për të bartur komandat operacionale si dhe të dhënat ndërmjet serverëve [7].
- 2) Bazat e të dhënave të vendosura në vende të ndryshme duhet të jenë logjikisht të lidhura [7]. Si shembull mund të jenë: ndarja skematike e bazave të të dhënave në serverë të ndryshëm, si dhe ndarja e tabelave të të dhënave në serverë të ndryshëm, të cilët përmbajnë volum të madh të të dhënave.
- 3) Siç përmendëm edhe më lart se bazat e të dhënave të cilat veprojnë në sistem të shpërndarë nuk është e nevojshme të jenë homogjene apo të njejta në aspektin harduerik, softuerik si dhe në të dhënat [7].

Çështjet rreth ndërtimit të rrjetit të komunikimit ndërmjet serverëve duhet të jenë të padukshme për përdoruesit (ang. end user) [7]. Por është shumë e rëndësishme të ceket se në ndërtimin e rrjetit të komunikimit, lloji dhe topologjia e rrjeteve kompjuterike mund të kenë ndikim të rëndësishëm në operimin efikas të një sistemi të shpërndarë të bazave të të dhënave [7]. Ndikimet mund të përfshijnë performancën e procesimit të transaksioneve të shpërndara si dhe projektimin e bazave të të dhënave të shpërndara [7].

Ruajtja e të dhënave në serverë të ndryshëm dhe bazat e të dhënave të cilat veprojnë në serverë të shpërndarë do të përmirësojnë performancën për transaksionet e lëshuara. Në vend të procesimit të transaksioneve në një server të vetëm, procesimi i transaksioneve në më shumë serverë do të përmirësojë performancën për përdoruesit dhe vetë sistemin.

Një nga faktorët më të rëndësishëm të sistemeve të shpërndara për të cilat qëndrojnë bazat e të dhënave janë:

Replikimi i të dhënave: Procesi i replikimit përfshinë ndarjen e të dhënave të të njëjtit nivel për të gjithë serverët. Të dhënat janë replikuar nga një bazë e të dhënave në një server te një bazë e të dhënave në serverin tjetër [9].

Disa nga disavantazhet e sisteme të shpërndara kanë të bëjnë me kompleksitetin e ndëtimin të tyre, koston e lartë, çështjet e sigurisë, verifikimin e integritetit të të dhënave, shpeshherë nga mungesa e eksperiencës dhe standardeve të sistemeve të tilla [8].

1.2. Llojet e bazave të të dhënave NoSQL

Në vijim do të përshkruajmë dhe analizojmë llojet e bazave të të dhënave NoSQL. Bazat e të dhënave NoSQL të cilat do t'i trajtojmë janë: lloji i bazave të të dhënave të bazuar në çelës/vlerë (ang. key/value), lloji i bazuar në dokumente, kolona si dhe në grafe.

1.2.1. Bazat e të dhënave të llojit të bazuara në çelës / vlerë

Modeli i ruajtjes çelës/vlerë bazohet në qasjen e të dhënave përmes çelës/vlerës së çiftit (ang. pair) të të dhënave duke iu qasur vlerës së të dhënës përmes çelësit (ang. key) të caktuar. Çelës/vlera e bazës së të dhënave është e strukturuar në tabela të thjeshta “imagjinare” që përmbajnë çiftet çelës/vlerë, ku secili çift është i shoqëruar me një vlerë e cila mund të përmbajë shumë attribute. Bazat e të dhënave të llojit çelës/vlerë mund të ndahen në grupe të caktuara (ang. bucket), ku secila përmban një çelës (ang. key) të veçantë. Pra, bazat e të dhënave të llojit të bazuar në çelës/vlerë mundësojnë krijimin e koleksioneve të të dhënave për të lehtësuar manipulimin me të dhënat.

Çelës/vlera ruajtja (ang. store) ekziston një kohë të gjatë dhe është influencuar nga DynamoDB Amazoni [3].

Ruajtja DynamoDB (ang. store) përdoret nga Amazoni për qëllime të ndryshme, duke ofruar algoritmin e ndarjes (particionimit) të të dhënave ndërmjet vendeve të ndryshme të ruajtjes. Gjithashtu, DynamoDB mundëson edhe replikimin e të dhënave nëpërmjet nyjeve të ndryshme për të ruajtur disponueshmërinë dhe qëndrueshmërinë e të dhënave (ang. availability and durability). DynamoDB-ja propagon edhe shtimin e nyjeve të reja në një rrethinë të shpërndarë si dhe heqjen e nyjeve të të dhënave të caktuara brenda sistemit të shpërndarë dhe paralel. Te shtimi i nyjeve të reja problemi që ndodh te DynamoDB-ja është koordinimi me nyjet e tjera të të dhënave (brenda sistemit), për të ruajtur vetitë e caktuara [3]. Sa i përket pjesës së trajtimit të dështimeve të nyjeve (serverëve) të të dhënave që mund të ndodhin në një rrethinë të shpërndarë, detektimi i dështimit të këtyre nyjeve bëhet përmes nyjeve tjera individuale në rrethinë të cilave ju pamundësohet dërgimi i kërkesave

te këto nyje [3]. DynamoDB-ja funksionon më shumë se sa një qendër e të dhënave (ang. data center), ku në esencë në një sistem të tillë të ngritur, serverët (ang. storage nodes) janë të shpërndarë në më shumë lokacione të ruajtjes së të dhënave. Shpërndarja e serverëve, siguron mbajtjen e kopjeve (replikimin) të të dhënave, e cila mundëson disponueshmërinë e sistemit dhe eliminimin e efekteve të dështimit të serverëve [3] [1].

Por, për të arritur qëndrueshmërinë e një sistemi të shpërndarë në rast dështimi përmes kopjimit të të dhënave në nyje apo serverë të ndryshëm të shpërndarë në qendra të të dhënave të ndryshme (ang. data centers), DynamoDB-ja përdor një protokoll të quajtur anti-entropy për sinkronizimin e kopjeve të të dhënave, që në rast dështimi të një kopje të vihen në dispozicion kopja tjetër për të siguruar qëndrueshmërinë (ang. durability).

1.2.1.1. Thjeshtësia

Bazat e të dhënave çelës/vlerë përmbajnë një model të thjeshtë dhe fleksibil të të dhënave. Në mënyrë strukturale një bazë e të dhënave ndahet në të ashtuquajtur “bucket” (ang. namespace), të cilat përmbajnë çiftet çelës/vlerë dhe një identifikues (ang. key) për “bucket”. Për të kryer një operacion për marrjen(selektimin) e një të dhëne nga një “bucket” i caktuar, duhet t’u qasemi përmes emrit të “bucket” dhe key (çelësit) të vlerës së kërkuar dhe si rezultat i kësaj bazat e të dhënave do të kthejnë një vlerë të caktuar. Ndërsa kur do të bëjmë përditësimin e ndonjë të dhëne brenda “bucket” të caktuar, përveç specifikimit të emrit të “bucket” (ang. namespace), duhet specifikuar çelësin e vlerës së caktuar si dhe vlera e re që do të përditësohet.

1.2.1.2. Shpejtësia

Qëllimi i ekspertëve dhe ofruesve (shitësve) të bazave të të dhënave është ofrimi i mjeteve të cilat do të përmirësonin operacionet për menaxhim me bazat e të dhënave. Bazat e të dhënave çelës/vlerë janë të njohura për shpejtësinë e tyre. Me një strukturë të thjeshtë të grupeve të të dhënave, bazat e të dhënave çelës/vlerë ofrojnë opecaione që mundësojnë “throughput” më të lartë dhe qasje më intensive në të dhënat. Për të mundësuar qasje të shpejtë në manipulimin e të dhënave, bazat e të dhënave çelës/vlerë ofrojnë një mënyrë për t’u qasur të dhënave nga memorja e përkohëshme (ang. RAM memory) [5] [1]. Përshkak se qasja dhe manipulimi i të dhënave përmes RAM-memorjes është shumë më e shpejtë se sa përmes diskut (memorjes me ruajtje persistente). Në bazat e të dhënave çelës/vlerë, kur një program dëshiron të përditësojë (ang. update) një vlerë të caktuar (përmes çelësit të caktuar) nga një “bucket”, së pari kjo vlerë përditësohet (ang. update) në memorje RAM dhe pastaj dërgohet një mesazh informues për programin apo klientin që vlera është përditësuar [5] [10]. Pas kësaj përderisa sistemi mund të vazhdojë me pranimin e kërkesave të tjera për qasje në të dhënave, bazat e të dhënave çelës/vlerë mund të shkruajnë vlerën aktualisht të përditësuar në disk, siç është ilustruar si më poshtë:

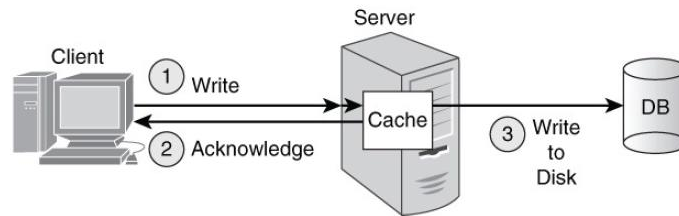


Fig 3. Struktura e operimit të transaksioneve OLTP bazuar në modelin çelës/vlerë të bazave të të dhënave NoSQL (Burimi: [5]).

Kur madhësia e bazave të të dhënave kalon madhësinë e RAM – ekzistojnë mënyra për menaxhimin e të dhënave në memorje [5]. Një nga mënyrat e përdorura është edhe algoritmi i njohur si LRU (ang. least recently used) – që do të thotë se rreshti më i fundit në memorje që më së paku është përdorur (të dhënat të përdorura më së paku në cache memorjen) apo nuk kanë qenë përdorur për një kohë, mund të jenë fshirë nga “cache” memorja. Siç është paraqitur në ilustrimin më poshtë [5]:

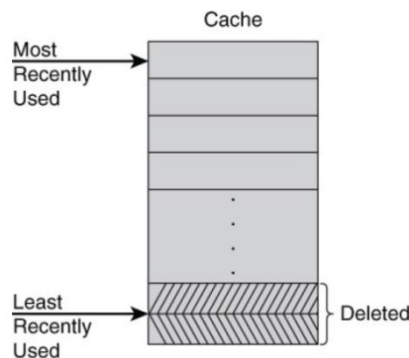


Fig 4 Menaxhimi i “cache” memorjes bazuar në algoritmin LRU, fshirja e të dhënave në “cache” të cilat më së paku janë përdorur në kohën e fundit (Burimi: [5]).

Përshkallëzimi (ang. scalability) është një pjesë e rëndësishme të bazat e të dhënave çelës/vlerë si të gjitha llojet e bazat e të dhënave NoSQL. Përshkallzueshmëria është aftësia për të shtuar apo fshirë serverë nga një cluster (grumbull i serverëve) sipas nevojës për të përshtatur ngarkesën në sistem. Aftësia për të përshtatur dy operacionet të transaksioneve OLTP, “Read” dhe “Write” në një sistem të shkallëzuar është një veti e rëndësishme. Bazat e të dhënave çelës/vlerë përdorin dy qasje për shkallëzimin e këtyre dy operacioneve të fundit si: replikimi “**Master-Slave**” dhe replikimi “**Masterless**” [5] [1].

Replikimi Master-Slave ka arkitetkturë të tillë që përbëhet nga një server kryesor si “Master” brenda një koleksioni të serverëve të tjerë, i cili ka për detyrë ruajtjen e gjitha shkrimeve të bëra si dhe replikimit dhe përditësimit e të dhënave për serverët e tjerë në koleksion. Ndërsa, serverët e tjerë, brenda koleksionit, të njohur si “Slave” kanë për detyrë vetëm të përgjigjen për ndonjë kërkesë për lexim (ang. read) të ndonjë të dhëne nga përdoruesi i caktuar [5]. Në rast se serveri kryesor “master” (që përmban të gjitha privilegjet) dështon për arsye të ndryshme, serverët e tjerë “Slaves” inicojnë një protokoll për të përkrahur një nga serverët e tjerë “Slave” brenda koleksionit si “master” server. Siç cekëm edhe në përshkrimin më lart të privilegjet e serverit kryesor “master”, këtu qëndron një problem të replikimi “master-slave” për arsye se vetëm serveri “master” pranon kërkesa për shkrim (ang. write) të të dhënave ndërsa serverët e tjerë nuk mundën, kjo e vështirëson

menaxhimin e një bazë të të dhënave kur kemi të bëjmë me aplikacione të ndryshme që kanë shumë kërkesa për shkrim (ang. write) të të dhënave nga përdorues të shumtë. Me vetëm një server që pranon kërkesa "Write" kemi një ngulfatje të sistemit dhe humbje të performancës dhe efikasitetit në sistem [5] [1].

Për të zgjidhur këtë problem, qasja tjetër e replikimit "Masterless" (që nuk përdor një server si "master") ofron një arkitekturë, e cila mundëson që të gjithë serverët në koleksion të serverëve të pranojnë kërkesa për lexim (ang. read) dhe shkrim (ang. write) të të dhënave nga përdoruesit. Në një sistem të tillë katër (4) kopje të replikimit janë të mjaftueshëm për t'u mbajtur nga serverët brenda koleksionit.

Do të trajtojmë një shembull. Marrim një grup prej tetë serverësh të cilët janë të lidhur së bashku në një strukturë unazore brenda një koleksioni, si më poshtë:

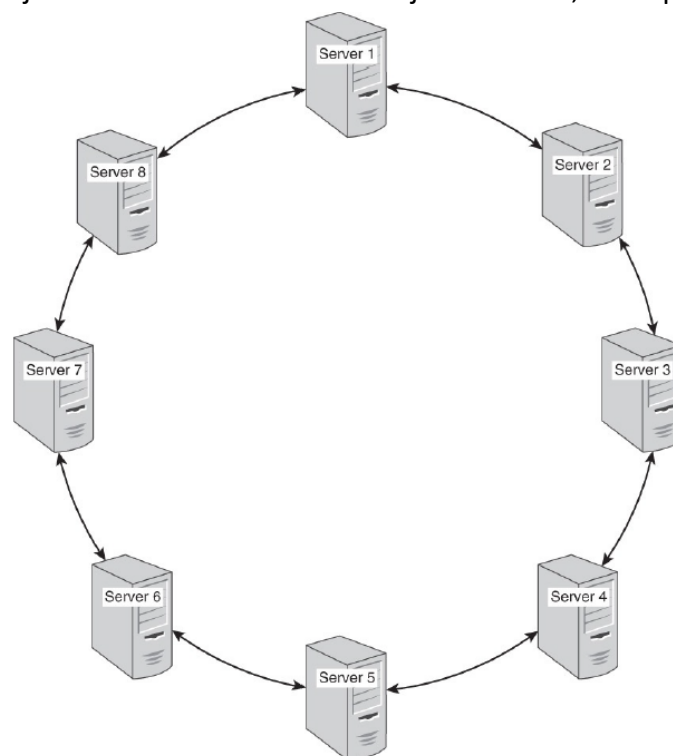


Fig 5 Një koleksion i tetë serverëve të vendosur në një strukturë unazore (Burimi: [5]).

Këtu secili server është i lidhur me fqinjin e tij më të afërt, serveri 1 është i lidhur me serverin 2 dhe serverin 8, poashtu ai 5 me serverin 6 dhe 4, pra në këtë mënyrë përshtë formuar një koleksion serverësh të cilët të gjithë pranojnë kërkesat për lexim dhe shkrim të të dhënave pa një server kryesor si master. Në këtë rast secili serverë replikon (kopjon) të dhënat e tij në dy serverët fqinj lidhur me të, si dhe te një server i tretë (që ndodhet dy nyje më përpara tij), siç kemi paraqitur në shembullin më poshtë:

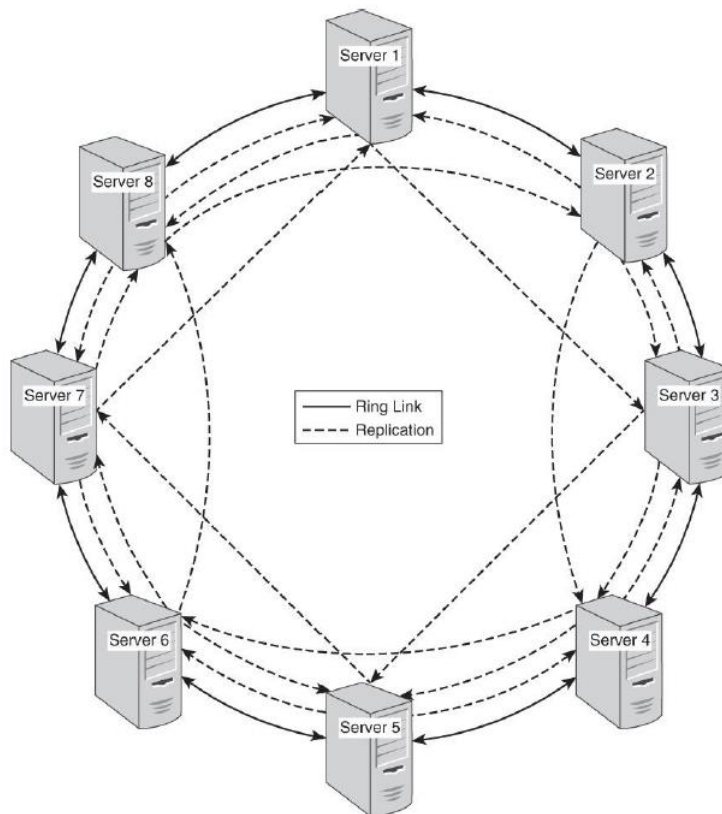


Fig 6. Një koleksion i tetë serverëve të vendosur në një strukturë unazore me një sistem të replikimi të katër kopjeve të serverëve (Burimi: [5]).

Kur më shumë instanca të bazës së të dhënave çelës/vlerë funksionojnë në një koleksion serverësh dhe kemi shtim më të madh të të dhënave dhe përdoruesve, atëherë mund të shtohen server plotësues në koleksion. Instancat çelës/vlerë do ndahen në serverët e ndryshëm për të balancuar ngarkesën. Pra, skema të ndryshme të ndarjes (ang. [partitioning]) janë zgjedhur për të shpërndarë ngarkesën në mënyrë sa më të barabartë të mundshme në serverët e caktuar.

1.2.2. Bazat e të dhënave të llojit të bazuar në dokumente

Për dallim nga çelës/vlera, te bazat e të dhënave të bazuar në dokument, në një dokument mund të ketë shumë attribute me formate të ndryshme. Bazat e të dhënave të bazuara në dokumente, përmbajnë një strukturë e cila lejon më tepër attribute brenda për një instancë (ang. key) të caktuar, ky lloj lejon manipulimin e të dhënave duke mbështetur transaksionet OLTP, si: leximin e të dhënave, insertimin, përditësimin dhe fshirjen e tyre [2].

1.2.2.1 MongoDB

Baza e të dhënave MongoDB ofron përshkallzueshmëri të lartë dhe qasje të shpejtë të të dhënave. MongoDB-ja mundëson kryerjen transaksioneve OLTP, të insertimit (ang. insert), të leximit (ang. read) të përditësimit (ang. update) dhe të fshirjes (ang. delete) së

dokumente apo pjesë brenda dokumenteve, duke përdorur kode të specifikuar.

MongoDB-ja ofron kopjimin e të dhënave (replikimin) në mënyrë asinkrone, ku vetëm një kryesore (ang. primary node) është ngarkuar së pari për të marrë rezultatet nga operacioni i shkrimit (ang. write). Baza e të dhënave MongoDB ofron kopjimin (replikimin) e të dhënave në një të serverëve të ndryshëm. Tek operacionet OLTP, kur përdoruesi bën një shkrim të një të dhëne në një nyje të caktuar, duke përdorur operacionin “write” dhe nëse ky operacion kryhet me sukses atëherë të dhënat e shkruara te një kryesore (ang. primary node) siç cekëm më lart kopjohen te të gjitha nyjet tjera. Në këtë rast kur një përdorues dëshiron të kryej një operacion tjetër si leximi i të dhënave “read” nga këto nyje që janë duke u shkruar (ang. write), atëherë duhet që operacionet e kryera përmes “write” të jenë të propaguara te të gjitha nyjet. Nëse kjo nuk ndodh atëherë leximi i një të dhëne mund të bëhet vetëm nga një kryesore e të dhënave (ang. primary node). Pra, MongoDB-ja nuk lejon leximin nga nyjet tjera, deri sa ata vetë nuk kanë realizuar shkrimin (ang. write) ose përditësimin nga një kryesore. Me këtë nënkuptojmë ruajtje të konsistencës sa i përket anomalive të transaksioneve, duke mos lejuar qasjen e përdoruesve të tjerë, pa pasur një përditësim të gjithë serverëve përkatës që mbajnë kopjet e të dhënave.

MongoDB-ja mëson përshkallëzimin horizontal nëpërmjet një arkitekture automatike të copëzimit (ang. sharding) të të dhënave, duke shpërndarë ato ndërmjet mijëra nyjeve të të dhënave (ang. nodes) me balancimin automatik të ngarkesave dhe të dhënave si dhe mbrojtjen e nyjeve të të dhënave nga dështimi [3]. Copëzimi (ang. sharding) është kuptuar si ndarja (particionimi) i të dhënave ndërmjet shumë makinave në mënyrë të ruajtjes së renditjes së atyre grupeve të të dhënave të shpërndara në serverë të ndryshëm (ang. order-preserving) [3]. Pra, duke kuptuar copëzimin (ang. sharding) si ndarjen (ang. partitionimin) e një grupi të të dhënave nga një nyje e vetme në më shumë nyje apo makina tjera kur kemi disbalancim të ngarkesave të të dhënave, duke ofruar menaxhim efikas dhe qasje të shpejtë të të dhënave të copëzuara në makina të ndryshme.

Copëzimi (ang. sharding) në MongoDB ndodh për koleksion, që ka kuptimin se dokumentet brenda koleksionit të caktuar mund të jenë ndarë në më shumë serverë. Pasi cekëm se MongoDB-ja ofron sharding automatik, atëherë MongoDB-ja automatikisht detekton (zbulon) se cili koleksion i dokumenteve është duke u rritur më shumë se sa mesatarja e caktuar, kështu që atë moment ai koleksion i dokumenteve është subjekt për ndarje në makina apo nyje të ndryshme të të dhënave, ndërsa koleksionet tjera që nuk e kalojnë këtë mesatare të ngarkesës së dokumenteve vazhdojnë akoma të qëndrojnë në nyjet e vetme të të dhënave (ang. single nodes).

Në pjesën e sharding MongoDB-ja lejon ndërtimin e kopjeve të këtyre të dhënave që janë copëzuar (ndarë në nyje të ndryshme të të dhënave). Për secilën ndarje të të dhënave një numër i nyjeve të të dhënave formon një grup të kopjeve (replikimit) të këtyre të dhënave të copëzuara (ang. sharded) [3]. Kështu që një nga nyjet apo serverët është vendosur të jetë si nyje primare, duke trajtuar të gjitha kërkesat për futjen e ndonjë të dhëne të re (ang. write) dhe pastaj i dërgon te serverët e tjerë sekondarë (dytësorë) për të

kopjuar (replikuar) të dhënat, duke u bërë në mënyrë të sinkronizuar. Në rast se serveri kryesor ndodh të dështojë, nyjet e tjera të mbetura zgjedhin një server të ri si server kryesor nëpërmjet një marrëveshje konsesuale të përgjithshme ndërmjet serverëve. Kështu, pavarësisht kësaj ndodhie funksionimi i serverëve të tjerë vazhdon pa ndërprerje, duke ofruar dhe siguruar vetinë “automatic failover” që nëse dështon një server i cili mban të dhënat e kopjuara (replikat), menjëherë vihet në dispozicion serveri tjetër duke ofruar po atë kopje të të dhënave.

Vërtetimi (autentikimi) dhe kontrolli i qasjes nuk është në dispozicion për këta serverë të kopjeve të të dhënave (ang. sharded) [10]. MongoDB -ja përdor qasjen e copëzimit të të dhënave për të optimizuar grupet e mëdha të të dhënave dhe operacionet për manipulimin e tyre, si transaksionet OLTP [11]. Duke pasur parasysh këtë fakt që një koleksion mund të përmbajë shumë dokumente të ndryshme të të dhënave, nga ku MongoDB-ja ofron një sistem automatik i cili shqyrton shtimin e madhësisë së ndonjë dokumenti dhe bënë copëzimin (ang. sharding) në serverë të ndryshëm të të dhënave.

Bazat e të dhënave bazuar në dokumente, për dallim nga ato çelës/vlerë që paraqesin një çelës (ang. key) për një rresht të caktuar, ruajta dokument paraqet një çelës (ang. key) identifikues për një dokument të vetëm i cili ruan shumë attribute të ndryshme të një entiteti. Mënyra për të insertuar, përditësuar apo fshirë të dhënat mund të bëhet përmes këtij çelësi identifikues të dokumentit.

1.2.3. Bazat e të dhënave të llojit të bazuar në kolona

Bazat e të dhënave të bazuara në kolona mund të ruajnë të dhënat në kolona të ndryshme dhe rreshtat e ndryshëm mund të kenë kolona të ndryshme, çka e dallon nga modeli relacional ku secila tabelë ka kolona të fiksuara për rreshtat. Kjo ndodh për arsye se bazat e të dhënave të bazuara në kolona nuk ofrojnë mbështetje për operacionin “JOIN” për lidhjen e tabelave të ndryshme si në bazat e të dhënave relacionale. Këto baza të të dhënave të bazuara në kolona janë të ndara në grupe (ang. batch) të cilët përmbajnë një çelës (ang. key) për rresht. Atributet në rreshtat mund të kenë atributet tjera të mbivendosura. Bazat e të dhënave të llojit të bazuar në kolona janë të dalluar për vetitë e përshkallëzueshmërisë, besueshmërisë dhe konsistencës eventuale [12].

Sinkronizimi i të dhënave të kopjuara në server paralel arrihet duke përdorur një qasje (ang. “a partial and temporary immutability”) “Copy-on-Write” dhe duke lejuar leximin dhe shkrimin për të procesuar në mënyrë paralele. Lloji i bazave të të dhënave të bazuara në kolona (ang. column-based) përmban një arkitekturë të procesimit paralel që mundëson ndërtimin (krijimin) e aplikacioneve me performancë të lartë, bazuar në përshkallëzueshmërinë e lartë.

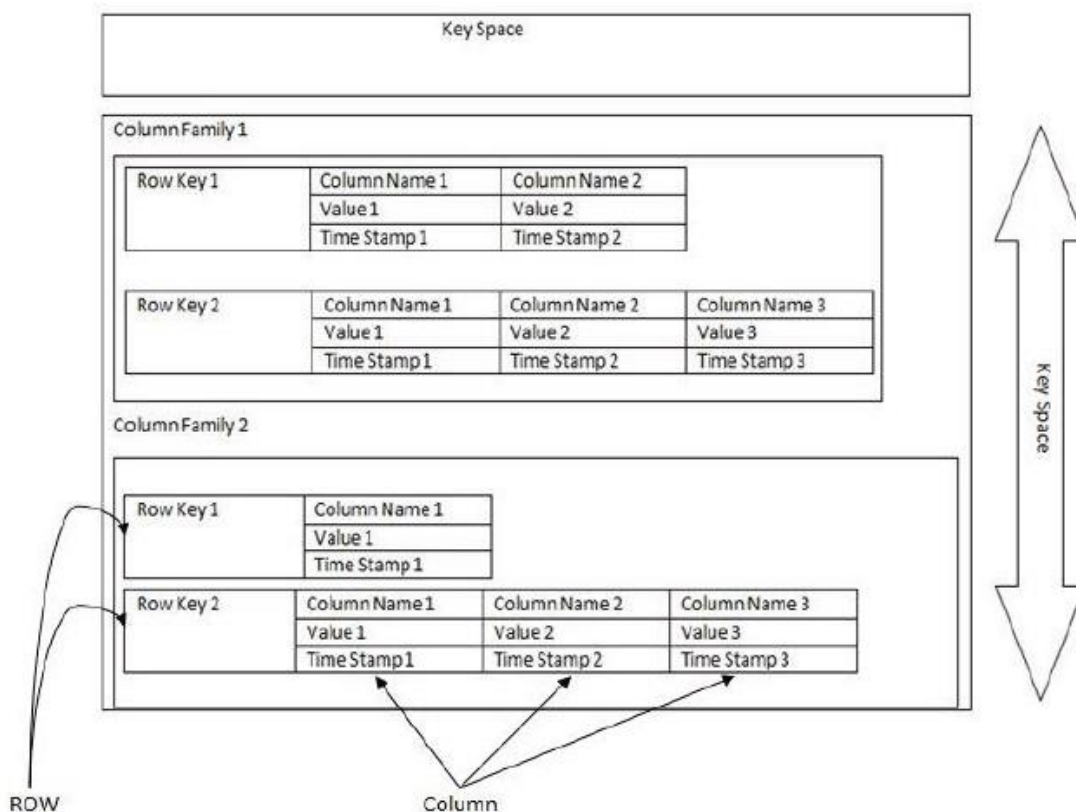


Fig 7. Strukturimi i bazave të të dhënave të llojit të bazuar në kolona (Burimi: [1]).

Bazat e të dhënave të llojit të bazuar në kolona shikuar edhe nga ilustrimi më sipër, përbëhen nga një çelës i hapsirës (ang. key – space) i cili përmban tërë strukturën e familjes së kolonave të bazës së të dhënave [1]. Familjet e kolonat (grupet) përmbajnë një numër të kolonave të shoqëruara me rreshtat e tyre. Secili rresht përmban një numër të caktuar të kolonave [1]. Strukturimi i tillë i ndarjes së kolonave të veçanta lejon marrjen e të dhënave në mënyrë më të shpejtë, duke zgjedhur vetëm të dhënat e kërkuara [1] [13]. Një qasje e tillë është gati e ngjajshme me operacionin për projeksion (ang. projection) te bazat e të dhënave relacionale SQL.

Inspirimi kryesor për këto baza të të dhënave të orientuara në kolona (ang. column-based) është Google Bigtable [3]. Sipas “Chang et al” gjëja më e rëndësishme në zhvillimin e Google’s Bigtable është thjeshtësia dhe qartësia e dizajnit të tij. Bigtable është përshkruar si një sistem i ruajtjes së shpërndarë për menaxhimin e të dhënave të strukturuar, e cila është dizajnuar (projektuar) për të përshkallëzuar një sasi shumë të madhe të të dhënave nëpër mijëra serverë [3]. Qëllimi i Google’s Bigtable ka qenë ndërtimi i një sistemi që do të mundësonte përshkallëzueshmëri dhe performancë të lartë në ndërtimin e aplikacioneve, zbatueshmërisë së gjerë dhe disponueshmërisë së lartë të shërbimeve të ofruara. (Për të ofruar shërbime sa më efikase dhe në mënyrë të shpejtë) Bigtable menaxhon mirë plotësimin e klasterit me nyje të reja për të përballuar dhe menaxhuar një fluks të madh të të dhënave dhe përdoruesve, duke mundur ofrimin e shërbimeve sa më efikase dhe në mënyrë më të shpejtë.

Nëse bëjmë një krahasim ndërmjet modelit të ruajtjes çelës/vlerë (ang. store) dhe Google Bigtable, konkludojmë se Bigtable është më e pasur, kjo për arsye se algoritmet që përdor modeli çelës/vlerë janë shumë të kufizuara, ku një aplikacion ruan dhe merr të dhëna në grupe të vogla të çelësve (ang. keys) [14]. Ndërsa ,Bigtable u mundëson përdoruesve shkrimin e një pyetësi (ang. query) që skenon (shqyrton) një numër të madh të rreshtave dhe kolonave.

1.2.3.1. Cassandra

Një platformë tjetër e ruajtjes së llojit të bazuar në kolona quhet Cassandra e cila ofron një model të thjeshtë të të dhënave që mbështet kontrollin dinamik mbi të dhënat. Për dallim nga llojet tjera të bazave të të dhënave të bazuar në kolona, Cassandra përmban një model tjetër të të dhënave. Strukturimi i kolonave bëhet përmes një niveli të ri të quajtur super – kolonat, të cilat përmbajnë shumë kolona për një rresht të caktuar [1].

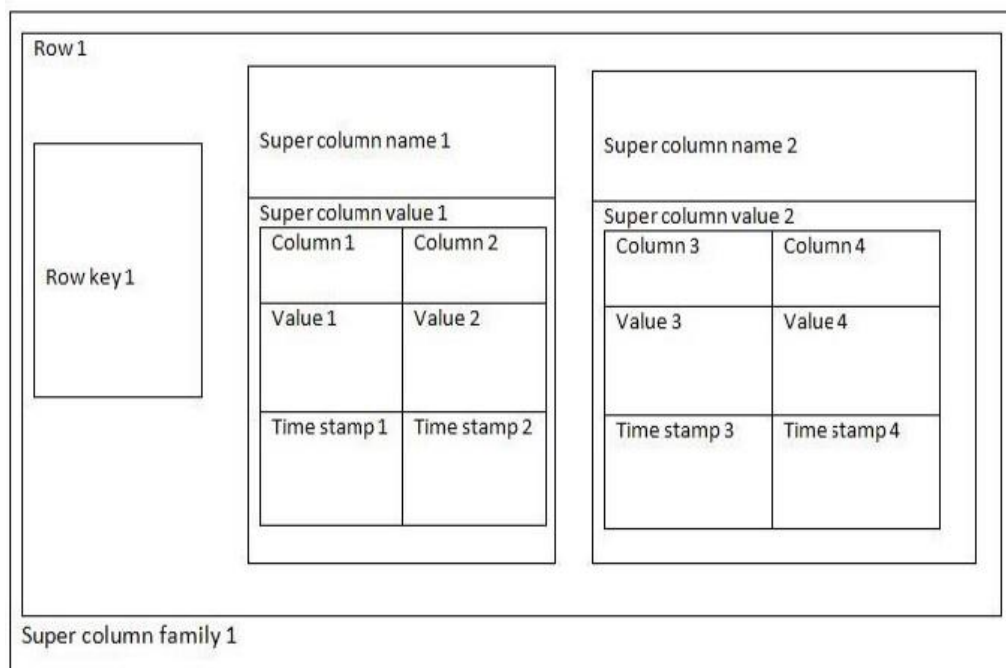


Fig 8. Strukturimi sipas bazës së të dhënave – Cassandra (Burimi: [1]).

Natyra e shkrimit të të dhënave përmes Cassandra-s, të dhënat e përditësuara janë ruajtur në regjistrin të skedarit (ang. log file commit). Nëse transaksioni ekzekutohet me sukses, hapi i ardhshëm do të jetë ruajta në memorjen e përkohëshme dhe pastaj në skedarët e diskut magnetik të ruajtjes në mënyrë të përhershme [10].

Detektimin (gjetja) e dështimeve të Cassandra në bazë të implementimeve dhe eksperimenteve të kryera rezulton se koha e gjetjes së një dështimi gjatë ruajtjes apo procesimit të të dhënave rritet në bazë të madhësisë së grumbullit (klasterit) të nyjeve (makinaeve). Në një eksperiment të kryer me një grumbull të serverëve të të dhënave që përbën 100 sosh, disa zbuluesëve të dështimeve u është dashur deri në dy minuta për të zbuluar një server të dështuar [3].

Platforma Cassandra ofron edhe disa karakteristika të bazave të të dhënave SQL tradicionale e cila mbështet vetitë e atomicitetit, izolimit dhe qëndrueshmërisë së transaksioneve. Vetitë e atomicitetit dhe izolimit të transaksionit janë mbështetur në nivel të rreshtit (ang. row – level) [15].

1.2.4. Bazat e të dhënave të llojit të bazuar në grafe

Bazat e të dhënave bazuar në graf (ang. graph-based) lidhin dy entitete në bazë të disa attributeve të përbashkëta mes tyre. Bazat e të dhënave të bazuara në grafe (ang. graph databases) përdorin nyjet (ang. edge) që paraqesin objektet dhe lidhjet (ang. relationship) ndërmjet tyre bazuar në disa attribute të përbashkëta. Një nyje (ang. edge) është një objekt që ka një identifikues dhe një grup të attributeve, ndërsa një lidhje (ang. relationship) është lidhja ndërmjet dy nyjeve që përmbajnë attribute rreth këtij relacioni [5]. Këto baza të të dhënave, bazuar në graf, përdoren kryesisht në rrjetet sociale, në një sistem të menaxhimit të transportit që lehtëson në gjetjen e të dhënave me shpejt, si dhe aplikacione të ngjajshme që përdorin një mënyrë të lidhjeve të rrjetizuar.

1.3. Qëllimi i punimit

Ky punim ka për qëllim të shqyrtojë një sistem, i cili menaxhon në mënyrë më efikase grupet e mëdha të të dhënave, të procesuara në kohë reale. Fokusi kryesor i temës është trajtimi i bazave të të dhënave NoSQL të cilat veprojnë në një rrethinë të shpërndarë dhe paralele. Punimi paraqet analizimin e llojeve të ndryshme të bazave NoSQL për të krahasuar performancën e tyre. Secila nga llojet e bazave NoSQL paraqet një strukturë të caktuar për të menaxhuar të dhënat. Veçanërisht, ky punim paraqet rëndësi në menaxhimin e grupeve të mëdha të të dhënave për të testuar operacionet CRUD në lidhje me NoSQL, bazën e caktuar. Si shkas, punimi synon të japë kontribut në këtë aspekt, i cili do të paraqiste përparësitë dhe mangësitë e sistemit të shpërndarë kur kemi të bëjmë me NoSQL.

1.4. Hipotezat

Bazuar në hulumtimin tonë, në bazë të analizës së bazave NoSQL në procesimin e grupeve të mëdha të të dhënave, do të shqyrtojmë hipotezat si në vijim:

1. Modelimi i të dhënave NoSQL përmes platformës Hadoop mundëson organizim efikas dhe të suksesshëm të të dhënave si dhe përdorimi i MapReduce Framework-ut i cili mundëson procesim paralel efikas.

2. Transaksionet OLTP të cilat veprojnë në bazat e të dhënave NoSQL, bazuar në platformën Hadoop kryhen në mënyrë të suksesshme, të sigurtë dhe me performancë të lartë.
3. Vetitë e teoremës CAP (ang. Consistency, Availability, Partition-tolerance) të zbatuara përmes bazave të të dhënave NoSQL implementohen në mënyrë të suksesshme përmes platformës Hadoop.

1.5 Pyetjet kërkimore

- Si mund të bëhet ndarja e të dhënave (konfigurimi) dhe menaxhimi i resurseve në nyje të ndryshme brenda Hadoop klasterit?
- Çfarë dallimi në performancën e ekzekutimit të operacioneve CRUD paraqet klasteri i përbërë nga tri nyje në raport me një nyje punuese?
- Si arrihet procesimi në mënyrë paralele i të dhënave në bazën e të dhënave MongoDB bazuar në vetitë e teoremës CAP?
- Çfarë ndikimi paraqet rritja e numrit të dokumenteve (në MongoDB) në kohën e ekzekutimit të operacioneve CRUD?

1.6 Metodologjia e hulumtimit

Për të realizuar hulumtimin tonë kemi përdorur disa metoda të cilat janë: metodat përshkruese, empirike, analizuese si dhe krahasuese. Në fillim kemi përshkruar në mënyrë teorike ndërtimin e një rrethine të shpërndarë me përparësitë dhe mangësitë e saj. Në bazë të hipotezave të ngritura do të analizojmë ndërtimin e një rrethine të shpërndarë e cila do të implementohet përmes Hadoop platformës, si dhe do të paraqesim rezultatet empirike nga te gjeturat e ndryshme në këtë drejtim. Bazuar në këtë do të ndërtohet një Hadoop klaster i përbërë prej katër nyjeve, që paraqet ndarjen e resurseve për secilën nyje, i cili mundëson procesimin paralel përmes strukturës Map-Reduce.

Në vazhdim do të konfigurojmë një klaster prej tri nyjesh punuese duke përdorur bazën e të dhënave MongoDB. Brenda bazës së të dhënave MongoDB do të vendosen grupe të ndryshme të të dhënave. Për të mundësuar një procesim të shpërndarë dhe paralel brenda MongoDB-së e cila do të përmbushte vetitë e teoremës CAP, do të përdorim një nga strategjitë e sharding-ut të njohur si Hash, e cila mundëson ndarjen e të dhënave në mënyrë

sa më të barabartë në të gjitha nyjet. Për të manipuluar me grupet e ndryshme të të dhënave janë përdorur operacionet CRUD.

Në fund, do të krahasojmë kohën e ekzekutimit të operacioneve CRUD për raste të ndryshme, kur kemi tërë klasterin dhe një nyje punuese. Gjithashtu, krahasimet do të bëhen në testimin e performancës së ekzekutimit të operacioneve CRUD për grupet e ndryshme të të dhënave.

2. GJENDJA MOMENTALE E PERFORMANCAVE TË BAZAVE NOSQL NË MJEDIS TE DISTRIBUARA

Bazat e të dhënave relacionale lejojnë ruajtjen dhe manipulimin e të dhënave duke përdorur një gjuhë standarde SQL (ang. structured query language). Bazat e të dhënave relacionale janë zgjedhje e duhur për shume aplikacione komerciale. Megjithatë me rrjedhjen në masë të madhe të të dhënave përmes aplikacioneve të ndryshme, bazat e të dhënave relacionale shfaqin disa kufizime në menaxhimin e tyre, p.sh. kufizimet në përshkallzueshmëri dhe ruajtje të të dhënave si dhe në humbjen e efikasitetit të procesimit të përzgjedhjes (pytësorëve) [16].

Në mënyrë për të hequr këto kufizime është projektuar një model i ri i bazave të të dhënave NoSQL [16]. Bazat e të dhënave NoSQL paraqiten të jenë një komplement për bazat e të dhënave relacionale. Këto baza të të dhënave janë projektuar që në mënyrë automatike të menaxhojnë dhe shpërndajnë të dhënat, si dhe në rast së ndodh ndonjë dështim të rregullojnë tërë sistemin në mënyrë automatike [16].

Fillimisht bazat e të dhënave NoSQL ishin karakterizuar nga mungesa e konsistencës së ruajtjes së të dhënave, si një nga vetitë themelore të bazave të të dhënave. Por aktualisht janë në dispozicion një numër i konsiderueshëm i bazave të të dhënave NoSQL të cilat ofrojnë karakteristika të reja dhe përparësi për t'i mbajtur të dhënat në një gjendje konsistente të plotë ose pjesërisht konsistente, varësisht prej kërkesave të sistemit [16]. Disa nga bazat e të dhënave NoSQL, si: MongoDB, DynamoDB dhe SimpleDB mbështesin konsistencën e fortë dhe konsistencën e mundshme, ndërsa CouchDB ofron karakteristika të konsistencës së mundshme. Bazat e të dhënave relacionale përmes vetive ACID mbështesin konsistencën e qëndrueshme të të dhënave [6]. Ndërsa, bazat e të dhënave të cilat nuk implementojnë në menyrë të plotë vetitë ACID do të kenë mbështetur konsistencë të mundshme të të dhënave [6]. Në parim, nëse nuk mbështesim konsistencën e qëndrueshme (ang. strong consistency), do të arrijmë një disponueshmëri më të mirë e cila do të përmirësojë në nivel të lartë përshkallzueshmërinë e bazave të të dhënave NoSQL [6].

Për rritjen e performancës së shpejtësisë së ekzekutimit të përzgjedhësve (pytësorëve), bazat e të dhënave NoSQL përdorin memorjen e përkohshme për të mbajtur të dhënat [16]. Procesimi i përzgjedhësve (pytësorëve) përmes memorjes së përkohshme redukton intervalin kohor të ekzekutimit të tyre. Lloje të ndryshme të bazave të të dhënave NoSQL ofrojnë mekanizma të ndryshëm për të ruajtur dhe për të marrë të dhënat, të cilat mund të ndikojnë direkt në performancën e sistemit të ngritur [16]. Gjithashtu, secila NoSQL përmban mënyra të ndryshme për të optimizuar përzgjedhësit (pytësorët), që rezultojnë në intervale të ndryshme kohore për të procesuar të tilla transaksione OLTP. Për shembull, MongoDB dhe Cassandra përdorin metodën e indeksimit të të dhënave bazuar në grupe të veçanta të tyre, ndërsa bazat e të dhënave Redis (e bazuar në llojin çelës/vlerë) mundësojnë ruajtjen dhe marrjen e të dhënave përmes cache memorjes, që ndikon në rritjen e performancës së ekzekutimit të transaksioneve OLTP [17]. Prandaj, ruajtja e të dhënave dhe

qasja e tyre prej memorjes së përkohshme siguron një performancë më të lartë në disponueshmërinë të dhënash, kur madhësia e grupit të të dhënave nuk tejkalon madhësinë e memorjes së përkohshme [18].

Duke pasur parasysh këto që cekëm më lart, në punimin [16], autorët V. Abramova, P. Furtado dhe J. Bernardino kanë analizuar dhe krahasur pesë baza të të dhënave NoSQL me tri lloje të ndryshme të tyre. Gjatë eksperimentit në fjalë janë përdorur këto baza të të dhënave NoSQL si në vijim [16]:

- **Cassandra** dhe **Hbase** - nga bazat e të dhënave të llojit të bazuar në Kolona.
- **MongoDB** dhe **OrientDB** - nga bazat e të dhënave të llojit të bazuar në Dokument
- Redis** – nga baza e të dhënave të llojit të bazuar në çelës/vlerë (ang. key/value)

Ky eksperiment paraqet testimin e këtyre bazave të të dhënave NoSQL të këtyre tri llojeve për të krahuar performancën e transaksioneve OLTP. Puna eksperimentale trajton kryesisht dy nga operacionet OLTP, si: leximi i të dhënave (ang. Read) dhe përditësimi i të dhënave (ang. update).

Testimet janë kryer në një makinë virtuale si Ubuntu Server 32bit me kapacitet prej 2GB RAM e vendosur në një kompjuter me sistemin operativ Windows 7 me kapacitet të memorjes së përkohshme prej 4GB RAM [16]. Për të testuar bazën e të dhënave janë gjeneruar në mënyrë randome (ang. randomly) 600,000 shënime (rekorde) të të dhënave, në të cilat janë kryer 1000 operacione mbi grupet e të dhënave [16].

Nga rezultatet e punimit [16], konkludojmë se disa nga bazat e të dhënave NoSQL, si: MongoDB, Redis dhe OrientDB paraqesin performancë më të mirë në procesimin e operacioneve për lexim (ang. Read), përderisa Cassandra dhe Hbase rezultojnë të kenë performancë më të mirë në procesimin e operacioneve për përditësimin e të dhënave, në performancën e procesimit të operacioneve në nivelin më të lartë që përdor memorjen e përkohshme për të procesuar të dhënat [16].

Në vazhdim, në punimin [19], autorët S.Ahirrao dhe R.Ingle, kanë testuar metodat për përshkallëzueshmërinë e transaksioneve në sistemet e shpërndara, në bazë të modeleve për ndarjen e të dhënave. Qëllimi kryesor i punimit ka të bëjë me mbështetjen e përshkallëzuar të transaksioneve për ueb-aplikacionet [19]. Impelementimi dhe testimi i modeleve të ndarjes (ang. partitioning) së të dhënave është kryer në bazat e të dhënave NoSQL si Hadoop HBase dhe Amazon SimpleDB.

Fushëveprimi eksperimental ishte kryer në një koleksion të pesë serverëve, ku të gjithë severët posedojnë karakteristikat kompjuterike si procesor me M3 General Purpose Extra Large me katër bërthama (4 x 3.25), 15 GB mermorje të përkohshme, diskun për ruajtje të perherëshme me 2 x 40 GB kapacitet, si dhe me sistemin operativ Windows 8 [19].

Modelet për ndarjen e të dhënave të cilat janë testuar janë:

- **Ndarja e të dhënave bazuar në nivel të Skemave** - (ang. Schema level partitioning) – e cila bazohet në ndarjen e të dhënave në bazë të çelësit të ndarjës (ang. partitioning key). Çelësi i ndarjës përbën grupin e një ose më shumë kolonave të cilat ndajnë rreshtat në grupe të ndryshme (në bazë të vlerave të kolonave) [20] [21]. Modeli i ndarjës i bazuar në skemë paraqet mënyrën statike të ndarjes së të dhënave [19].
- **Ndarja e të dhënave bazuar në Graf** - (ang. Graph Partitioning) – paraqet ndarjen e të dhënave në mënyrë statike në bazë të volumit të të dhënave [19]. Ndarja e bazuar në graf siguron balancimin e ndarjes së të dhënave në mënyrë të barabartë dhe në mënyrë paralele. Modeli në fjalë mundëson procesimin e shumë kërkesave për të dhënat nga përdorues të ndryshëm njëkohësisht [22] [23]. Për t’iu shmangur transaksioneve të shpërndara, rekordet e të dhënave të cilat janë qasur nga një transaksion janë vendosur në njësi të përbashkët të ndarjës [19].
- **Ndarja e të dhënave bazuar në modelet e qasjes së të dhënave** – algoritmi për ndarjen e të dhënave në mënyrë të përshkallëzuar ka kosto më të optimizuar se dy modelet e cekura më lart. Në bazë të këtij modeli ndarja e të dhënave bëhet përmes analizës së transaksioneve dhe modeleve të qasjes së të dhënave. Modelet e të dhënave diktojnë se si do të krijohen ndarjet, në bazë të lëvizjes së të dhënave në mënyrë periodike [19]. Modeli i tillë i ndarjës së të dhënave do të përmirësonte përshkallëzueshmërinë dhe do të zvogëlonte koston e transaksioneve të shpërndara [19]. Sipas analizës së bërë nga punimi [19], autorët në fjalë kanë konkluduar se lloji i ndarjës bazuar në modelin e qasjes së të dhënave jep një “throughput” më të lartë, kohë më të ulët të përgjigjes për përzgjedhësit (ang. querying) si dhe reduktimin e transaksioneve të shpërndara, në raport me dy modelet tjera të ndarjës [19].

Siç theksuam edhe më lart në analizën rreth teoremës CAP konsistenca dhe disponueshmëria janë arritur përmes përshkallëzimit horizontal, ruajtjes së kopjeve për të dhënat, si dhe ofrimin e modelit të të dhënave sa më të thjeshtë [24]. Në punimin [24], autori J.Klein sëbashku me një grup të tjerë autorësh, kanë testuar performancën dhe përshkallëzueshmërinë e transaksioneve në lidhje me sistemin elektronik të regjistrave shëndetësor. Gjatë testimit janë krahasuar tri nga bazat e të dhënave NoSQL, si : MongoDB në verzionin 2.2, Cassandra në verzionin 2.0 si dhe Riak në verzionin 1.4 [24], në bazë të transaksioneve të shpërndara. Analiza ishte kryer në një koleksion prej nëntë serverëve dhe tri modelve për transaksionet si read- only, write-only dhe read-write. Nga testimet e bëra, është konkluduar se Cassandra është baza e të dhënave me performancë më të mirë, (me throughput më të lartë) e ndjekur nga Riak dhe MongoDB [24].

Sfidat të ndryshme në procesimin e volumit të madh të të dhënave janë paraqitur përmes punimit [25], ku autori R.R.Shetty me një grup autorësh kanë trajtuar rastin e grupeve të mëdha të të dhënave për qështjet mjekësore që kanë të bëjnë me sëmundjet kardiovaskulare dhe vdekshmërinë e foshnjëve. Autorët kanë për qëllim që të krijojnë një sistem të tillë që do të trajtonte në mënyrë sa më efikase grupe të mëdha të të dhënave të

pa strukturuara, sigurimin e të dhënave sa më të sakta për hulumtuesit e fushës mjekësore. Për trajtimin e këtyre grupeve të mëdha të të dhënave kanë përdorur bazat e të dhënave NoSQL, llojin e bazuar në dokumente me bazën e të dhënave si MongoDB. Nga eksperimentet e bëra është konkluduar se përmes procesimit paralel të të dhënave, do të kenë ngarkimin dhe marrjen e të dhënave në mënyrë më të shpejtë dhe efikase. Ekperimenti tregon se duke trajtuar të dhënat me bazat e të dhënave NoSQL kemi rritje në performancë bazuar në një sistem të përshkallëzuar, i cili kishte ofruar zgjidhje të duhura dhe efikase për hulumtuesit e fushës mjekësore.

Modeli i përzgjedhësve të agreguar (ang. aggregate query model) mund të jetë një pershtatje e mirë për bazat e të dhënave të fokusuar kryesisht në leximin e të dhënave (ang. read-mostly) siç përshkruhet nga punimi [26] në të cilin autori D.Pasqualin sëbashku me një grup autorësh kanë prezentuar një skemë të objekteve të përbëra të implementuar në bazat e të dhënave të llojit të bazuar në dokumente. Hulumtimi ishte kryer mbi bazën e të dhënave prej 300.000 rekordeve të të dhënave, në lidhje me ministrinë e komunikimit në shtetin e Brazilit. Qëllimi themelor i këtij hulumtimi është që të ofrojë një skemë të tillë që do të grumbullonte të dhënat nga dokumentet e ndryshme, bazuar kjo në elementet e përbashkëta ndërmjet tyre. Nga punimi në fjalë ishte konkluduar se përdorimi i bazës së të dhënave të llojit të bazuar në dokumente është më i përshtatshëm në dizajnimin e sistemit të fokusuar kryesisht në leximin e të dhënave (ang. read-mostly). Autorët vijnë në përfundim se bazat e të dhënave të llojit të bazuar në dokument, mundësojnë ruajtjen e hapsirës së të dhënave kur kemi të bëjmë me grumbullimin e të dhënave të përbashkëta.

Për të balancuar ngarkesat e grupeve të mëdha të të dhënave është përdorur ndarja e të dhënave në mënyrë horizontale duke i shpërndarë ato në vende të ndryshme. Megjithatë ndarja e të dhënave në mënyrë të pa kontrolluar mund të rezultojë në transaksione të shpërndara të cilat mund të kenë kosto shumë të lartë. Në punimin [27], autorët Zh.Wei, G.Pierre, Ch.Chi, kanë analizuar dhe testuar sistemin e ndarjes së të dhënave në llojin e bazuar në graf. Qëllimi i tyre ishte që të ofrojnë një zgjidhje adekuate për të minimizuar transaksionet e shpërndara duke pasur si bazë përshkallëzueshmërinë.

Puna eksperimentale është kryer duke përdorur një nga bazat e të dhënave të llojit të bazuar në dokumente si CouchDB. Megjithatë, Neo4j, baza e të dhënave e llojit të bazuar në graf, është përdorur për të integruar të dhënat nga CouchDB. Nga puna eksperimentale e bërë është arritur në përfundim se modeli i bazuar në Graf, përmes karakteristikave të modelit të rrjetit, mundëson minimizimin e transaksioneve të shpërndara.

3. TRANSAKSIONET OLTP

Ndër pjesët më të rëndësishme në aspektin e menaxhimit me bazat e të dhënave është procesimi i transaksioneve (ang. Online Processing Transaction). OLTP-ja kryen një rol të rëndësishëm në menaxhimin e aplikacioneve të orientuara në transaksione.

Duke parë volumin e madh e të dhënave në ditët e sotme, i cili është gjithnjë e një rritje, ka bërë që edhe kërkesat transaksionale të rriten. Faqet e rrjeteve sociale si “Facebook”, apo “Linkedin”, të cilët kanë një numër të madh përdoruesish, kanë bërë që të ketë kërkesa më të mëdha dhe më komplekse për transaksionet OLTP [28].

Transaksionet OLTP mundësojnë procesimin e kërkesave transaksionale që veprojnë në kohën aktuale (ang. real – time). OLTP-të janë të dobishme për aplikacionet e tregtisë elektronike (ang. e – commerce), platformave sociale si dhe aplikacionet tjera të cilat veprojnë në kohë reale në rrjet.

Rritja e volumit të madh e të dhënave në të gjitha aspektet, ka bërë që të kemi transaksione më komplekse. Zhvillimi i modelit të Ueb 2.0 ka lejuar paisjet, si: telefonat e mençur (ang. smartphone)dhe tabletet (ang. tablet) të jenë në gjendje të kryejnë transaksione kudo dhe në çdo kohë. Aplikacionet e biznesit të llojeve të ndryshme janë bërë më dinamike në procesimin e transaksioneve të niveleve të ndryshme, duke bashkëvepruar dhe përmirësuar eksperiencën e përdoruesve. Besueshmëria e sistemit dhe integriteti i të dhënave janë pjesa më kryesore në ndërtimin e aplikacioneve [29].

Në kuptimin e përgjithshëm, një transaksion është një detyrë procedurale i cili kryen disa hapa për të arritur qëllimin e caktuar. Në jetën e përditshme shpesh ndeshemi me lloje të ndryshme të transaksioneve, duke filluar nga procesi i thjeshtë i blerjes së një produkti në një lokal biznesi më të afërt, deri te transaksionet më komplekse që mund të ndodhin qoftë edhe përmes internetit. Të gjitha këto transaksione të çfarëdo lloji veprojnë në disa hapa për të realizuar qëllimin e caktuar të tyre [29].

Transaksionet OLTP mbështesin aplikacionet interaktive, në të cilat sistemi është në gjendje për t’iu përgjigjur në mënyrë të menjëhershme kërkesave të përdoruesve [29].Detyra e procesimit të transaksioneve përmes internetit (ang. OLTP) është të bëjë procesimin e operacioneve për manipulimin e të dhënave, të cilat pastaj do të analizohen përmes mjedisit të procesimit analitik në internet (ang. OLAP) [30].

Siç theksuam edhe më lart, bazat e të dhënave NoSQL të shpërndara janë projektuar për të mbështetur një sistem i cili ofron përshkallëzueshmëri horizontale dhe disponueshmëri më të lartë se sa bazat e të dhënave tradicionale, por me më pak konsistencë të garantuar dhe më pak mënyra për të krijuar përzgjedhës (ang. query). Për të përmbushur kërkesat e aplikacioneve të sotme, gjithnjë në rritje të përdoruesve dhe të të dhënave, modeli i bazave të të dhënave tradicionale SQL nuk mund të jetë i përshtatshëm në të raste të shumta. Duke pasur parasysh këtë zhvillim të kohëve të fundit, trajtimi i të dhënave në një sistem të shpërndarë kërkon që transaksionet të jenë të përshkallëzuar [4].

Bazat e të dhënave NoSQL janë më të përshtatshme dhe më efikase në procesimin e transaksioneve kur bëhet fjalë për grupe të të dhënave me volum të madh [4].

Procesimi i transaksioneve në kohë reale (ang. online) të cilat veprojnë në grupe të mëdha të të dhënave mund të jetë më kompleks. Për të trajtuar këto kërkesa, është krijuar një nga platformat e njohur si CumuloNimbos e financuar nga BE (Bashkimi Evropian) dhe realizuar nga Universiteti Politeknik i Madridit. Qëllimi i këtij projekti është që të ofrojë mbështetje për procesimin e transaksionve shumë të përshkallëzuar [31]. CumuloNimbo është një platformë që mundëson konsistencën e të dhënave, lehtë e programueshme si dhe mbështet përshkallëzueshmërinë në sistemet transaksionale [31]. Një shembull i përshkallëzueshmërisë transaksionale është se, CumuloNimbo mundëson procesimin e transaksioneve për përditësimin (ang. update) e të dhënave në shkallën e një milion transaksioneve për sekond [31]. Performance e transaksioneve OLTP është përmirësuar edhe me zhvillimin e hovshëm të paisjeve harduerike, në rritjen e njësisë qendrore si CPU për procesimin e transaksioneve.

3.1 Transaksionet OLTP për bazat e të dhënave NoSQL

Në një sistem të shpërndarë dhe paralel të dhënat janë ndarë në mënyrë horizontale në më shumë serverë. Ky lloj i ndarjës së të dhënave ndërmjet shumë serverëve të ndryshëm lejon që të kemi transaksione që veprojnë në shumë serverë të ndarjës së të dhënave [4] [19] [6]. Këto njihen si transaksione të shpërndara të cilat kërkojnë disa protokole të komunikimit dhe koordinimit ndërmjet serverëve [4] [19] [3]. Për sistemet e shpërndara ku transaksionet veprojnë mbi një grup të serverëve, është e vështirë që të ruhet konsistenca dhe replikimi i të dhënave. Vështirësitë mund të vijnë nga dështimet që mund të ndodhin në ndonjë serverë në koleksionin e shpërndarë [32]. Për zgjidhjen e këtyre problemeve, një nga protokolet e komunikimit dhe koordinimit ndërmjet serverëve është përdorur, që njihet si Paxos protokoli. Paxos (që merr emrin nga ishulli grek Paxos) fillimisht ishte publikuar në vitin 1989, detyra e të cilit është të koordinojë operacionet ndërmjet serverëve të ndryshëm për të arritur një konsensus ndërmjet tyre [4]. Konsensusi ndërmjet serverëve do të thotë se të gjithë pjesëmarrësit në sistemin e shpërndarë të ngritur janë në gjendje të bashkëpunojnë me njëri tjetrin për të arritur një rezultat konkret [4].

Procesimi i përzgjedhësve (ang. queries) në të shumtën e rasteve bazohet nga: modeli i sistemit të shpërndarjës, ruajtja e konsistencës si dhe nga modeli i të dhënave [33]. Përzgjedhësit për marrjen e të dhënave bazuar në një identifikues unik (ang. ID) mbështeten r nga të gjitha sistemet e bazave të të dhënave NoSQL [33]. Modelet të ndryshme të bazës së të dhënave NoSQL mbështesin teknika të ndryshme për ndarjen e të dhënave. Dy nga teknikat e përdorura për ndarjen e të dhënave janë ndarja përmes hash modelit (ang. hash partition) dhe vargut (ang. range partition).

Hash modeli për ndarjen – përdor një hash çelës (ang. hash key) për të shpërndarë

rreshtat në mënyrë të barabartë nëpër tabela të ndryshme të të dhënave [34].

Range (varg) modeli për ndarjen – paraqet ndarjen e të dhënave bazuar në vargun e vlerave për një fushë të caktuar. Të dhënat janë shpërndarë ndërmjet grupeve të ndarjës bazuar në vlerat e fushës së përdorur si hash ndarje [35]. Te modeli i ndarës bazuar në varg, të dhënat janë ndarë bazuar në çelësin e ndarjës. Të dhënat mund të jenë ndarë në një kolonë që përmban datat e caktuara, psh. ndarja mund të bëhet në bazë të periudhës Mars 2018 deri në Dhjetor 2018. Kjo paraqet një ndarje bazuar në një periudhë të caktuar, të gjitha të dhënat që plotësojnë këtë kriter vendosen në këtë ndarje.

Ndarja bazuar në varg kërkon që ndarja e të dhënave të bëhet në mënyrë të barabartë në grupe të ndryshme të ndarjës së tyre. Duke lehtësuar ndarjen e të dhënave dhe qasjen më të shpejtë në grupe më të vogla të të dhënave, modeli i ndarjës bazuar në varg mbështet transaksionet OLTP të cilat veprojnë në aplikacionet me performancë të lartë.

Në të shumtën e rasteve mos efikasiteti i indeksimit primar në qasjen e të dhënave, bënë që të kemi në dispozicion indekset sekondare si ndihmesë në qasjen më efikase të të dhënave. Kemi dy lloje të indekseve sekondare të cilat janë: indeksi sekondar lokal (ang. local secondary index) dhe indeksi sekondar global (ang. global secondary index) [33] [36].

Indeksi lokal sekondar (dytësor) - paraqet indeksin i cili përmban të njejtin çelës të ndarjës sikur tabela bazë e të dhënave. Secila ndarje e tabelës bazë përmban çelësat (ang. keys) e njejtës tabelë bazë, indekset janë menaxhuar në bazë të ndarjeve të tabelës bazë [37] [33].

Indeksi global sekondar (dytësor) – dallimi nga indeksi lokal qëndron në atë se këtu mund të kemi çelësa primar të cilët mund të jenë të ndryshëm nga ato në tabelën bazë. Indeksi global sekondar lejon përzgjedhësit të përfshijnë të gjitha të dhënat nga tabela bazë, nga të gjitha ndarjet e tyre [37] [33]. Gjithashtu, indeksi global sekondar përdoret edhe për kërkimin e teksit të plotë (ang. full-text search). [33].

Qëllimi i përdorimit të indekseve sekondare është për të përmirësuar qasjen e shpejtë dhe efikase në të dhënat, në procesimin e transaksioneve OLTP [33].

Te baza e të dhënave si DynamoDB, indeksi global sekondar përmirëson performancën e operacioneve për leximin dhe shkrimin e të dhënave, kur kemi të bëjmë me rritjen e papritur në masë të madhe të të dhënave [36]. Gjithashtu me rënien e rrjedhjes së të dhënave, DynamoDB përmes përshkallëzimit automatik do të mundësojë të ruajë kapacitetin e pa përdorur duke ulur throughput-in [36].

Detyra e planifikimit të ekzekutimit të përzgjedhësve (ang. queries) ka të bëjë me gjetjen e një plani për optimizimin e përzgjedhësve, që do të ulte koston e ekzekutimit të tyre [33].

Për arsye se disa lloje të bazave të të dhënave NoSQL përdorin çelësin primar (ang. primary key) për qasjen në të dhënat, mund të paraqesin vështirësi në optimizimin e përzgjedhësve [33].

Llojet e bazave të të dhënave të bazuar në NoSQL paraqesin performancë më të mirë për të përmbshuar disa nga kërkesat jo funksionale. Duke pasur parasysh që bazat e të dhënave NoSQL ofrojnë përshkallëzueshmëri, disponueshmëri dhe throughput më të lartë në raport me bazat e të dhënave bazuar në SQL, në të shumtën e rasteve. Tabela e mëposhtme përshkruan disa nga llojet e bazave të të dhënave të bazuar në NoSQL dhe një në SQL.

	Scan Queries	ACID Transactions	Conditional Writes	Joins	Sorting	Filter Queries	Full-Text Search	Analytics	Data Scalability	Write Scalability	Read Scalability	Elasticity	Consistency	Write Latency	Read Latency	Write Throughput	Read Availability	Write Availability	Durability	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared-Disk	Transaction Protocol	Sync. Replication	Async. Replication	Primary Copy	Update Anywhere	Logging	Update-in-Place	Caching	In-Memory Storage	Append-Only Storage	Global Indexing	Local Indexing	Query Planning	Analytics Framework	Materialized Views	
MongoDB	X		X		X	X	X	X	X	X			X	X	X	X	X		X	X					X	X	X		X		X	X	X							
Redis	X	X	X							X			X	X	X	X	X		X								X	X		X		X								
HBase	X		X		X			X	X	X	X	X	X			X			X	X					X		X		X		X			X						
Riak							X	X	X	X	X	X		X	X	X	X	X	X	X	X		X				X		X	X	X	X	X			X	X		X	
Cassandra	X		X		X		X	X	X	X	X	X		X		X	X	X	X	X	X						X		X	X	X	X	X		X	X	X		X	
MySQL	X	X	X	X	X	X	X	X		X		X							X					X			X	X		X	X	X				X	X			

Tab 1. Bazat e të dhënave të bazuar në NoSQL dhe SQL në raport me kërkesat e ndryshme (Burimi: [33]).

Nga tabela e mësipërme vërejmë se MongoDB dhe Hbase mbështesin konsistencë më të zhvilluar se sa llojet e tjera të bazave të të dhënave NoSQL. Duke mbështetur më shumë kërkesat më funksionale që nga shqyrtimi i përzgjedhësve (ang. Scan queries) deri te çështjet analitike (ang. Analytics). Ndërsa Cassandra dhe Riak ofrojnë mbështetje më shumë për kërkesa jo funksionale të cilat paraqiten nga përshkallëzueshmëria e të dhënave (ang. data Scalability) deri te qëndrueshmëria (ang. durability). Për shkak të përdorimit të memorjes kryesore (ang. main memory) të menaxhimit me të dhënat, Redis rezulton të jetë baza e të dhënave NoSQL me throughput-in më të lartë nga të gjitha bazat e të dhënave të paraqitura [33].

3.2 Transaksionet OLTP të cilat veprojnë në sistemet e shpërndara

Një transaksion në sistemet e shpërndara paraqet përbërjen e shumë operacioneve që rrjedhin nga bazat e të dhënave, të cilat veprojnë në shume baza të të dhënave të shpërndara [12]. Shpesh herë kemi për të ndërtuar aplikacione të cilat kërkojnë përshkallëzueshmëri të lartë dhe fleksibilitet në menaxhimin e të dhënave. Bazat e të dhënave relacionale bazuar në SQL të cilat kërkojnë që të implementohen në tërësi vetitë ACID si dhe të ketë një konsistencë të plotë mbi transaksionet nuk janë të afta për të krijuar aplikacione të tilla (me përshkallëzim në nivelin më të lartë dhe fleksibile) [12]. Nga punimi [12], autorët Dharavath dhe Kumar kishin propozuar një kornizë (ang. framework) që ofron zgjidhje për ruajtjen e atomicitetit dhe izolueshmërisë së transaksioneve të cilat veprojnë me bazat e të dhënave NoSQL, në një sistem të shpërndarë. Hulumtimi ishte bërë duke

përdorur një model arkitekturor të përshkallëzuar, të quajtur “three - tier” i cili paraqet tri nivele: klientin, middle tier dhe bazat e të dhënave të shpërndara. Si shkas hulumtimi ishte kryer me bazat e të dhënave të llojit të bazuar në kolona. Hulumtimi rezultoi në mbështetjen e transaksioneve të përshkallëzuar duke përdorur bazat e të dhënave NoSQL. Përshkallëzueshmëria ishte arritur duke përdorur modelin e komunikimit “Master - Slave”, ku serverët punues (të njohur si slave) mundësojnë përshkallëzueshmërinë e transaksioneve. Serverët kryesor (ang. master) sigurojnë disponueshmërinë dhe besueshmërinë e sistemit. Ndërsa “two – phase commit” protokollin ishte përdorur për të siguruar atomicitetin e transaksioneve [12].

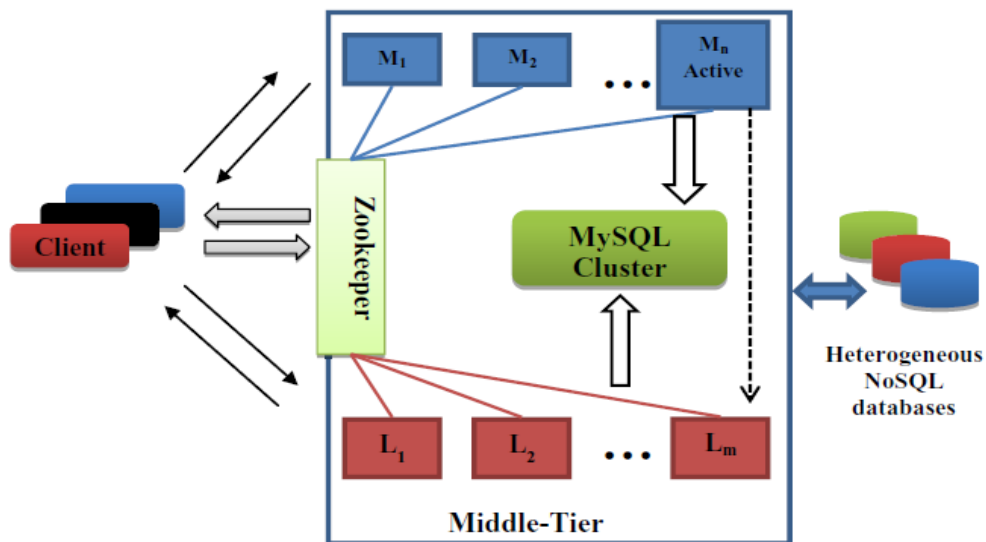


Fig 9. Menaxhimi i transaksioneve në një sistem të shpërndarë (Burimi: [12]).

Ilustrimi i mësipërm paraqet një model të menaxhimit të transaksioneve në një sistem të shpërndarë i cili ofron përshkallëzueshmërinë dhe atomicitetin e transaksioneve. Sistemi bazohet në modelin e komunikimit “Master-Slave” ku kemi një apo disa server kryesor (ang. master), detyra e të cilëve është për të menaxhuar dhe kontrolluar serverët e tjerë punues. Një nga elementet kryesore është Zookeeper, detyra e të cilit është të koordinojë proceset e aplikacioneve të shpërndara [12]. Apache Zookeeper gjithashtu ofron shërbime për aplikacionet e shpërndara ,si: sinkronizimin e bazave të të dhënave të shpërndara, ruajtjen e informacionit të konfigurimit, emërtimet dhe një grup të shërbimeve të tjera [38]. Zookeeper-i ndihmon në shërimin dhe përmirësimin e defekteve që mund të ndodhin gjatë sistemit të shpërndarë, si dhe lehtëson kompleksitetin e ndërtimit të aplikacioneve të shpërndara [38].

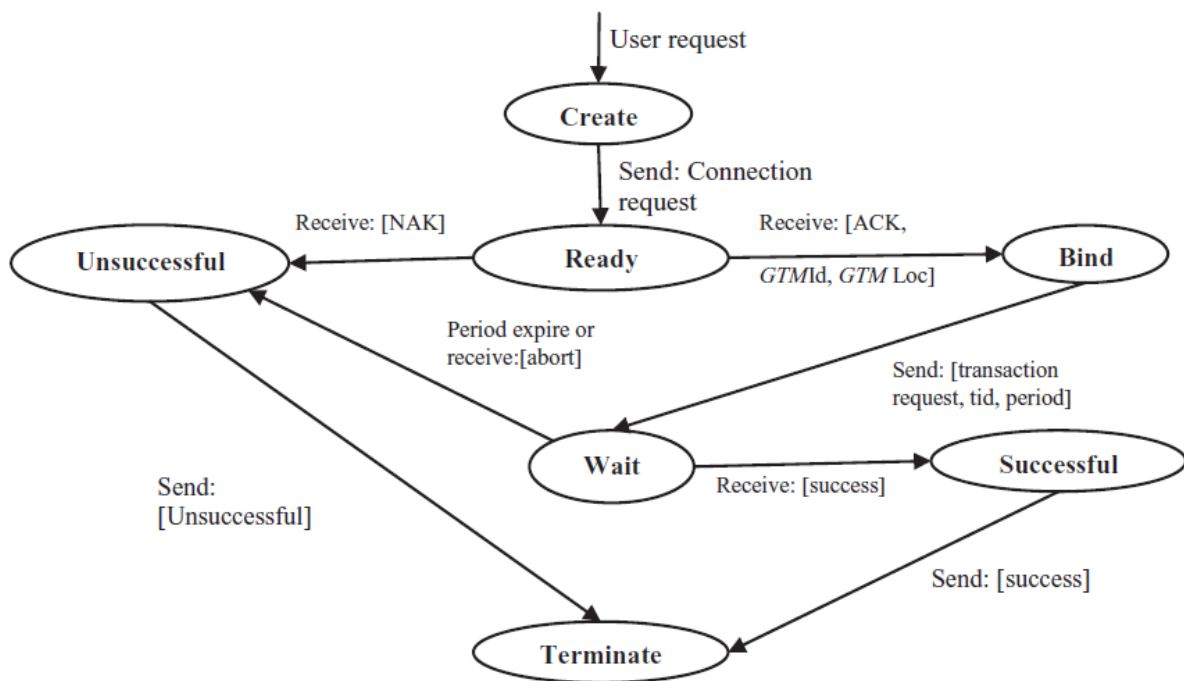


Fig 10. Procesi i lëshimit të kërkesës transaksionale nga klienti (Burimi: [12]).

Diagrami i mësipërm përshkruan procesin e lëshimit të një transaksioni nga **përdoruesi** si kërkesë. Në këtë rast nëse një përdorues dëshiron të lëshojë një kërkesë për transaksion, do të krijohet një proces i klientit. Procesi i klientit kërkon që të lidhet me një menaxher i cili është i aktivizuar (ang. master) bazuar në arkitekturën “Master - Slave”. Pasi të krijohet lidhja ndërmjet klientit dhe menaxherit të serverëve (ang. master) kalohet në gjendjen e pritjes (ang. wait) si në diagram, nga ku klienti pret për dërgimin e kërkesës për transaksionin e caktuar te serverët përkatës (ang. slave). Nëse të gjitha hapat kryhen në mënyrë të suksesshme dhe nuk paraqitet ndonjë defekt, procesi i dërgimit të kërkesës transaksionale të klientit përfundon si i suksesshëm (në pikën “Successful”). Përndryshe nëse nuk mund të procesohet në disa nivele të diagramit të mësipërm, për shkak të gjendjes së menaxherit të serverëve apo kërkesave tjera, procesi përfundon si i pa suksesshëm [12]. Detyra e procesit i cili kryhet nga klienti është të paraqesë transaksionin nga përdoruesi te shtresat tjera për procesim, si dhe të mbledhë rezultatin për transaksionin e procesuar.

Transaksionet procesohen nga një menaxhues global i transaksioneve i cili merret me kërkesat transaksionale. Menaxhuesi global i transaksioneve (ang. GTM) kordinon operacionet me serverin kryesor (ang. master) për të procesuar transaksionet, si dhe siguron ruajtjen e integritetit të të dhënave [12]. Gjithashtu, përmes algoritmit të menaxhuesit global të transaksioneve është arritur përshkallëzueshmëria e transaksioneve, e cila mund të veprojë në më shumë serverë njëherësh [12].

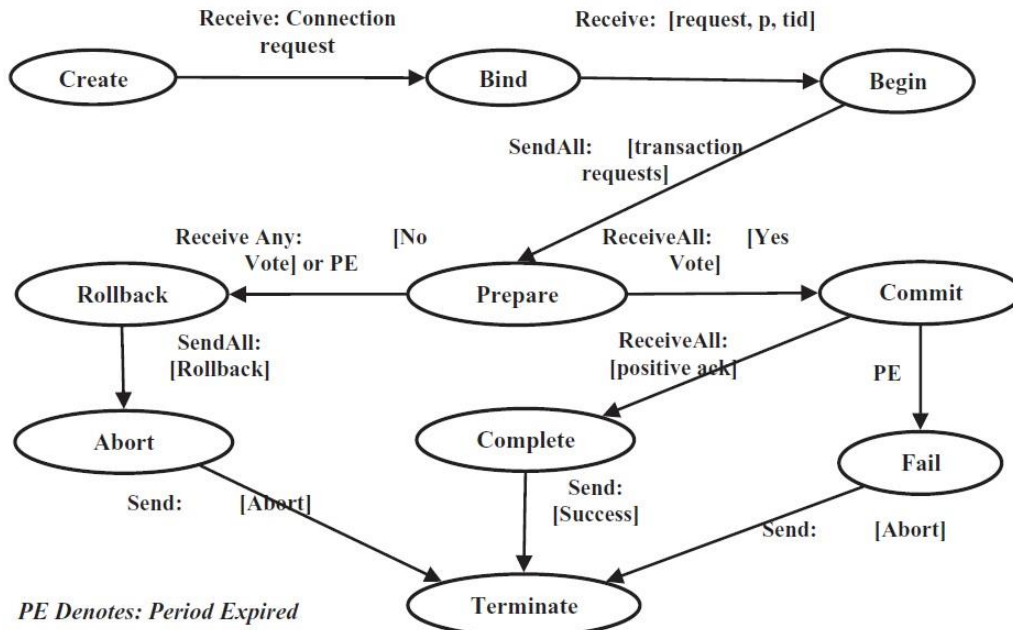


Fig 11. Procesi i lëshimit të kërkesës transaksionale nga klienti (Burimi: [12])

Pasi kërkesa e klientit procesohet nga menaxheri kryesor (ang. master) i serverëve, serverët punues (ang. slave) krijojnë një proces që quhet menaxhimi global i transaksioneve. Diagrami për këtë shtresë është paraqitur në ilustrimin e mësipërm, detyra e të cilit është të procesoj kërkesat transaksionale deri te serverët përkatës. Nëse të gjitha bazat e të dhënave përkatëse janë në gjendje të procesojnë kërkesat transaksionale nga menaxhuesi global i transaksioneve, atëherë transaksioni i lëshuar do të përfundojë me sukses fazën e dytë të procesimit. Përndryshe nëse një nga serverët e bazave të të dhënave nuk është në gjendje të procesojë kërkesën e lëshuar nga menaxhuesi global i transaksioneve, atëherë i gjithë procesi do të abortohet dhe do të ndërpritet.

Në shtresën e fundit të algoritmit, zhvillohet procesi i serverit të bazës së të dhënave, i cili pranon kërkesën transaksionale nga menaxhuesi global i transaksioneve.

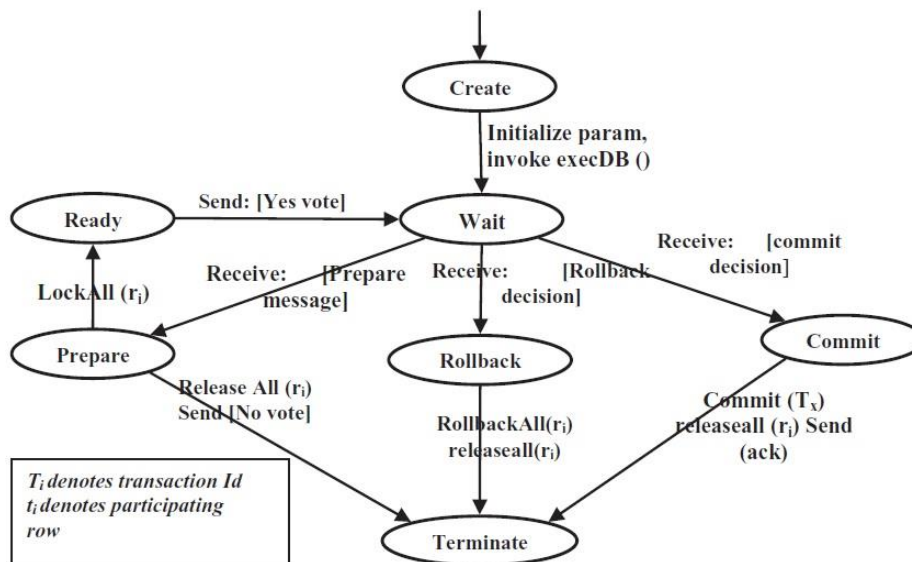


Fig 12. Procesi i lëshimit të kërkesës transaksionale nga klienti (Burimi: [12])

Serverët e bazave të të dhënave do të finalizojnë kërkesën transaksionale të lëshuar nga menaxhuesi global i transaksioneve. Ky proces i serverëve të bazës së të dhënave (shtresës së fundit të arkitekturës tri-shtresore) do të veprojë në të dhënat e kërkuara. Nëse serverët përkatës do të kryejnë procesimin e transaksionit në mënyrë të suksesshme (ang. commit), përdoruesi do të marrë përgjigje pozitive të transaksionit të kryer. Në të kundërtën nëse serverët nuk janë në gjendje të procesojnë transaksionet (apo nën transaksionet që veprojnë në më shumë baza të të dhënave), procesi do të përfundojë si i pa sukseshëm dhe do të kthehet përgjigje negative për përdoruesin e caktuar.

Protokolli i arkitekturës treshtresore garanton një proces të përshkallëzuar disponueshëm për të procesuar transaksionet në sistemin e shpërndarë duke pasur parasysh ruajtjen e atomicitetit dhe izolueshmërisë së transaksioneve.

3.3. Transaksionet OLTP për llojet e bazave të të dhënave NoSQL

Siç e përmendëm edhe më lart, se në bazat e të dhënave NoSQL, kur kemi të bëjmë me volum të madh të të dhënave dhe me të dhëna shumë heterogjene, ato mbështesin një disponueshmëri dhe shpejtësi të lartë në procesimin e të dhënave të tilla [39]. Por, mangësia qëndron në ruajtjen e integritetit të të dhënave gjatë procesimit të transaksioneve bazuar në llojet e ndryshme të NoSQL-ve [39].

3.3.1 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në Kolona

Bazat e të dhënave të llojit të bazuar në kolona paraqesin vështirësi në procesimin e transaksioneve të cilat veprojnë në shumë rreshta (ang. multi – row). Procesimi i transaksioneve në shumë rreshta ka të bëjë me atë se operacionet transaksionale që veprojnë në dy ose më shumë rreshta janë grupuar dhe procesuar si një njësi e vetme [39].

Lloji i bazuar në kolona përmban një model i cili paraqet ruajtjen e të dhënave si një grup i rreshtave dhe kolonave [16]. Lloji i bazuar në familjen e kolonave (ang. Column Family) paraqet një strukturë që përmban një numër të pakufizuar (të shumtë) të rreshtave [6].

Bazat e të dhënave të llojit të bazuar në kolona janë të ndara në koleksione të kolonave individuale, ku secila kolonë është e ruajtur individualisht në disk [13]. Ky model i ruajtjes së të dhënave do të mundësojë transaksionet të veprojnë vetëm në atributet të cilat janë të nevojshme. Pra, në këtë rast menjanojmë nevojën për leximin e tërë diskut të të dhënave, duke përmirësuar bandwidth-in e shfrytëzimit të mermorjës kryesore për procesim [13].

Pasi që përmes llojit të bazuar në kolona mund të ndajmë kolonat me attribute të përbashkëta të ashtuquajtura si tabela, kërkesat transaksionale për leximin e të dhënave të cilat veprojnë në më shumë tabela të kolonave janë më komplekse. [40].

Në punimin [40], autorët D.Bermbach, S.Muller, J.Eberhardt dhe S.Tai, kanë propozuar disa zgjidhje skematike për optimizimin e përzgjedhësit për leximin e të dhënave (ang. OLTP – Read). Autorët në fjalë kishin zhvilluar disa metoda për reduktimin e operacioneve për leximin e të dhënave, duke bërë bashkimin e disa përzgjedhësve (ang. queries) në këtë aspekt, të cilat mund të prodhojnë rezultat të njejtë [40]. Kështu që disa tabela të krijuar nga disa përzgjedhës janë bashkuar për të reduktuar numrin e përzgjedhësve të cilët mund të jenë edhe të pa nevojshëm. Dy nga bazat e të dhënave të llojit të bazuar në kolona, si: Cassandra dhe HBase janë optimizuar për të procesuar transaksionet që veprojnë në grupe të mëdha të të dhënave [41].

Sipas punimit [24] të J.Klein së bashku me një grup autorësh, gjatë vlerësimit të bazave të të dhënave të llojeve të ndryshme në bazë të kushteve të caktuara kishin konkluduar se Cassandra ofron performancë më të mirë dhe më efikase se llojet e tjera të NoSQL bazave, në procesimin e transaksioneve OLTP për lexim (ang. read) dhe shkrim (ang. write). Kjo bazohet në arkitekturën “peer-to-peer” dhe veçoritë që përmban për qendrën e të dhënave (ang. data center) [24].

3.3.2 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në Dokumente

Bazat e të dhënave të llojit të bazuar në dokumente përmbajnë një strukturë më komplekse të të dhënave, ku secili dokument është i identifikuar me një çelës unik [41]. Për dallim nga çelës/vlera bazat e të dhënave, lloji i bazuar në dokumente ruan vlerat për atributet e tij në bazë të formateve si JSON dhe XML [42]. Kur punojmë me aplikacione në të cilat të dhënat janë ndryshuar shpesh herë atëherë lloji i bazuar në dokumente është një zgjedhje e duhur [42]. Si në bazat e tjera të të dhënave të bazuar në NoSQL edhe te lloji i dokument operacioni për bashkimin e të dhënave si “JOINS” nuk është mbështetur. Kjo për arsye se modeli i të dhënave në fjalë përdor dokumentet e të dhënave të mbivendosura apo të ngulitura (ang. embedded), prandaj nuk kemi nevojë për kërkesat transaksionale që lidhin (ang. JOIN) dokumentet e tjera si në bazat e të dhënave relacionale SQL [42] [43]. Procesimi i përzgjedhësve më kompleks në MongoDB bëhet duke marrë informacionin shkallë – shkallë në bazë të disa hapave [42].

Për të rregulluar procesimin konkurent të transaksioneve dhe sigurimin e konkurrencës optimistike, Hbase-ja dhe MongoDB-ja përdorin një mënyrë e cila menaxhon dhe ruan versionin (kohën e përditësimit) e çdo të dhëne e cila quhet timestamp [44].

Procesimi i transaksioneve të shumta në mënyrë konkurente varet nga modeli i biznesit për aplikacionin e ndërtuar. Disa aplikacione të ndërtuara kërkojnë konsistencë më strikte të të dhënave, ndërsa disa të tjera vetëm konsistencë eventuale [44]. Për të menaxhuar përditësimin (ang. update) e grupeve të të dhënave me volum të madh MongoDB përdor protokolin “Two – phase commit” [44]. Protokoli Two – Phase commit është përdor në MongoDB për të ofruar një metodë për procesimin e transaksioneve në mënyrë atomike si dhe për të krijuar një mjedis që do të shërojë (ang. recover) transaksionin në rast

dështimi [44]. Pavarësisht përdorimit të protokollit “two – phase commit”, MongoDB-ja paraqet vështirësi në procesimin e transaksioneve që veprojnë në mënyrë konkurrenente. Vështirësitë paraqiten në humbjen e vetisë së atomicitetit, që shkakton të humbet edhe konsistenca e të dhënave, për të cilat është procesuar transaksioni [44].

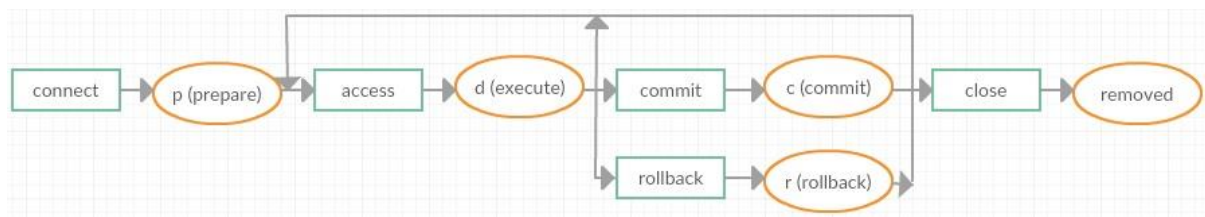


Fig 13. Struktura e procesimit të transaksionit bazuar në llojin dokument (Burimi: [44])

Struktura algoritmitike e paraqitur më sipër, tregon kalimin nëpër hapa të ndryshum të procesit të transaksionit. Në hapin e parë (ang. connect) i cili tregon se transaksioni duhet lidhur me bazën e të dhënave dhe pastaj për t’u procesuar për më tutje. Mbasi është kryer manipulimi i të dhënave (hapi, ang. execute), në rast të procesimit të suksesshëm (përshkak edhe të anomalive të cekura më lart), transaksioni do të përfundojë me sukses (në hapin, ang. commit.). Përndryshe në rast të ndonjë dështimi gjatë procesimit të transaksionit ai do të kthente bazën e të dhënave në gjendjen e mëparshme (hapi, ang. rollback) [44].

Hbase-ja, një bazë e të dhënave bazuar në llojin dokument, mbështet transaksionet OLTP të procesuara në kohë reale. (ang. Realtime) [45]. Transaksionet, përditësimet e të dhënave, janë vendosur në memorjën kryesore dhe në mënyrë periodike janë shkruar në disk për ruajtje të përhershme [46]. Ngjajshëm sikurse me MongoDB-në, Hbase-ja përdor operacione për ruajtjen e kontrollit të konkurrencës optimistike [46]. HBase si një nën projekt i Hadoop-it, mundëson marrjen e shpejtë të të dhënave, e cila mbështet ueb – aplikacionet me performancë më të lartë [19].

3.3.3 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në çelës/vlerë (ang. key/value)

Bazat e të dhënave të llojit të bazuar në çelës/vlerë përmbajnë një arkitekturë të thjeshtë të ndërtimit të tyre, në të cilat procesimi i transaksioneve është më i lehtë krahasur me llojet e tjera të NoSQL. Me arkitekturën çelës / vlerë të ndërtuar, mënyra e vetme për të qasur të dhënave është përmes çelësit i cili përmban vlerën e caktuar. Modeli çelës/vlerë është ndërtuar për të trajtuar transaksionet apo qasjet të cilat veprojnë në mënyrë shumë konkurrenente nga shumë përdorues [1]. Çelës/vlera është një pikënisje për modelet e tjera të NoSQL të cilat si qëllim kanë thjeshtësinë e ndërtimit të aplikacioneve, përshkallëzueshmërinë dhe ofrimin e performancës sa më të lartë. Siç e cekëm edhe më lart, për arsye të thjeshtësisë së modelit çelës/vlerë, ruajtja dhe marrja e të dhënave kryhet

në mënyrën më të shpejtë. Për të lehtësuar marrjën e të dhënave, çelës/vlerë përdorin funksionin “Hash” (perzirje) për të gjetur vendndodhjen e të dhënave [1].

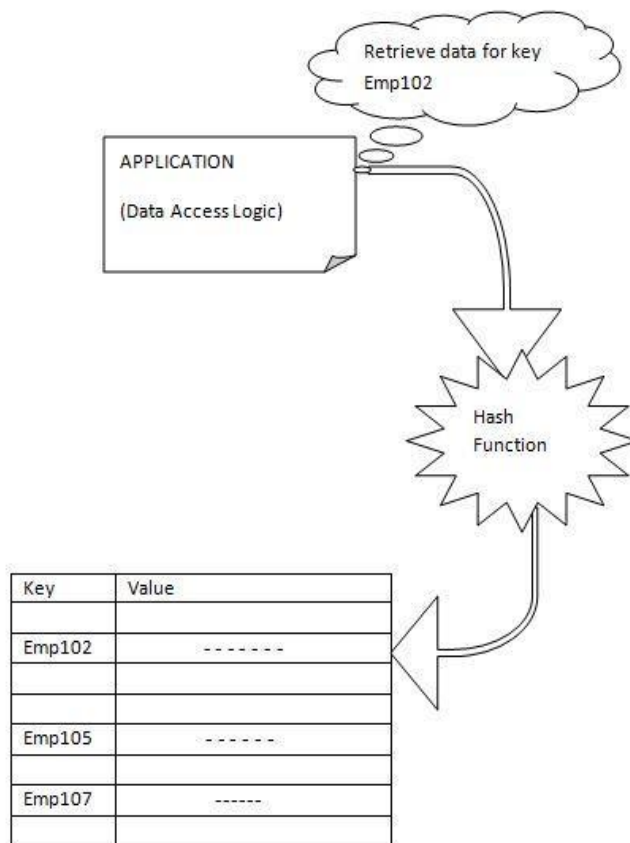


Fig 14. Leximi i të dhënave duke përdorur “Hash” funksionin (Burimi: [1])

3.3.4 Procesimi i transaksioneve OLTP në bazat e të dhënave të llojit të bazuar në graf

Pasi bazat e të dhënave të llojit të bazuar në graf (ang. graph – based) janë të përshtatshme për objektet të cilat janë të ndërlidhura në bazë të attributeve të caktuara, transaksionet veprojnë në nivel të lidhjeve ndërmjet tyre [47]. Këto ofrojnë strukturim të volumit të madh të të dhënave, që paraqiten të lidhura ndërmjet veti dhe semantikisht të ndërtuara mirë [48].

Modeli i llojit të bazuar në graf paraqet një sistem të menaxhimit të bazave të të dhënave përmes internetit (në kohë reale, ang. online), të cilat mbështesin transaksionet OLTP. Bazat e të dhënave në fjalë janë projektuar që të ofrojnë integritet transaksional, përveq mbështetjes së performancës së transaksioneve [49]. Bazat e të dhënave të llojit të bazuar në graf janë projektuar për kërkimet ndërmjet entiteteve lokalisht të vendosura [48]. Mënyra më superiore për kërkimin e të dhënave përmes llojit të bazuar në graf është kërkimi i të dhënave nga një nyje e vetme e grafit, duke vazhduar sistematikisht në nyjet tjera të tërë grafit të ndërtuar.

Për të ngritur performancën dhe për të ruajtur strukturën e modelit Graf, disa nga bazat e të dhënave përdorin metodat e indeksimit që të bëjnë kërkimin më të shpejtë të përzgjedhësve (ang. queries [49]. Përmes metodës së indeksimit kemi mundësi të kërkojmë një grup të entitetve të të dhënave në tërësi (përzgjedhësi -myEntity.findAll) në vend se të kërkojmë secilin entitet veçmas [48]. Qëllimi i përdorimit të indeksimit të graf-i është për të kërkuar një nyje (ang. nodes) ose lidhje (ang. relationship) specifike, duke përshkuar rrugën në mënyrë direkte [49].

Neo4J-ja, një nga bazat e të dhënave të llojit të bazuar në graf ndërtuar në Java, e cila ofron performancë dhe përshkallëzueshmëri. Në Neo4j-në klasat kryesore të cilat janë përdor për të krijuar një model të grafit janë: nyjet (ang. nodes) dhe lidhjet (ang. relationship), duken pasur paraysh vetitë e tyre [48]. Cypher-i një nga gjuhët e bazuar në modelin graf e cila ofron sintaksë deklarative mbështet transaksionet ACID e cila përdor SQL. Cypher-i mundëson që përzgjedhësit shumë të komplikuar të shprehen në një mënyrë shumë të lehtë [48].

4. PUNA EKSPERIMENTALE

4.1. Platforma HADOOP për ndërtimin e një rrethine të shpërndarë

Pasi pjesa teorike sqaron mënyra të ndryshme të ngritjes së një rrethine të shpërndarë, detyra praktike e mëposhtme do të paraqesë ngritjen e një rrethine të tillë përmes platformës Hadoop.

Hadoop-i mundëson përpunimin e shpërndarë të grupeve të mëdha të të dhënave përmes disa koleksioneve të kompjuterëve (nyjeve) të cilat kanë karakteristika të ngjajshme. Hadoop-i bënë ruajtjen dhe procesimin e të dhënave në një rrethinë të shpërndarë. Hadoop-i mundëson përshkallëzueshmërinë horizontale duke vepruar në mijëra server të ndryshëm [50]. Gjithashtu, posedon aftësitë për analizën e të dhënave në formate të ndryshme, duke filluar nga të dhënat e pa strukturuara, gjysmë të strukturuara deri te të dhënat e strukturuara (mirë të formatuara) [51].

Platforma Hadoop përmban tri komponente kryesore të cilat janë:

1. Shtresa e ruajtjes – HDFS (ang. Hadoop distributed file system)
2. Shtresa e procesimit - MapReduce
3. Shtresa për menaxhimin e resurseve – Yet Another Resource Negotiator

Shtresa e ruajtjes (ang. HDFS) - mundëson ruajtjen e besueshme për grupe të mëdha të të dhënave si dhe ofron qasje më të lehtë në të dhënat [51] [50]. HDFS-ja bënë ruajtjen e të dhënave në shumë nyje kompjuterike si dhe mundëson kopjimin e tyre për të ruajtur sistemin në rast dështimi [50].

Shtresa e procesimit (ang. MapReduce) – paraqet një strukturë softuerike e cila bënë procesimin e grupeve të mëdha të të dhënave në mënyrë paralele [51].

Shtresa për menaxhimin e resurseve (ang. YARN) – bënë menaxhimin e resurseve për aplikacionet e ndryshme të cilat veprojnë brenda Hadoop klasterit. Gjithashtu, bënë caktimin e detyrave të cilat do të ekzekutohen në nyje të ndryshme [52].

Për të testuar platformën Hadoop kemi ndërtuar një koleksion prej katër nyjeve virtuale duke përdorur paketën softuerike për virtualizim 'VirtualBox'. Brenda paketës VirtualBox për secilën nyje virtuale janë bërë konfigurimet harduerike si dhe është mundësuar komunikimi ndërmjet tyre. Secila nyje virtuale përmban konfigurim të njejtë harduerik dhe softuerik.

Eksperimenti në fjalë është kryer në një laptop me procesor core i7 – 7500U, me 16GB RAM, që përmban hard drive prej rreth 1 TB (ang. Terabyte), nga i cili janë simuluar katër paketa të sistemit operativ Linux Ubuntu 14.04.5 LTS.

Platforma Hadoop së bashku me komponentët e saj është ndërtuar në gjuhën programuese Java. Për të mundësuar funksionimin e komponenteve brenda platformës Hadoop, në secilën nga nyjet punuese është instaluar dhe konfiguruar Java.

Në vazhdim, platforma Hadoop është shkarkuar në një version të caktuar e cila është instaluar. Pas instalimit të Hadoop-së, janë bërë konfigurimet e variablave të tij brenda sistemit operativ Linux Ubuntu.

Në mënyrë që të ndërtojmë një rrethinë të shpërndarë brenda platformës Hadoop duhet të inicializohen dy nga komponentët e saj si: NameNode dhe DataNode. NameNode është vendosur në nyjen kryesore Master, e cila ruan metadata për të gjitha DataNode [53]. NameNode është përgjegjës për të procesuar kërkesat nga përdoruesit të cilët dëshirojnë për të kërkuar, shtuar apo fshirë ndonjë fajll [54].

DataNode ruan të dhënat në sistemin e file-ve të Hadoop (ang. HDFS), e cila vepron në nyjet e tjera si slave. Në rast se DataNode nuk është në dispozicion apo dështon për ndonjë arsye, nuk do të ndikojë në Hadoop klasterin, pasi NameNode (si nyje kresore) do të bëjë replikimin e të dhënave për DataNode të tillë [53].

Pasi është bërë konfigurimi i komponenteve kryesore të Hadoop-ës, klasteri do të jetë gati për testim. Për të ngritur klasterin Hadoop bazuar në konfigurimin e lartpërmendur duhet të startohen dy nga komponentet kryesore si sistemi i file-ve të shpërndarë (ang. HDFS) si dhe menaxhuesi i resurseve (ang. YARN). Në rastin tonë do të rezultojë me klasterin e ngritur i cili përbën një nyje master si dhe tri të tjera slave.

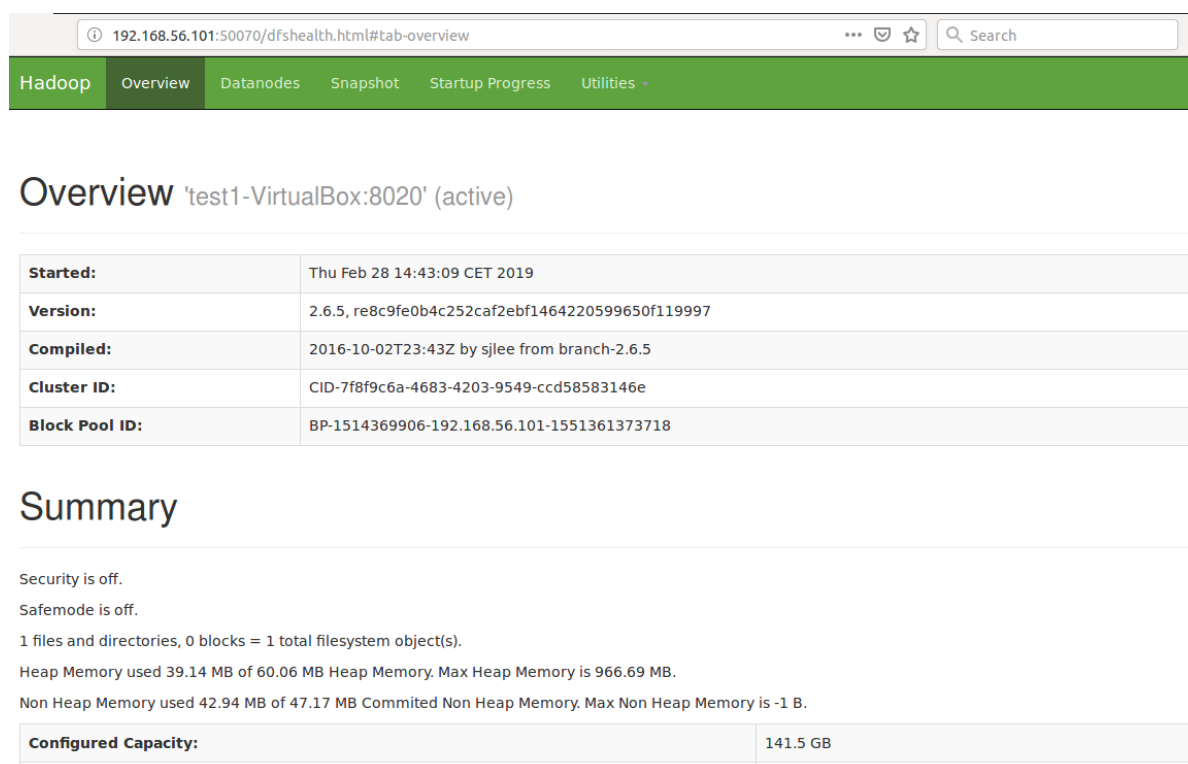


Fig 15. Të dhënat rreth klasterit Hadoop

Figura më sipër pasqyron ngritjen e klasterit Hadoop, ku secila nga nyjet punuese përmban një hapsirë të caktuar për ruajtjen e të dhënave. Hadoop klasteri i ngritur paraqet një rrethinë të shpërndarë dhe paralele e cila siguron vetinë e përshkallëzueshmërisë si dhe koston e ruajtjes së të dhënave

4.2. MongoDB

MongoDB-ja është projektuar për të ngritur një sistem i cili menaxhon me grupe të mëdha të të dhënave, i cila siguron disponueshmërinë dhe përshkallëzueshmërinë horizontale në mënyrë më të lehtë, ër të ofruar një mjedis për ndërtimin e aplikacioneve moderne [55].

MongoDB-ja përbën një kombinim të elementeve të bazave të të dhënave relacionale dhe ato të bazuar në NoSQL. Sistemi i ruajtjes në-memorje (ang. In-memory) ofron një qasje analitike për të dhënat në kohë reale për aplikacione të ndryshme [55].

Të dhënat në MongoDB janë të ruajtura si dokumente. Modeli i ruajtjes së të dhënave ofron një performancë shumë të lartë dhe optimizuese përmes mënyrës së dokumenteve të mbivendosura. Duke pasur parasysh këtë, manipulimi me të dhënat në një sistem i cili mbart përshkallëzueshmërinë horizontale është shumë më efikas dhe i shpejtë [55]. Si gjuhë kryesore (ang. by default) për përzgjedhësit MongoDB përdor sintaksën e gjuhës programuese JavaScript.

MONGODB NË NJË RRETHINË TË SHPËRNDARË

MongoDB-ja mundëson ndarjen e grupeve të të dhënave në më shumë nyje të vendosura në vende të ndryshme (qendra të të dhënave). Ndërtimi i një sistemi të shpërndarë në MongoDB arrihet përmes një teknike të quajtur Sharding. Teknika e sharding-ut (ndarjes) shpërndan të dhënat në nyje të ndryshme të cilat njihen si shards (pjesë të ndara) [56] [43]. Në MongoDB-në, sharding-u mund të jetë vendosur në nivel bazës së të dhënave apo në nivel të koleksionit të vetëm. Procesi i ndarjes së të dhënave si dhe balancimi i tyre në nyje të ndryshme bëhet në mënyrë automatike [57]. Mënyra e lëshimit të pyetësorëve (ang. queries) mbetet e njejtë pavarësisht numrit të shards, e cila tregon një transparencë nga ana e sharding-ut për aplikacionet e ndryshme [56]. Sipas punimit [57], autori Kristina Chodorow thekson tri nga qëllime kryesore të sharding-ut të cilat janë:

- 1.) Nyjet shards brenda klasterit duhet të jenë të abstraguara në raport me aplikacionet. Të gjitha operacionet për manipulimin e të dhënave në të gjitha shards janë bërë përmes instancës së MongoDB-së e quajtur *mongos router*, për shards të cilat janë të lidhura për të. Kështu që, përdoruesit e aplikacioneve nuk kanë nevojë për të ditur për vendndodhjen e shards. Qëllimi i *mongos router* është për të pranuar kërkesa

nga aplikacionet për manipulimin e të dhënave dhe për të kthyer përgjigjen adekuate tek ato [57].

2.) Klasteri duhet të jetë gjithmonë në dispozicion për t'u përgjigjur kërkesave për leximin dhe futjen e të dhënave

Për të arritur këtë qëllim, MongoDB-ja përdor replikimin për secilin nga grupet e të dhënave për shards [57].

3.) Resurset brenda klasterit duhet të shtohen në mënyrë proporcionale

Në bazë të rritjes së të dhënave brenda klasterit si dhe kërkesave nga aplikacionet, duhet të shtohet kapaciteti për ruajtjen dhe menaxhimin e tyre [57].

Replikimi paraqet një pjesë të rëndësishme në ndërtimin e një rrethine të shpërndarë të kompjuterëve [3]. Me replikimin e të dhënave arrihet disponueshmëria e të dhënave, që siguron vetinë e tolerimit të gabimeve në rast dështimi të ndonjë nyje kompjuterike apo dëmtimit të të dhënave nga palët e treta. Replikimi i të dhënave në server të ndryshëm (të cilat ndodhën në vende të ndryshme) do të mundësonte një disponueshmëri të të dhënave për aplikacionet e shpërndara. Gjithashtu, do të mundësonte përdoruesve të ndryshëm t'u qasen të dhënave për lexim në serverë të ndryshëm.

Replikimi në MongoDB paraqet një arkitekturë të tillë që secila nyje përmban kopje të njëjta të të dhënave. Por brenda nyjeve (serverëve) duhet të jetë një nyje e cila do të konsiderohet si kryesore ndërsa të tjerat do të jenë nyje dytësore (sekondare). Vetëm nyja primare është ajo që pranon operacionet për shkrimin e të dhënave (ang. write) [58].

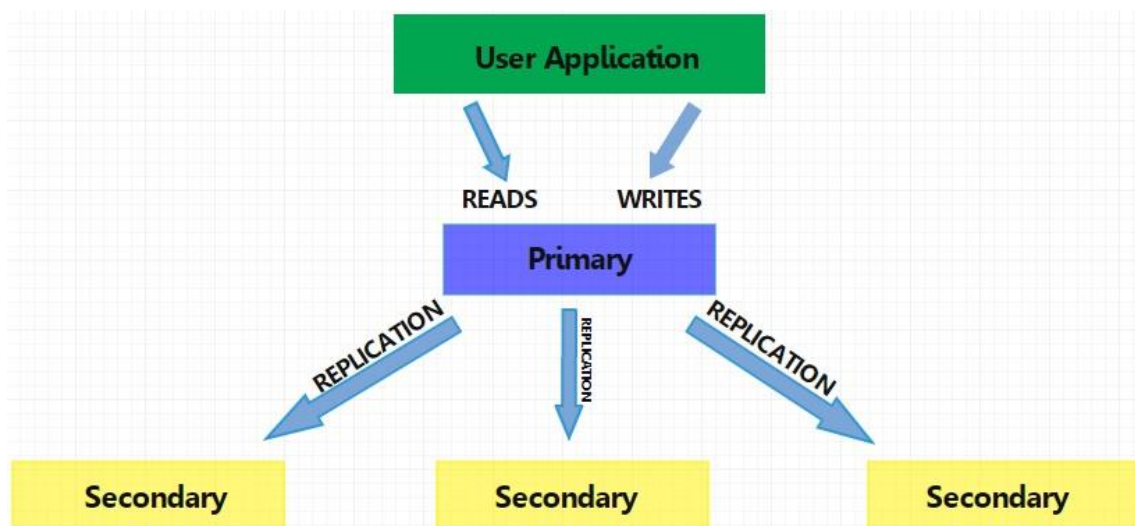


Fig 16. Procesi i Replikimit në bazë të ndërtimi të klasterit

Nyjet dytësore (sekondare) ruajnë të njëjtat të dhëna sikur nyja kryesore (primare). Në rast se nyja kryesore dështon për arsye të ndryshme dhe nuk është në dispozicion, një nga nyjet sekondare e cila është më e pranueshme (për të qenë nyje kryesore) do të vihet në vend të nyjës kryesore [58] [3].

4.3. Metodologjia e hulumtimit

Në këtë pjesë të punimit do të paraqesim analizën empirike të testimeve për bazën e të dhënave MongoDB-së në rrethinë të shpërndarë. MongoDB-ja paraqet veti të mira optimizuese në menaxhimin dhe manipulimin e grupeve të mëdha të të dhënave. Si hap i parë është kryer instalimi dhe konfigurimi i MongoDB-së në të gjitha nyjet punuese. Klasteri për testimin e MongoDB-së përbën gjithësej tri nyje punuese.

Në MongoDB-në është krijuar një bazë të të dhënave e cila vepron në të gjithë klasterin. Brenda bazës së të dhënave është krijuar një koleksion i cili përmban grupe të të dhënave bazuar në raste të ndryshme.

Testimet për koleksionin e dokumenteve

Koleksioni i të dhënave paraqet një grup të dokumenteve të cilat mund të kenë fusha dhe të dhëna të llojeve të ndryshme [59]. Dokumentet brenda koleksionit përmbajnë të dhëna të cilat janë gjeneruar në mënyrë arbitrare (ang. random). Të dhënat brenda koleksionit paraqesin një dokument i cili përmban një rekord me dy vlera të caktuara. Të dhënat paraqesin vlera të ndryshme për secilin dokument të cilat mund të gjenerohen me lehtësi në bazën e të dhënave MongoDB. Për të mundësuar shpërndarjen dhe procesimin e të dhënave në një rrethinë të shpërndarë, brenda MongoDB-së është lejuar teknika e sharding-ut. Për testimet tona, teknika e sharding-ut është aplikuar në nivel të koleksionit.

Gjenerimi i të dhënave në MongoDB është bërë përmes gjuhës programuese Python. Gjuha programuese Python përbënë një model të programimit të orientuar në objekte e cila përmban një sintaksë të thjeshtë. Arsyeja për përdorimin e gjuhës programuese Python qëndron për faktin se Python paraqet një gjuhë dinamike në ndërtimin e aplikacioneve të cilat gjenerojnë grupe të mëdha të të dhënave [60].

Në raport me gjuhët e tjera programuese Python del të jetë shumë më e shpejtë në gjenerimin e grupeve të mëdha të të dhënave, e cila ofron një numër të madh të moduleve (librarive) [61]. Për të mundësuar manipulimin e grupeve të të dhënave në MongoDB përmes gjuhës programuese Python është përdorur moduli i quajtur *'pymongo'*.

Në testimet tonamanipulimi i të dhënave bëhet përmes katër operacioneve bazike të cilat janë: Krijimi, Leximi, Përditësimi dhe Fshirja. (ang. CRUD operations).

Koleksionet të cilat do të testohen përmbajnë mostra të caktuara të dokumenteve si në vijim: **Dokumentet për koleksion - 100, 1000, 5000, 10000, 50000, 100000.**

Skriptat të cilat përdoren për të manipulimin e të dhënave përbëjnë një kombinim të sintaksës së gjuhës programuese Python me sintaksën e gjuhës së përzgjedhësve të MongoDB-së. Secila nga operacionet CRUD është shkruar në një skriptë të veçantë në gjuhën programuese Python.

Për të realizuar objektivat e lartpërmendura, është kryer analiza krahasuese për

operacionet CRUD në MongoDB. Analiza krahasuese paraqet testimin e dy mjedise të ndryshme të konfigurimit të nyjeve:

- 1) Testimi i operacioneve CRUD në një nyje të vetme,
- 2) Testimi i operacioneve CRUD në tërë klasterin

Në vazhdim do të paraqesim rezultatet e arritura nga analiza e punës hulumtuese.

4.4. Analiza dhe interpretimi i rezultateve

Në fund të punës sonë eksperimentale do të paraqesim rezultatet e përgjithshme të procesimit të ' operacioneve CRUD të cilat veprojnë në bazën e të dhënave MongoDB, si në vazhdim:

INSERTIMI (ang. CREATE), $n(i)$ numri i inserimeve, $n(n)$ numri i nyjeve), koha e matur në sekondë,

$n(i)$ $n(n)$	100	1000	5000	10000	50000	100000
1	0.233	1.420	5.280	9.410	46.330	95.400
3	0.339	2.700	10.260	21.690	85.200	195.000

Tab 2. Të dhënat rreth operacionit për insertim

Grafikoni për operacionin e Insertimit

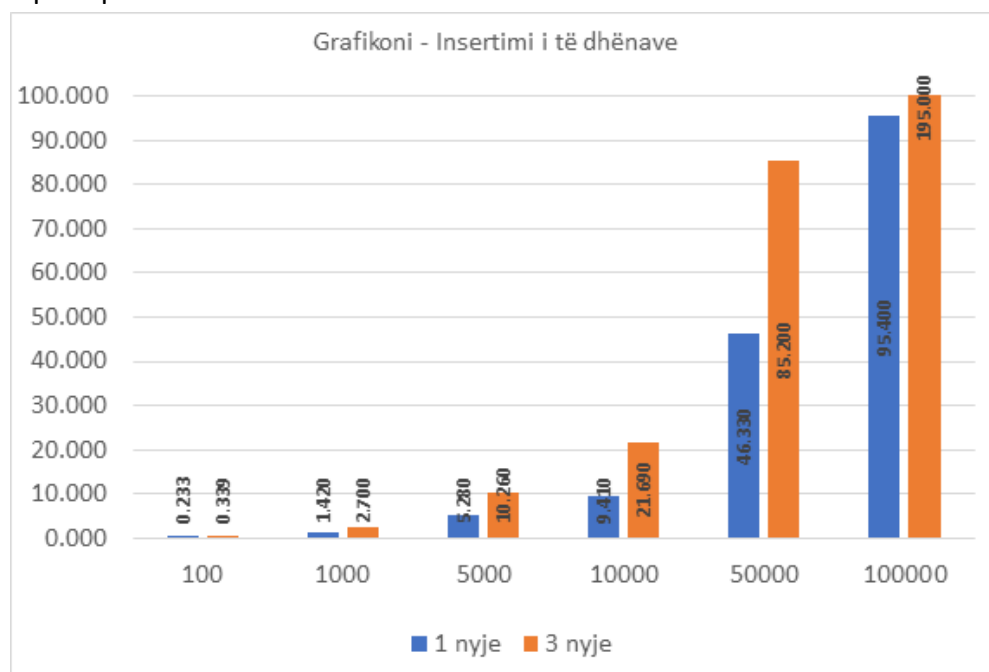


Fig 17. Grafikoni – Insertimi i të dhënave

LEXIMI (ang. READ), $n(i)$ numri i inserimeve, $n(n)$ numri i nyjeve), koha e matur në sekondë,

$n(i)$ $n(n)$	100	1000	5000	10000	50000	100000
1	0.165	1.490	6.600	13.560	64.200	129.000
3	0.203	1.740	7.490	14.780	64.800	126.000

Tab 3. Të dhënat rreth operacionit për lexim

Grafikoni për operacionin e Leximit

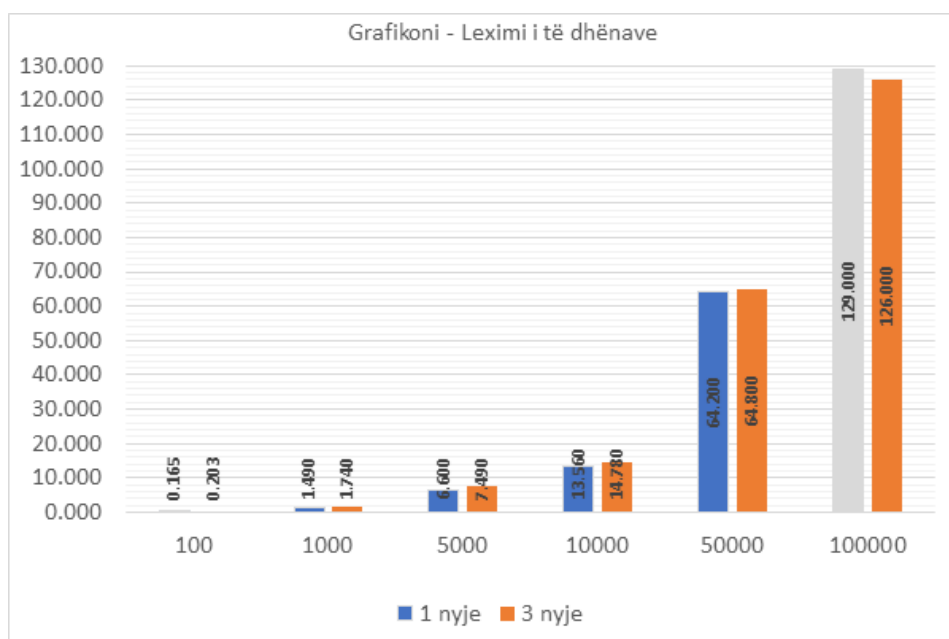


Fig 18. Grafikoni – Leximi i të dhënave

PËRDITËSIMI (ang. UPDATE), $n(i)$ numri i inserimeve, $n(n)$ numri i nyjeve), koha e matur në sekondë,

$n(i)$ $n(n)$	100	1000	5000	10000	50000	100000
1	0.0251	0.0373	0.1471	0.2812	0.9624	1.4700
3	0.0323	0.0427	0.0941	0.1593	0.7324	1.2600

Tab 4. Të dhënat rreth operacionit për përditësim

Grafikoni për operacionin e Përditësimit

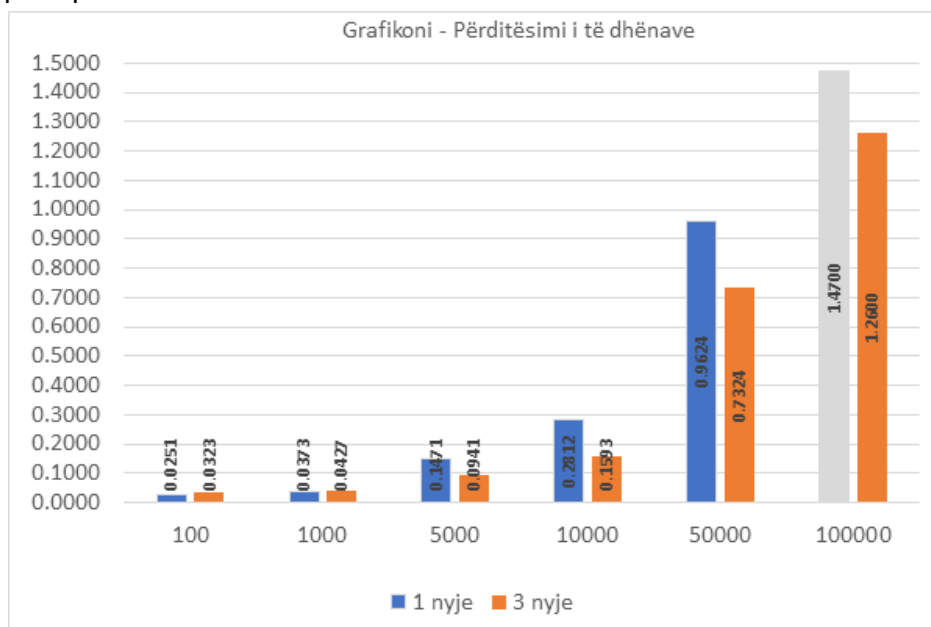


Fig 19. Grafikoni – Përditësimi i të dhënave

FSHIRJA (ang. DELETE), n(i) numri i inserimeve, n(n) numri i nyjeve), koha e matur në sekondë,

$\frac{n(i)}{n(n)}$	100	1000	5000	10000	50000	100000
1	0.0221	0.0417	0.2123	0.3945	1.0900	2.0700
3	0.0329	0.0429	0.0758	0.1889	0.8540	1.3500

Tab 5. Të dhënat rreth operacionit për fshirje

Grafikoni për operacionin e Fshirjes

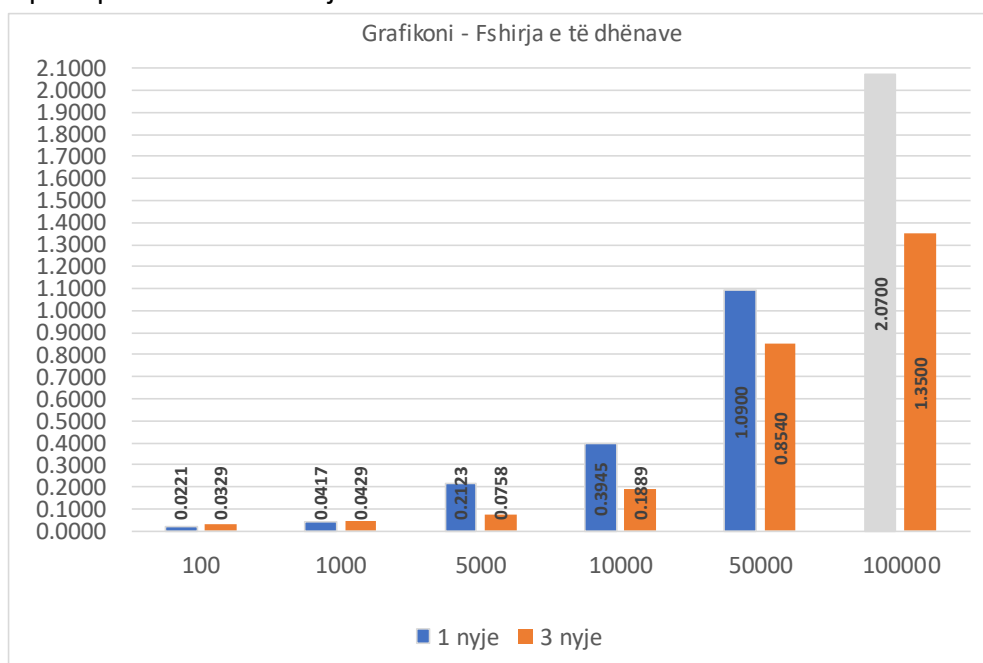


Fig 20. Grafikoni – Leximi i të dhënave

Rezultatet e mësipërme paraqesin kohën mesatare të ekzekutimit për secilin nga ‘ operacionet CRUD për dokumentet përkatëse.

REZULATET për për procesimin e operacioneve CRUD të kryera në MongoDB, duke pasur për bazë një rrethinë të shpërndarë janë paraqitur si në vijim:

- Në bazë të analizës së operacioneve CRUD në raport me ekzekutimin e koleksioneve të ndryshme, rezulton se operacionet për Insertim dhe Lexim paraqesin kohën maksimale në procesimin e të dhënave.
- Klasteri i përbërë nga tri nyjet rezulton në kohën më optimale (duke ulur kohën e ekzekutimit) për operacionet për Përditësim dhe Fshirje. Ndërsa te operacionet për Insertim dhe Fshirje, në përgjithësi paraqitet kohë më e madhe kur procesimi kryhet në një nyje të vetme.
- Rezultatet e mësipërme tregojnë që në përgjithësi në tri nyjet e klasterit, rritja e numrit të dokumenteve ndikon në uljen e kohës së ekzekutimit. Ky rezultat vlen për tri nga operacionet për Lexim, Përditësim dhe Fshirje.

- Dallueshmëria në kohën e ekzekutimit për koleksionet të cilat përmbajnë më pak dokumente është më e vogël në raport me koleksionet të cilat përmbajnë më shumë. Më këtë nënkuptojë që rritja e numrit të dokumenteve për shumëfish nuk rrit në të njëjtën mënyrë edhe kohën e ekzekutimit të tyre. Kjo mund të vërehet në veçanti te koleksionet të cilat përmbajnë 100 deri në 1000 dokumente. Por, gjithashtu këtë dallueshmëri të ulët në kohën e ekzekutimit mund ta vërejmë edhe te koleksionet e mëdha për disa operacione CRUD. Si për shembull, tek operacioni për Lexim, në procesimin e 50000 dhe 100000 dokumenteve.
- Në përgjithësi për shkak të thjeshtësisë së vlerave që përmbajnë dokumentet, koha për procesimin e të dhënave rritet në mënyrë lineare dhe paraqet kohë më optimizuese.

5. KONKLuzionet

Në këtë punim, ne adresuam disa nga problemet që kanë të bëjnë me bazat e të dhënave tradicionale në menaxhimin e grupeve të mëdha të të dhënave. Në fillim të temës përshkruam disa nga arsyet kryesore që kishin çuar në lindjen e bazave të të dhënave NoSQL, të cilat mundësojnë një menaxhim më efikas të këtij volumi të madh të të dhënave.

Bazat e të dhënave NoSQL ishin paraqitur si nevojë për kompanitë e mëdha të cilat kanë të bëjnë me rrjetet sociale, komerciale dhe të tjera të cilët kishin vështirësi në menaxhimin e një fluski të madh të të dhënave dhe përdoruesve. Për të realizuar qëllimet e bazave të të dhënave NoSQL disa teorema dhe koncepte ishin prezantuar për të krijuar një sistem të qëndrueshëm dhe sa më efikas në menaxhimin e grupeve të mëdha të të dhënave.

Pra, qëllimi kryesor i këtij punimi është specifikisht ofrimi i një sistemi i cili do të mundësonte përshkallëzueshmërinë, disponueshmërinë si dhe konsistencën në menaxhimin e të dhënave. Implementimi i këtij qëllimi ishte bërë duke pasur për bazë arkitekturën e një rrethinë të shpërndarë dhe paralele të kompjuterëve. Krijimi i një rrethine të shpërndarë dhe paralele është implementuar përmes platformës Hadoop.

Në vazhdim të punimit kemi përshkruar teorikisht të gjitha llojet e bazave të të dhënave NoSQL dhe se si ato mund të implementojnë objektivat e paraqitura gjatë këtij punimi.

Pas shqyrtimit të llojeve të NoSQL, janë prezantuar disa nga studim rastet e bëra nga autorë të ndryshëm, të cilat paraqesin rezultate shumë domethënëse dhe krahasuese. Si rrjedhojë për të testuar vetinë e përshkallëzueshmërisë, disponueshmërisë dhe konsistencës janë përdorur operacionet CRUD. Rezultatet e këtij punimi kanë të bëjnë me testimin e bazës së të dhënave MongoDB të bazuar në dokumente. Operacionet CRUD janë testuar për kohën e procesimit të tyre në një rrethinë të shpërndarë bazuar në koleksione të ndryshme të dokumente në MongoDB. Testimet janë kryer në dy rrethana: kur kemi një nyje punese dhe në rastin tjetër në tërë klasterin. Për të gjitha testimet e kryera janë paraqitur kohët mesatare të ekzekutimit.

Bazuar në rezultatet e punës sonë praktike mund të konkludojmë se klasteri i përbërë prej tri nyjeve ndikon në rritjen e shpejtësisë së ekzekutimit për operacionet CRUD të Leximit, të Përditësimit dhe të Fshirjes në një mesatare prej 25.51%. Ndërsa për operacionin CRUD të Insertimit, klasteri prej tri nyjeve rezulton në një ulje të kohës së ekzekutimit prej 53.10 %.

Gjithashtu, mund të konkludojmë se në përgjithësi për të gjitha operacionet CRUD në të cilat veprojnë tri nyjet punuese, rritja e numrit të dokumenteve ndikon në uljen e kohës së ekzekutimit.

Nga rezultatet e paraqitura më lartë shohim se modeli i të dhënave NoSQL në një rrethinë të shpërndarë bazuar në bazën e të dhënave MongoDB paraqet një procesim të të

dhënave më efikas, më të shpejt dhe me performancë më të lartë. Në bazë të këtyre konkludimeve të arritura, në modelimin e të dhënave përmes bazave NoSQL në një rrethinë të shpërndarë si dhe manipulimin e të dhënave bazuar në operacionet CRUD, mund të konstatojmë se pjesërisht pranoen hipotezat e parashtruara gjatë temës ku:

- Platforma Hadoop mundëson një organizim më efikas dhe të sigurtë të grupeve të të dhënave duke bërë alokimin e memorjes si dhe menaxhimin e resurseve për secilën nyje në mënyrë të barabartë (barasvlefshme).
- Shumica e llojeve të transaksioneve OLTP të cilat veprojnë në një rrethinë të shpërndarë paraqesin performancë më të mirë në kohën e ekzekutimit të tyre në raport me një sistem të vetëm. Përveç operacionit për insertim, i cili paraqet një performancë më të dobët kur kemi procesim të të dhënave në një rrethinë të shpërndarë.
- Klasteri i përbërë prej tri nyjeve përmbush dy nga vetitë e teoremës CAP, duke implementuar teknikën e sharding-ut mbi grupet e të dhënave. Përveç përshkallëzueshmërisë së të dhënave, sistemi garanton vetitë e konsistencës dhe tolerimit të ndarjes në nyje të ndryshme (si dy nga vetitë e -teoremës CAP).

Në fund mund të themi se modelimi i të dhënave përmes bazave NoSQL mundëson të kemi një sistem me performancë të lartë dhe të zbatueshëm kur kemi të bëjmë me procesimin e grupeve të mëdha të të dhënave.

BIBLIOGRAFIA

- [1] B. Sethi, S. Mishra and P. Patnaik, "A Study of NoSQL Database," *International Journal of Engineering Research & Technology*, vol. III, no. 4, pp. 1131-1135, 2014.
- [2] M. A B M and A. H. Syed, "NoSQL Database: New Era of Databases for Bid data Analytics - Classification, Characteristics and Comparison," *International Journal of Database Theory and Application*, vol. 6, pp. 1-13, 2013.
- [3] C. Strauch, "NoSQL Databases," p. 104, 2011.
- [4] G. Pang, "Scalable Transactions for Scalable Distributed Database," *EECS Department, University of California, Berkeley*, pp. 1-131, 2015.
- [5] D. Sullivan, NoSQL FOR MERE MORTALS, G. Wiegand, Ed., New Jersey: Pearson Education, 2015, p. 48.
- [6] J. Pokorny, "NoSQL databases: a step to database scalability in web environment," *International Journal of Web*, vol. IX, pp. 69-82, 2013.
- [7] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, Boston: Addison-Wesley, 2010.
- [8] J. Gamper, "Distributed Databases," SRV College of Education, Tamil Nadu , 2009.
- [9] Search SQL Server Group, "Search SQL Server," April 2012. [Online]. Available: <https://searchsqlserver.techtarget.com/definition/database-replication>. [Accessed 4 April 2018].
- [10] C. Strauch, "NoSQL Databases," pp. 1-149, 2009.
- [11] S. Kleinman, K. Kim, R. Kumar, S. Renaker, j. mongo and E. Costello, "Mongo DB Documentation," 2016. [Online]. Available: <https://docs.mongodb.com/manual/sharding/>. [Accessed 22 February 2018].
- [12] R. Dharavath and C. Kumar, "A scalable generic transaction model scenario for distributed," *The Journal of Systems and Software*, p. 43–58, 2014.
- [13] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden, "The Design and Implementation of Modern Column-Oriented Database Systems," *Foundations and Trends in Databases*, vol. 5, p. 197–280, 2013.
- [14] K. Ramasamy, "Quora," 6 August 2014. [Online]. Available: <https://www.quora.com/What-is-difference-in-between-a-keyValue-and-a-columnOriented-database>. [Accessed 22 November 2017].

- [15] DataStax Enterprise, "DataStax," 16 March 2018. [Online]. Available: <https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlTransactionsDiffer.html>. [Accessed 19 March 2018].
- [16] V. Abramova, J. Bernardino and P. Furtado, "Which NoSQL Database? A Performance Overview," *Research Online Publishing*, vol. 1, no. 2, pp. 17-24, 2014.
- [17] A. Goel, "https://medium.com," A Medium Corporation, 29 September 2016. [Online]. Available: <https://medium.com/@amangoeliitb/improving-database-performance-with-redis-dbd38fdf3cb>. [Accessed 16 May 2019].
- [18] G. Garefe, V. Hars, K. Hideaki, H. Kuno, J. Tucek, M. Lillibridge and A. Veitch, "In-Memory Performance for Big Data," *VLDB Endowment*, vol. 8, pp. 37-48, 2014.
- [19] S. Ahirrao and R. Ingle, "Scalable transactions in cloud data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, pp. 1-14, 15 November 2015.
- [20] L. Ashdown and T. Kyte, "https://docs.oracle.com," Oracle Inc, May 2015. [Online]. Available: <https://docs.oracle.com/cloud/latest/db112/CNCPT/schemaob.htm#CNCPT711>. [Accessed 20 February 2018].
- [21] E. Macauley, J. Gnanayutham, R. Byham and C. Guyer, "https://docs.microsoft.com/," Microsoft Inc., 4 October 2017. [Online]. Available: <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-partition-scheme-transact-sql>. [Accessed 20 February 2018].
- [22] D. Dai, W. Zhang and Y. Chen, "IOGP: An Incremental Online Graph Partitioning Algorithm for," *ACM Symposium on High-Performance Parallel and Distributed Computing*, pp. 219-229, 30 June 2017.
- [23] J. Alonso, "Aerospace Design Laboratory,," Stanford.edu, 23 April 2014. [Online]. Available: http://adl.stanford.edu/cme342/Lecture_Notes_files/lecture7-14.pdf. [Accessed 20 February 2018].
- [24] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham and C. Matser, "Performance Evaluation on NoSQL Databases: A Case Study," *Association for Computing Machinery*, pp. 5-10, 2015.
- [25] R. R. Shetty, A. M. Dissanayaka, S. Mengel, L. Gittner, H. Khan and H. Khan, "Secure NoSQL Based Medical Data Processing and Retrieval: The Exposome Project," *A Review of MongoDB and Singularity Container Security in regards to HIPAA Regulations*, pp. 99-105, 2017.
- [26] D. Pasqualin, G. Souza and E. L. Buratti, "A Case Study of the Aggregation Query Model in Read-Mostly NoSQL Document Stores," *Monitoring System of the Ministry of Communication of Brazil*, pp. 1-6, 2014.
- [27] Z. Wei, G. Pierre and C.-H. Chi, "CloudTPS: Scalable Transactions for Web Applications in the Cloud," *IEEE TRANSACTIONS ON SERVICES COMPUTING*, pp. 1-16, 2011.

- [28] P. VENKATESH, "Open Source For You," EFY Enterprise Pvt. Ltd., 30 January 2012. [Online]. Available: <http://opensourceforu.com/2012/01/newsqI-handle-big-data/>. [Accessed 9 April 2018].
- [29] IBM Corporation, "Transaction Processing: Past, Present, Future," pp. 1-64, 2012.
- [30] S. S. Conn, "OLTP and OLAP Data Integration: A Review of Feasible Implementation Methods and Architectures for Real Time Data Analysis," pp. 1-6, 2005.
- [31] R. J. Peris, M. P. Martinez, K. Magoutis, A. Bilas and I. Brondino, "CumuloNimbo: A Highly-Scalable Transaction," *ERCIM News*, pp. 34-35, 2012.
- [32] H. Howard, D. Malkhi and A. Spiegelman, "Flexible Paxos: Quorum Intersection Revisited," *Leibniz International Proceedings in Informatics*, pp. 1-17, 2016.
- [33] F. Gessert, W. Wingerath, S. Friedrich and N. Ritter, "NoSQL Database System: A Survey and Decision Guidance," *Computer Science - Research and Development*, vol. 32, no. 3-4, p. 353–365, 2016.
- [34] M. Cadot, "Oracle FAQ's," 22 December 2012. [Online]. Available: http://www.oraFAQ.com/wiki/Hash_partitioning. [Accessed 28 March 2018].
- [35] Oracle Group, "Oracle Help Center," 2016. [Online]. Available: <https://docs.oracle.com/database/121/VLDBG/GUID-D6CC12F9-81A5-4E89-872F-024C6161C0E8.htm>. [Accessed 28 March 2018].
- [36] Amazon Group, "Amazon DynamoDB," *Developer Guide*, pp. 1-925, 10 August 2012.
- [37] Amazon Group, "AWS Amazon Documentation," Amazon.com, Inc, 10 August 2012. [Online]. Available: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html>. [Accessed 29 March 2018].
- [38] The Apache Software Foundation, "Apache ZooKeeper," The Apache Software Foundation, 2017. [Online]. Available: <https://zookeeper.apache.org/>. [Accessed 2 April 2018].
- [39] J. J. Kim, "Transaction Processing Method for NoSQL Based Column," *Journal of Information Processing Systems*, vol. 13, pp. 1575-1576, 2017.
- [40] D. Bermbach, S. Muller, J. Eberhardt and S. Tai, "Informed Schema Design for Column Store-based Database Services," *Service-Oriented Computing and Applications (SOCA)*, pp. 1-10, 2016.
- [41] R. Mehra, N. Lodhi and R. Babu, "Column Based NoSQL Database, Scope and Future," *International Journal of Research and Analytical Reviews*, vol. 2, no. 4, pp. 105-113, 2015.
- [42] R. Arora and R. R. Aggarwal, "Modeling and Querying Data in MongoDB," *International Journal of*

Scientific & Engineering Research, vol. 4, no. 7, pp. 141-144, 2013.

- [43] M. D. documentation, "Mongo DB," Mongo DB, Inc., 2018. [Online]. Available: <https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/>. [Accessed 8 April 2018].
- [44] T. Kudo, M. Ishino, K. Saotome and N. Kataoka, "A proposal of transaction processing method for MongoDB," *Procedia Computer Science* 96, p. 801 – 810, 2016.
- [45] Quora Authors Group, "Quora," 28 December 2016. [Online]. Available: <https://www.quora.com/How-do-I-develop-an-OLTP-system-using-Hadoop-and-Big-data-technologies>. [Accessed 9 April 2018].
- [46] R. Cattell, "Scalable SQL and NoSQL Data Stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12-27, 2011.
- [47] J. Zollmann, "NoSQL Databases," pp. 1 - 21, 2012.
- [48] P. A. Bungama, "Graph Database Application using Neo4j," *International Journal of Engineering and Technical Research*, vol. 4 International Journal of Engineering and Technical Research, no. Graph Database Application using Neo4j, pp. 999 -1002, 2015.
- [49] S. A. T. Mpinda, L. C. Ferreira, M. X. Ribeiro and M. T. P. Santos, "Evaluation of graph databases," vol. 6, pp. 87 - 98, 2015.
- [50] "https://www.tutorialspoint.com," [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_introduction.htm. [Accessed 27 April 2019].
- [51] K. G. Srinivasa and A. K. Muppalla, "Getting Started with Hadoop," in *Guide to High Performance Distributed Computing*, Springer International Publishing, 2015, pp. 33-71.
- [52] M. Rouse, "https://searchdatamanagement.techtarget.com/," TechTarget, March 2018. [Online]. Available: <https://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator>. [Accessed 27 April 2019].
- [53] H. Team, "https://www.hadoopinrealworld.com," Hadoop In Real World, 12 July 2015. [Online]. Available: <https://www.hadoopinrealworld.com/namenode-and-datanode/>. [Accessed 27 April 2019].
- [54] "https://wiki.apache.org," Hadoop Wiki, 10 September 2011. [Online]. Available: <https://wiki.apache.org/hadoop/NameNode>. [Accessed 27 April 2019].
- [55] MongoDB, "MongoDB Architecture Guide," *MongoDB, Inc*, pp. 1-15, 2015.
- [56] MongoDB, "MongoDB Multi-Data Center," *MongoDB, Inc*, pp. 1-16, 2017.

- [57] K. Chodorow, *Scaling MongoDB*, Sebastopol: O'Reilly Media, 2011.
- [58] M. group, "<https://docs.mongodb.com>," MongoDB, Inc, [Online]. Available: <https://docs.mongodb.com/manual/replication/>. [Accessed 28 March 2019].
- [59] M. documentation, "<https://docs.mongodb.com>," 2018. [Online]. Available: <https://docs.mongodb.com/manual/reference/glossary/>. [Accessed 29 April 2019].
- [60] "<https://www.python.org>," Python Software Foundation , [Online]. Available: <https://www.python.org/doc/essays/blurb/>. [Accessed 29 April 2019].
- [61] A. "<https://www.edureka.co>," Brain4ce Education Solutions Pvt, 19 July 2018. [Online]. Available: <https://www.edureka.co/blog/why-you-should-choose-python-for-big-data>. [Accessed 29 April 2019].
- [62] Collabnix, "<http://collabnix.com>," Collabnix, 2 November 2014. [Online]. Available: <http://collabnix.com/running-hadoop-on-ubuntu-14-04-multi-node-cluster/>. [Accessed 26 March 2019].
- [63] A. Verma, "<https://www.whizlabs.com>," Whizlabs, 3 September 2018. [Online]. Available: <https://www.whizlabs.com>. [Accessed 26 March 2019].
- [64] A. Fox , "<https://www.maketecheasier.com>," 23 January 2018 . [Online]. Available: <https://www.maketecheasier.com/what-is-bashrc/>. [Accessed 27 March 2019].
- [65] H. Team, "<https://www.hadoopinrealworld.com>," 12 July 2015. [Online]. Available: <https://www.hadoopinrealworld.com/namenode-and-datanode/>. [Accessed 27 March 2019].
- [66] "<https://www.edureka.co>," 26 February 2019. [Online]. Available: <https://www.edureka.co/blog/explaining-hadoop-configuration/>. [Accessed 27 March 2019].
- [67] . X. Chen, "<https://blog.cloudera.com>," Cloudera, 20 December 2016. [Online]. Available: <https://blog.cloudera.com/blog/2016/12/hdfs-datanode-scanners-and-disk-checker-explained/>. [Accessed 28 March 2019].

Shtojca. PROCESI DOKUMENTUES PËR INSTALIMIN E PLATFORMËS HADOOP

Hadoop mundëson përpunimin e shpërndarë të grupeve të mëdha të të dhënave përmes disa koleksioneve të kompjuterëve (nyjeve) të cilat kanë karakteristika të ngjajshme. Duke u bazuar në temën time, në vazhdim është paraqitur pjesa dokumentuese e instalimit dhe konfigurimit të platformës Hadoop në një rrethinë të shpërndarë.

Eksperimenti në fjalë është kryer në një laptop me procesor core i7 – 7500U, me 16GB RAM, që përmban hard drive prej rreth 1 TB (ang. Terabyte). Detyra praktika përmban simulimin e 4 paketave të sistemit operativ Linux Ubuntu 14.04.5 LTS duke përdorur paketën softuerike për virtualizim si “VirtualBox”. Të katërt paketat virtuale ndajnë një rrjet të përbashket për të komunikuar së bashku.

1) Si hap i parë është bërë instalimi i sistemit operativ Linux Ubuntu 14.04.5 LTS - Desktop Image 64-bit në secilën nga simulimet virtuale brenda mjetit të virtualizimit -VirtualBox

- Makina e parë virtuale – OS_test1 - (user: test1) me Ubuntu Desktop 14.04, i cili përmban procesor: CPU Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz (4 CPUs), me RAM 4.00 GB dhe 70 GB hard disk. I vendosur si Server nyje për Hadoop

Nyjet tjera punuese (ang. data nodes) në vazhdim:

- Makina e dytë virtuale - OS_test2 - (user: test2) me Ubuntu Desktop 14.04, i cili përmban procesor: CPU Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz (4 CPUs), me RAM 2.00 GB dhe 50 GB hard disk.

Sikur nyja e dytë e paraqitur më lart, të gjitha nyjet e tjera (ang. slaves) përmbajnë të njejtin konfigurim.

Krijimi i katër makinave virtuale është paraqitur më poshtë:

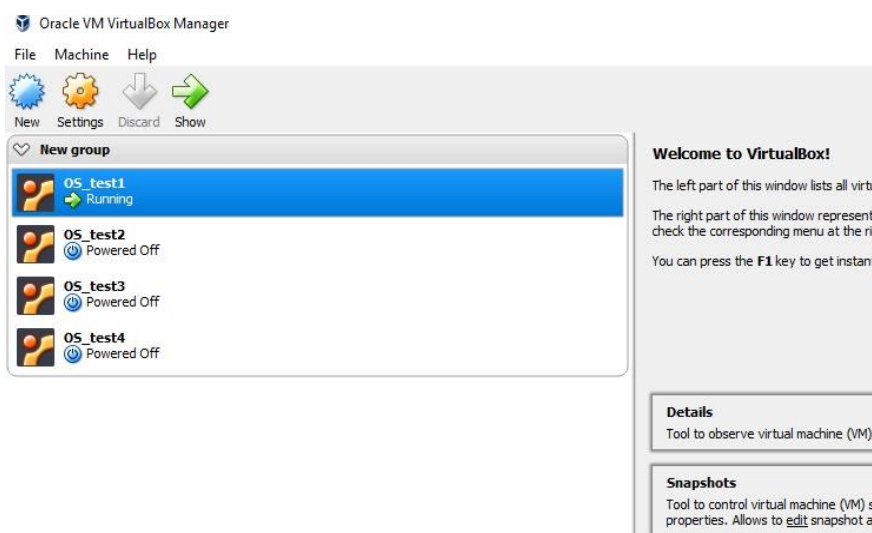


Fig 21. Katër makinat virtuale me sistem operativ Ubuntu 14.04 brenda VirtualBox

Për të siguruar komunikimin brenda nyjeve të instaluara duhet konfiguruar një rrjet i përbashkët brenda paketës VirtualBox, që i mundëson platformës Hadoop të krijojë një koleksion të shpërndarë të nyjeve punuese. Një nga llojet e rrjetit për komunikim është caktuar të jetë: **“Host-only Adapter”**:

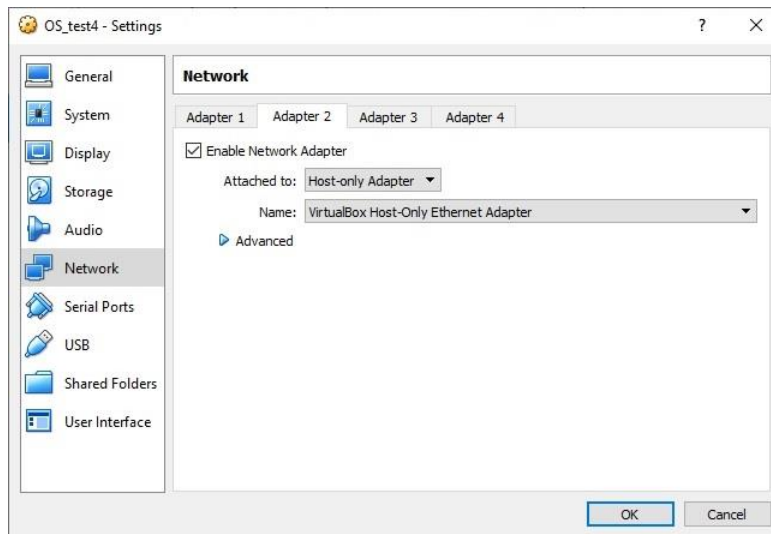


Fig 22. Rregullimi i rrjetit brenda makinës virtuale

Gjithashtu, secila nyje është konfiguruar për të pasur qasje të drejtpërdrejt në internet duke përdorur llojin e rrjetit “NAT”.

Pas instalimi të makinave virtuale, kemi filluar procesin e konfigurimit të mjedisit për instalimin e platformës Hadoop.

- a. Si pikë e parë është bërë shtimi i adresave IP për secilen nyje (hostname) për të qenë në gjendje që të komunikojnë ndërmjet veti. Ndryshimi është bërë brenda fajllit të konfigurimit të hostname dhe adresave IP si /etc/hosts me të dhënat në vazhdim:

Shtimi i këtyre linjave në të gjitha nyjet e klasterit:

```
#master
127.0.1.1 test1-VirtualBox

#secondary name node
127.0.1.2 test2-VirtualBox

#slave 1
127.0.1.3 test3-VirtualBox

#slave 2
127.0.1.4 test4-VirtualBox
```

- b) Përmes këtyre adresave IP, kemi aktivizuar një protokoll të internetit (ang. network protocol) i cili mundëson komunikimin e nyjës master me të gjitha nyjet që veprojnë në klaster. Ky protokoll i cili ofron komunikim të sigurt quhet “SSH”

nga i cili kemi gjeneruar një çelës privat dhe publik ku çelësi publik është shpërndarë në nyjet e tjera operuese për të mundësuar qasje.

Kodi për gjenerimin e një SSH RSA key-pair:

```
test1@test1-VirtualBox: ~
test1@test1-VirtualBox:~$ cd /etc/
test1@test1-VirtualBox:/etc$ sudo gedit hosts
test1@test1-VirtualBox:/etc$ cd ~
test1@test1-VirtualBox:~$ bash
test1@test1-VirtualBox:~$ ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/home/test1/.ssh'.
Your identification has been saved in /home/test1/.ssh/id_rsa.
Your public key has been saved in /home/test1/.ssh/id_rsa.pub.
The key fingerprint is:
f4:83:76:2c:ae:22:92:64:12:30:f8:d6:b0:ed:2b:0b test1@test1-VirtualBox
The key's randomart image is:
+--[ RSA 2048 ]-----+
|
|+
|.o =
|. + o . +
| o . S =
|.o . o o .
|E. . .
|o.o o .
|..+ ..
+-----+
test1@test1-VirtualBox:~$
```

Fig 23. Gjenerimi i SSH çelësit (ang. ssh key)

Për të mundësuar komunikimin e njëjës kryesore (serverit) me nyjet e tjera brenda klasterit, duhet gjeneruar një kopje e çelësit publik për të gjitha këto nyje dytësore (ang. slave). Ky do të jetë një hap parësor dhe shumë i rëndësishëm në formimin e klasterit dhe ngritjen e platformës Hadoop në një rrethinë të shpërndarë. Në vazhdim do të paraqesim kopjimin e çelësit publik në një nga nyjet dytësore:

```
test1@test1-VirtualBox: ~
test1@test1-VirtualBox:~$ ssh-copy-id -i /home/test1/.ssh/id_rsa.pub test3@192.168.56.102
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
test3@192.168.56.102's password:

Number of key(s) added: 1

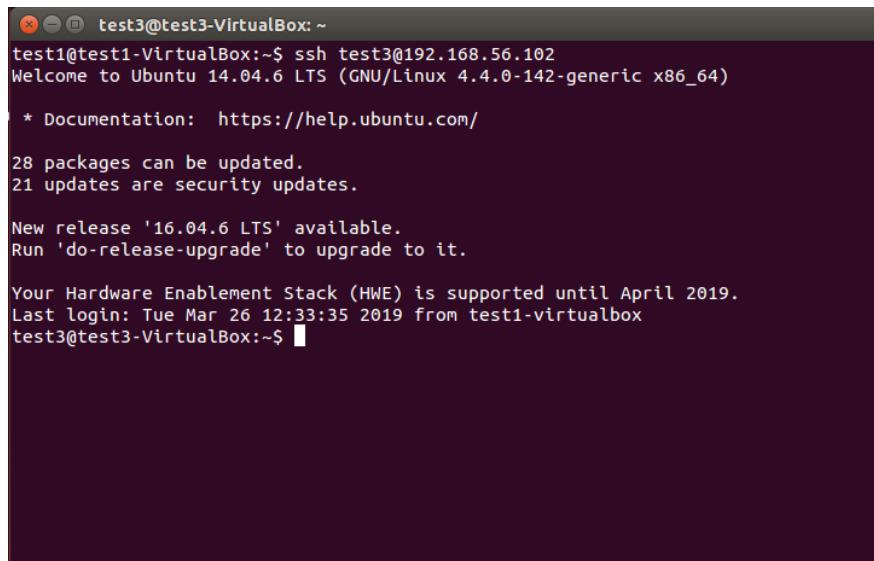
Now try logging into the machine, with: "ssh 'test3@192.168.56.102'"
and check to make sure that only the key(s) you wanted were added.

test1@test1-VirtualBox:~$
```

Fig 24. Kopjimi i çelësit publik në njën përkatëse

Për të mundësuar komunikimin brenda klasterit, një kopjim i tillë i çelësit publik për “SSH” është bërë në të gjitha nyjet (serverët) përkatës. Çelësi publik i kopjuar nga një Master do të ruhet në një fajll të quajtur ‘authorized_keys’ në të gjitha nyjet tjera (ang. slave) që mundëson qasjen direkte në këto nyje, pa nevojën për ndonjë password eventual

në nyjet përkatëse. Një konfigurim i tillë i çelësit publik mundëson një qasje të përhershme në nyjet e tjera për të zhvilluar një komunikim të duhur pa pasur nevojën e konfigurimeve shtesë në të ardhmen. Figura në vazhdim pasqyron një rrethanë të tillë, e cila paraqet qasjen në nyjën tjetër përkatëse nga Master serveri:



```
test3@test3-VirtualBox: ~
test1@test1-VirtualBox:~$ ssh test3@192.168.56.102
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 4.4.0-142-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

28 packages can be updated.
21 updates are security updates.

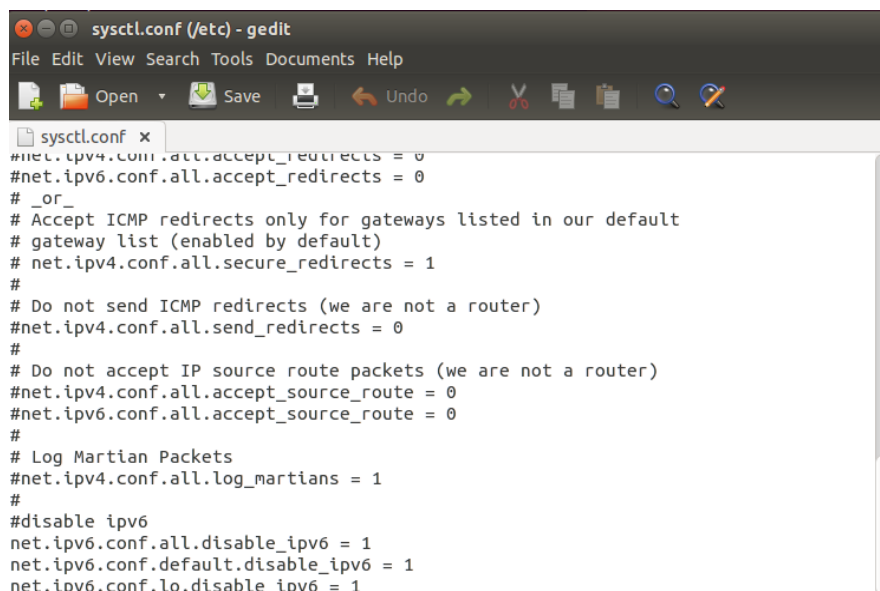
New release '16.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2019.
Last login: Tue Mar 26 12:33:35 2019 from test1-virtualbox
test3@test3-VirtualBox:~$
```

Fig 25. Qasja nga Master në nyjën e njohur si 'test3'

Platforma Hadoop ka qenë zhvilluar dhe testuar që të punoj me protokolin e rrjetit IPv4, ku vetëm serverët që kanë të aktivizuar protokolin IPv4 mund të komunikojnë brenda klasterit. Shpesh disa gabime gjatë komunikimit të nyjeve brenda platformës Hadoop janë paraqitur përshkak se disa versione të Linux përdorin në mënyrë të parazgjedhur (ang. by default) IPv6.

Pasi platforma Hadoop nuk mbështet qasjen ndërmjet nyjeve duke përdorur protokolin IPv6 të rrjetit, është e rekomandueshme për të bërë konfigurimin jo aktive për protokolin IPv6 . Deaktivizimi për protokolin IPv6 do të paraqitet si në vijim [62]:



```
sysctl.conf (/etc) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo
sysctl.conf x
#net.ipv4.conf.all.accept_redirects = 0
#net.ipv6.conf.all.accept_redirects = 0
#_or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#
#disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Fig 26. Deaktivizimi i IPv6 protokolit

Një nga komponentët për procesimin e të dhënave si MapReduce është një framework (strukturë) e programimit i bazuar në Java platformën [63]. Në vazhdim (brenda Master nyjës) të ndërtimit të klasterit Hadoop është shumë e domosdoshme që të konfigurojmë platformën Java, e cila mundëson funksionalitetin e komponentëve brenda platformës Hadoop. Platforma Java e instaluar është me version '11.0.2', siç shihet në figurën e mëposhtme:

```
test1@test1-VirtualBox:~$ java -version
java version "11.0.2" 2019-01-15 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.2+9-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.2+9-LTS, mixed mode)
test1@test1-VirtualBox:~$
```

Fig 27. Java platforma e instaluar

Pas instalimit të platformës Java, duhet bërë konfigurimi i variablave për Java në mënyrë që sistemi operativ të dijë për vendndodhjen e tij, si dhe të jetë në dispozicion për përdorim, ku janë vendosur komandat si më poshtë brenda terminalit:

```
test1@test1-VirtualBox: ~
test1@test1-VirtualBox:~$ sudo update-alternatives --install "/usr/bin/javac" "javac" "/usr/local/java/jdk-11.0.2/bin/javac" 1
test1@test1-VirtualBox:~$ sudo update-alternatives --set javac /usr/local/java/jdk-11.0.2/bin/javac
test1@test1-VirtualBox:~$
```

Fig 28. Konfigurimi i variablave për Java

Pasi është bërë instalimi dhe konfigurimi i platformës Java për një Master, është vazhduar me të njëjtën procedurë edhe me tri nyjet tjera punuese. Në të gjitha nyjet e tjera kemi instaluar versionin e njëjtë pra 'Java 11.0.2' si dhe kemi bërë konfigurimin e variablave mjedisore për të mundësuar sistemit që të komunikoj me të. Demonstrimi i instalimit dhe konfigurimit të Java-s në një nga nyjet të quajtur 'test4' është paraqitur si në vijim:

```
test4@test4-VirtualBox: ~
test4@test4-VirtualBox:~$ java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
test4@test4-VirtualBox:~$ echo $JAVA_HOME
/usr/local/java/jdk-11.0.2
test4@test4-VirtualBox:~$
```

Fig 29. Instalimi dhe konfigurimi i Java platformës në një 'test4'

Pas gjithë këtyre hapave në ngritjen e klasterit Hadoop, hapi i ardhshëm ka të bëjë specifikisht me aspektin e instalimit dhe konfigurimit të tij. Para se gjithash kemi shkarkuar dhe instaluar Hadoop-në në një Master dhe në të gjitha nyjet e tjera punuese. Platforma Hadoop të cilën e kemi implementuar përmban të dhënat si në vazhdim:

Në fillim kemi shkarkuar platformën Hadoop duke e vendosur në një lokacion specifik (brenda folderit 'hadoop') të shfaqur si më poshtë:

```
test1@test1-VirtualBox: ~/Downloads
test1@test1-VirtualBox:~$ cd Downloads/
test1@test1-VirtualBox:~/Downloads$ sudo tar -xvzf hadoop-2.6.5.tar.gz -C /usr/local
```

Fig 30. Instalimi i platformës Hadoop

Pasi kemi shkarkuar Hadoop-në, në mënyrë që të kemi qasje dhe të ekzekutojmë fajllat brenda folderit Hadoop nga përdoruesi i caktuar (user:test1), duhet që t'i japim autorizimin përkatës përmes komandës:

```
test1@test1-VirtualBox: ~/Downloads
test1@test1-VirtualBox:~$ cd Downloads/
test1@test1-VirtualBox:~/Downloads$ sudo chown -R test1 /usr/local/hadoop
```

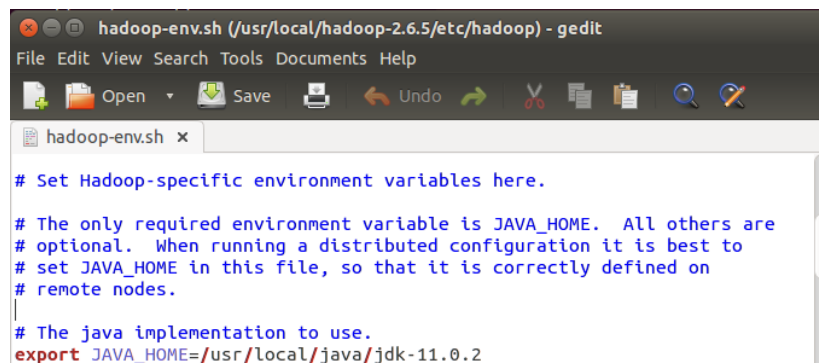
Fig 31. Autorizimi i përdoruesit 'test1' për të pasur qasje brenda fajllave Hadoop

Pasi kemi instaluar Hadoop-në dhe kemi vendosur autorizimet për përdorimin e tij, në vazhdim duhet të vendosim variablat mjedisore për Hadoop-në si dhe Java-n brenda fajllit `'.bashrc'` për të mundësuar ekzekutimin e këtyre variablave në mënyrë automatike. `'.bashrc'` është një *shell* skriptë brenda sistemit operativ Linux, i vendosur brenda folderit apo drejtorisë (ang. *directry*) kryesore si `'home'`. Qëllimi i `.bashrc` shell skriptës është që të ekzekutojë komandat e vendosura brenda tij në mënyrë interaktive, sa herë që ato thirren brenda terminalit. Komandat e vendosura brenda `'.bashrc'` kanë një funksion të përhershëm, të cilat nuk kemi nevojë t'i shkruajmë sa herë që na duhen në terminal. Prandaj, që të kemi në dispozicion të gjitha komponentët brenda Hadoop-së dhe Java-s më poshtë janë paraqitur komandat përkatëse të vendosura në `'.bashrc'` shell skriptën [64].

```
.bashrc (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
.bashrc x
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export JAVA_HOME=/usr/local/java/jdk-11.0.2
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
Plain Text Tab Width: 8 Ln 1, Col 1 INS
```

Fig 32. Konfigurimi i variablave mjedisore për Hadoop dhe Java

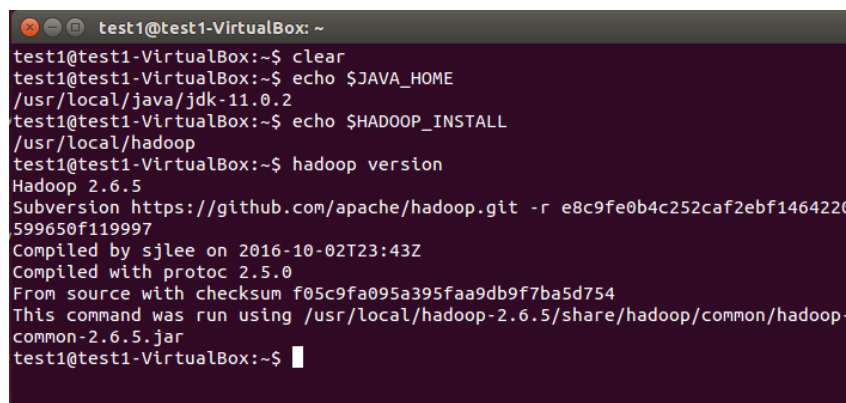
Në mënyrë që Hadoop-ja të dijë se ku është instaluar Java, duhet bërë disa ndryshime brenda Hadoop-së shell skriptës të quajtur *'hadoop-env.sh'*. Kodi në vazhdim paraqet vendsojen e lokacionit për folderin Java të njohur si JAVA_HOME.



```
hadoop-env.sh (/usr/local/hadoop-2.6.5/etc/hadoop) - gedit
File Edit View Search Tools Documents Help
hadoop-env.sh x
# Set Hadoop-specific environment variables here.
# The only required environment variable is JAVA_HOME. All others are
# optional. When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.
# The java implementation to use.
export JAVA_HOME=/usr/local/java/jdk-11.0.2
```

Fig 33. Vendosja e lokacionit për Java të instaluar brenda skriptës Hadoop

Pas vendosjes dhe konfigurimit të variablave për Hadoop-në si dhe të tjera, do të konkludohet për instalimin e platformës Hadoop në një Master si dhe në të gjitha nyjet e tjera punuese. Figura si më poshtë paraqet instalimin e suksesshëm të klasterit Hadoop në një Master, që është e gatshme për vendosjen e klasterit.



```
test1@test1-VirtualBox: ~
test1@test1-VirtualBox:~$ clear
test1@test1-VirtualBox:~$ echo $JAVA_HOME
/usr/local/java/jdk-11.0.2
test1@test1-VirtualBox:~$ echo $HADOOP_INSTALL
/usr/local/hadoop
test1@test1-VirtualBox:~$ hadoop version
Hadoop 2.6.5
Subversion https://github.com/apache/hadoop.git -r e8c9fe0b4c252caf2ebf1464220
599650f119997
Compiled by sjlee on 2016-10-02T23:43Z
Compiled with protoc 2.5.0
From source with checksum f05c9fa095a395faa9db9f7ba5d754
This command was run using /usr/local/hadoop-2.6.5/share/hadoop/common/hadoop-
common-2.6.5.jar
test1@test1-VirtualBox:~$
```

Fig 34. Konfigurimi dhe versioni i Hadoop-ve të instaluar

Në bazë të fotos që shohim më sipër janë konfiguruar edhe nyjet e tjera punuese të cilat kanë të njëjtin version të Hadoop-ve të instaluar si dhe të njëjtin konfigurim të tij. Pas këtij hapi të rëndësishëm, do të bëhet konfigurimi i Master nyjës si dhe nyjeve tjera punuese për vendosjen e klasterit Hadoop në një rrethinë të shpërndarë. Master-i si server kryesor si dhe nyjet e tjera mbartin disa konfigurime brenda moduleve të tyre për të mundësuar krijimin e një rrethine të shpërndarë. Për një Master kryesor duhet krijuar një folder i quajtur *'NameNode'* i cili ruan metadata (të dhënat që përshkruajnë disa të dhëna) të sistemit të file-ve për Hadoop (ang. HDFS). Ndërsa për nyjet e tjera punuese (ang. slave) duhet krijuar një folder i quajtur *'DataNode'*. DataNode ruan të dhënat aktuale të instraktuara nga NameNode në sistemin e file-ve të Hadoop-së (ang. HDFS). Për të pasur një klaster funksional, NameNode Master duhet të ketë komunikim të vazhdueshëm me nyjet e tjera DataNode [65]. Për të menaxhuar NameNode dhe DataNode të dhënat në një Master, duhet bërë konfigurimi i *hdfs-site.xml* file-it si shfaqur më poshtë:

```

<configuration>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///home/test1/hdfs/datanode</value>
  <description>DataNode directory</description>
</property>

  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///home/test1/hdfs/namenode</value>
    <description>NameNode directory for namespace and transaction
logs storage.</description>
  </property>

  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>

  <property>
    <name>dfs.permissions</name>
    <value>>false</value>
  </property>

  <property>
    <name>dfs.datanode.use.datanode.hostname</name>
    <value>>false</value>
  </property>

  <property>
    <name>dfs.namenode.datanode.registration.ip-hostname-check</
name>
    <value>>false</value>
  </property>
</configuration>

```

Fig 35. Kofigurimi i *hdfs-site.xml* për sistemin e file-ve të Hadoop-it (ang. HDFS)

Hapi i ardhshëm ka të bëjë me konfigurimin e file-it *core-site.xml* i cili shfaq të gjitha konfigurimet të cilat kanë të bëjnë me sistemin e file-ve të Hadoop-it (ang. HDFS). Ky file përmban adresën IP për Master njëën si dhe portin e tij, gjithashtu mund të përmbajë shënime tjera përshkruese.

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

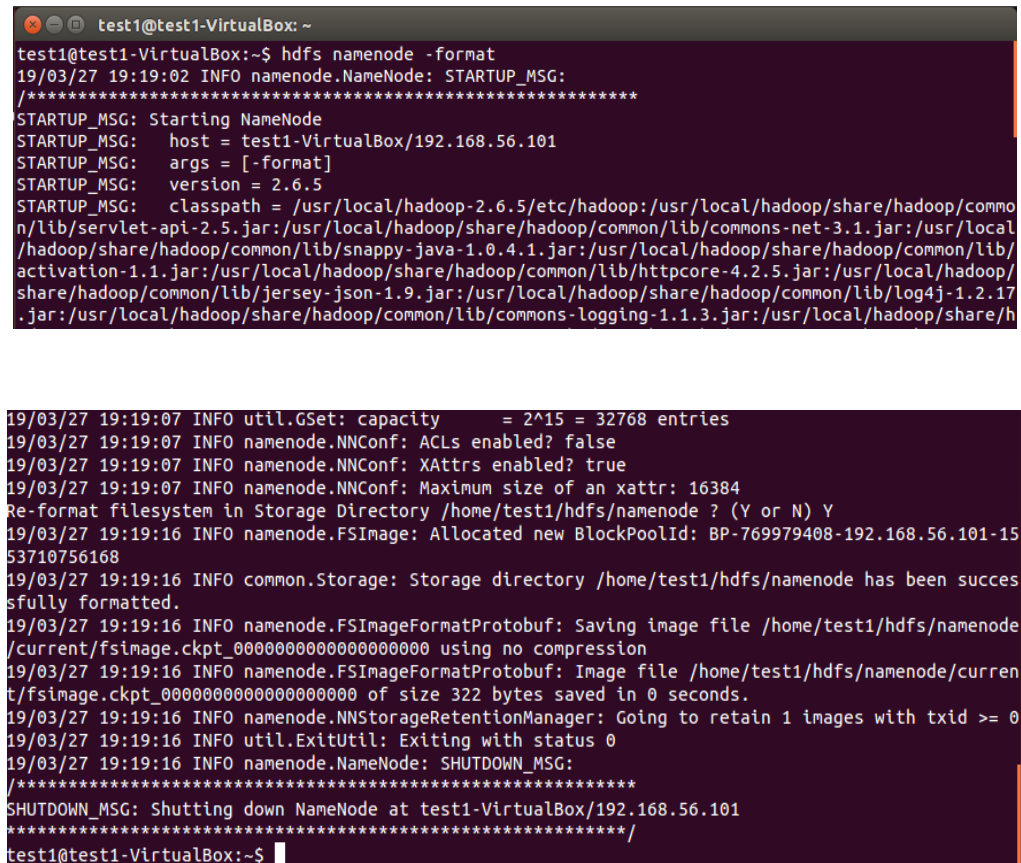
<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://192.168.56.101</value>
  <description>NameNode URI</description>
</property>
</configuration>

```

Fig 36. Konfigurimi për file-in *core-site.xml*

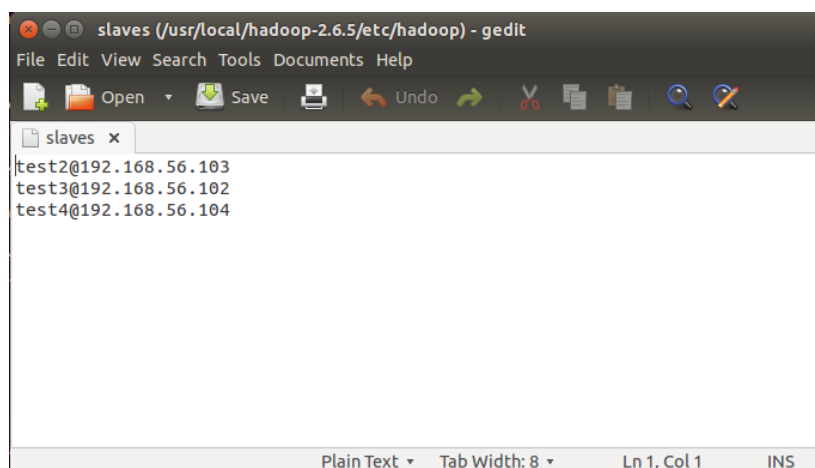
Në mënyrë që të inicializohen të dhënat e konfiguruar brenda file-it `hdfs-site.xml`, duhet të formatohet NameNode i cili ka për detyrë të ruajë metadata për DataNode-n [66].



```
test1@test1-VirtualBox: ~
test1@test1-VirtualBox:~$ hdfs namenode -format
19/03/27 19:19:02 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = test1-VirtualBox/192.168.56.101
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.6.5
STARTUP_MSG: classpath = /usr/local/hadoop-2.6.5/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/servlet-api-2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-net-3.1.jar:/usr/local/hadoop/share/hadoop/common/lib/snappy-java-1.0.4.1.jar:/usr/local/hadoop/share/hadoop/common/lib/activation-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-json-1.9.jar:/usr/local/hadoop/share/hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-logging-1.1.3.jar:/usr/local/hadoop/share/h
19/03/27 19:19:07 INFO util.GSet: capacity = 2^15 = 32768 entries
19/03/27 19:19:07 INFO namenode.NNConf: ACLs enabled? false
19/03/27 19:19:07 INFO namenode.NNConf: XAttrs enabled? true
19/03/27 19:19:07 INFO namenode.NNConf: Maximum size of an xattr: 16384
Re-format filesystem in Storage Directory /home/test1/hdfs/namenode ? (Y or N) Y
19/03/27 19:19:16 INFO namenode.FSImage: Allocated new BlockPoolId: BP-769979408-192.168.56.101-1553710756168
19/03/27 19:19:16 INFO common.Storage: Storage directory /home/test1/hdfs/namenode has been successfully formatted.
19/03/27 19:19:16 INFO namenode.FSImageFormatProtobuf: Saving image file /home/test1/hdfs/namenode/current/fsimage.ckpt_00000000000000000000 using no compression
19/03/27 19:19:16 INFO namenode.FSImageFormatProtobuf: Image file /home/test1/hdfs/namenode/current/fsimage.ckpt_00000000000000000000 of size 322 bytes saved in 0 seconds.
19/03/27 19:19:16 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
19/03/27 19:19:16 INFO util.ExitUtil: Exiting with status 0
19/03/27 19:19:16 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at test1-VirtualBox/192.168.56.101
*****/
test1@test1-VirtualBox:~$
```

Fig 37. Formatimi i NameNode sistemit të file-ve Hadoop (ang. HDFS)

Për të mundësuar komunikimin dhe funksionimin e shëbimeve të nyjeve punuese (ang. slave) duhet konfiguruar file 'slaves' i cili gjendet brenda Hadoop-së në një Master, si vijon:



```
slaves (/usr/local/hadoop-2.6.5/etc/hadoop) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
slaves x
test2@192.168.56.103
test3@192.168.56.102
test4@192.168.56.104
Plain Text Tab Width: 8 Ln 1, Col 1 INS
```

Fig 38. Shënimet për nyjet punuese (ang. slave) në Master

Shënimet për DataNode-n apo nyjet punuese (ang. slave) përmbajnë emrin e përdoruesëve dhe adresën IP të serverëve të cilëve do t'u qasemi. Këto shënime të dhënave

në këtë formë janë shumë të rëndësishme kur të lëshojmë skriptat të cilat thërrasin shërbimet e tyre (në nyjet përkatëse).

Siç cekëm edhe më lart se DataNode e cila gjendet në nyjet punuese (ang. slave) ruan të dhënat në sistemin e file-ve të Hadoop-së (ang. HDFS). Sikur në një Master, për të mundësuar ruajtjen e të dhënave duhet konfiguruar hdfs-site.xml file i cili gjendet brenda Hadoop-së në secilën nga nyjet punuese (ang. slave). Në vazhdim është paraqitur konfigurimi i file në fjalë në një punuese të quajtur 'test2'.

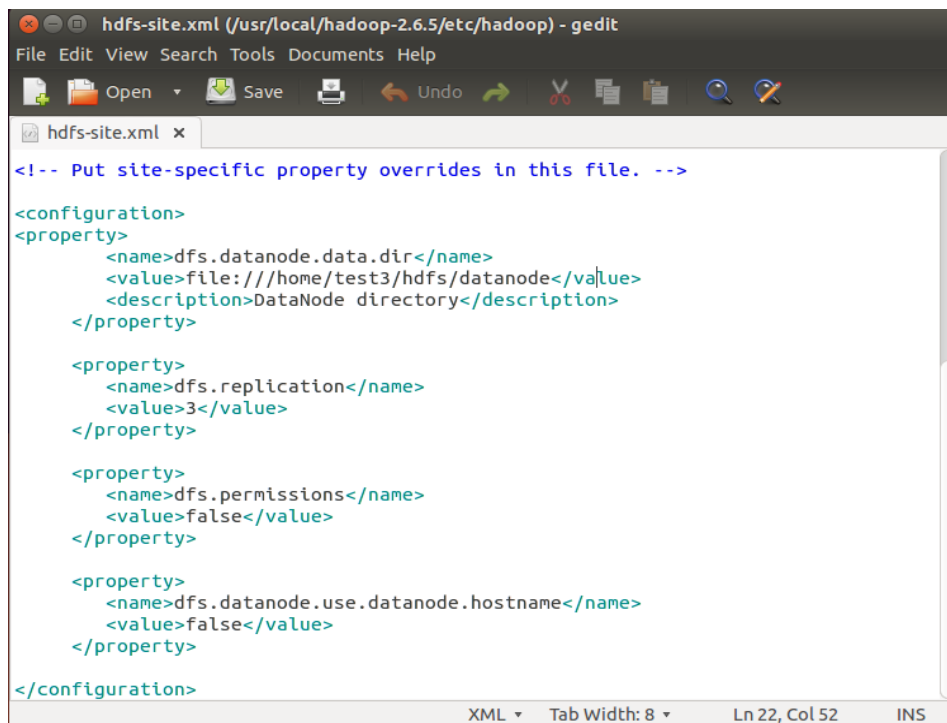


Fig 39. Konfigurimi i hdfs-site.xml për sistemin e file-ve të Hadoop -së (ang. HDFS) për një punuese (test3)

Gjithashtu, të gjitha nyjet punuese (ang. slave) duhet të dijnë për vendndodhjen e nyjes kryesore Master, duke konfiguruar core-site.xml file me të njejtat të dhëna sikur në një Master.

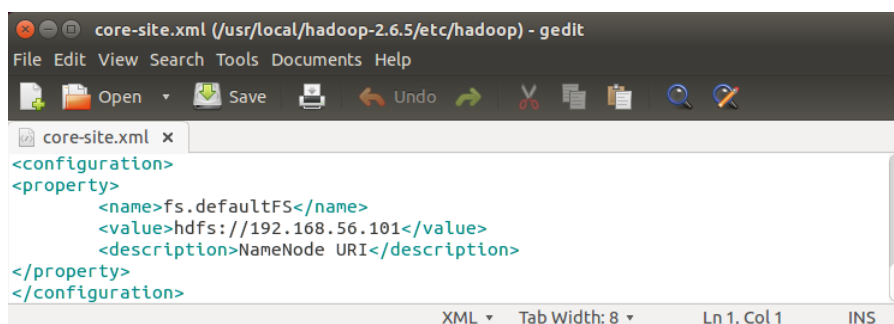


Fig 40. Konfigurimi për file-in core-site.xml në një punuese 'test3'

Pas kryerjes të të gjitha konfigurimeve për Hadoop-në, në fund do të shohim ndërtimin e suksesshëm të të gjitha komponenteve dhe nyjeve që përbëjnë këtë klaster. Do të testohen të gjitha nyjet brenda klasterit për të parë funksionalitetin e tyre.

Për të ngritur klasterin së pari duhet të lëshojmë të gjitha komponentët e tij përmes komandës:

\$ Start-all.sh

```
test1@test1-VirtualBox: ~  
test1@test1-VirtualBox:~$ start-all.sh  
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh  
WARNING: An illegal reflective access operation has occurred  
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/usr/local/hadoop-2.6.5/share/hadoop/common/lib/hadoop-auth-2.6.5.jar) to method sun.security.krb5.Config.getInstance()  
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil  
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations  
WARNING: All illegal access operations will be denied in a future release  
Starting namenodes on [test1-VirtualBox]  
test1-VirtualBox: starting namenode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-test1-namenode-test1-VirtualBox.out  
test4@192.168.56.104: starting datanode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-test4-datanode-test4-VirtualBox.out  
test3@192.168.56.102: starting datanode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-test3-datanode-test3-VirtualBox.out  
test2@192.168.56.103: starting datanode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-test2-datanode-test2-VirtualBox.out  
Starting secondary namenodes [0.0.0.0]  
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-test1-secondarynamenode-test1-VirtualBox.out  
WARNING: An illegal reflective access operation has occurred  
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/usr/local/hadoop-2.6.5/share/hadoop/common/lib/hadoop-auth-2.6.5.jar) to method sun.security.krb5.Config.getInstance()  
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil  
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations  
WARNING: All illegal access operations will be denied in a future release  
Starting yarn daemons  
Starting resourcemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-test1-resourcemanager-test1-VirtualBox.out  
test3@192.168.56.102: starting nodemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-test3-nodemanager-test3-VirtualBox.out  
test4@192.168.56.104: starting nodemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-test4-nodemanager-test4-VirtualBox.out  
test2@192.168.56.103: starting nodemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-test2-nodemanager-test2-VirtualBox.out  
test1@test1-VirtualBox:~$
```

Fig 41. Lëshimi i shell skriptës kryesor për ndërtimin e klasterit Hadoop

Siç shihet më lart, të gjitha nyjet që veprojnë brenda klasterit janë thirrur përmes protokollit të komunikimit 'SSH'. Në fillim janë thirrur të gjitha komponentët kryesorë të Hadoop-së si sistemi i file-ve të shpërndarë të Hadoop (ang. HDFS) dhe YARN. Pastaj të gjitha këto komponente janë në dispozicion për të gjitha nyjet brenda klasterit Hadoop, i cili funksionon në një rrethinë të shpërndarë.

Pasi skriptat për të ngritur një klaster Hadoop janë lëshuar, verifikojmë se të gjitha nyjet dhe proceset brenda tij janë duke funksionuar. Ky testim është bërë duke përdorur komandën 'jps'.

```
test1@test1-VirtualBox: ~  
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.rity.authentication.util.KerberosUtil  
WARNING: Use --illegal-access=warn to enable warnings of further illegal refl access operations  
WARNING: All illegal access operations will be denied in a future release  
starting yarn daemons  
starting resourcemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-test1-emanager-test1-VirtualBox.out  
test3@192.168.56.102: starting nodemanager, logging to /usr/local/hadoop-2.6. yarn-test3-nodemanager-test3-VirtualBox.out  
test4@192.168.56.104: starting nodemanager, logging to /usr/local/hadoop-2.6. yarn-test4-nodemanager-test4-VirtualBox.out  
test2@192.168.56.103: starting nodemanager, logging to /usr/local/hadoop-2.6. yarn-test2-nodemanager-test2-VirtualBox.out  
test1@test1-VirtualBox:~$ jps  
4038 Jps  
3510 NameNode  
3721 SecondaryNameNode  
3868 ResourceManager  
test1@test1-VirtualBox:~$
```

Fig 42. Testimi i të gjitha shërbimeve të lëshuara përmes 'jps' komandës

Rezultatet e shfaqura në foton mësipërme tregojnë një Master me të gjitha proceset tjera janë lëshuar me sukses. Verifikimet e tilla janë bërë në të gjitha nyjet punuese. Në vazhdim do të paraqesim lëshimin e proceseve brenda njëjës punuese të quajtur test2’

```
test2@test2-VirtualBox: ~
test2@test2-VirtualBox:~$ jps
3074 Jps
2771 DataNode
2886 NodeManager
test2@test2-VirtualBox:~$
```

Fig 43. Testimi i të gjitha shërbimeve të lëshuara përmes ‘jps’ komandës në njëjën ‘test2’

Përmbledhja e ndërtimit të klaterit Hadoop, që paraqet të gjitha detajet për sistemin e file-ve të tij (ang. HDFS) si dhe të dhënat për të gjitha nyjet të cilat veprojnë brenda klasterit, mund të shihen nën adresën (ang. URL) në vazhdim:

<http://192.168.56.101:50070> – që përbën hostname dhe port-in e njëjës kryesore Master nën të cilën veprojnë të gjitha nyjet tjera (ang. slave):

Overview 'test1-VirtualBox:8020' (active)

Started:	Thu Feb 28 14:43:09 CET 2019
Version:	2.6.5, re8c9fe0b4c252caf2ebf1464220599650f119997
Compiled:	2016-10-02T23:43Z by sjlee from branch-2.6.5
Cluster ID:	CID-7f8f9c6a-4683-4203-9549-ccd58583146e
Block Pool ID:	BP-1514369906-192.168.56.101-1551361373718

Summary

Security is off.
Safemode is off.
1 files and directories, 0 blocks = 1 total filesystem object(s).
Heap Memory used 39.14 MB of 60.06 MB Heap Memory. Max Heap Memory is 966.69 MB.
Non Heap Memory used 42.94 MB of 47.17 MB Committed Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	141.5 GB
-----------------------------	----------

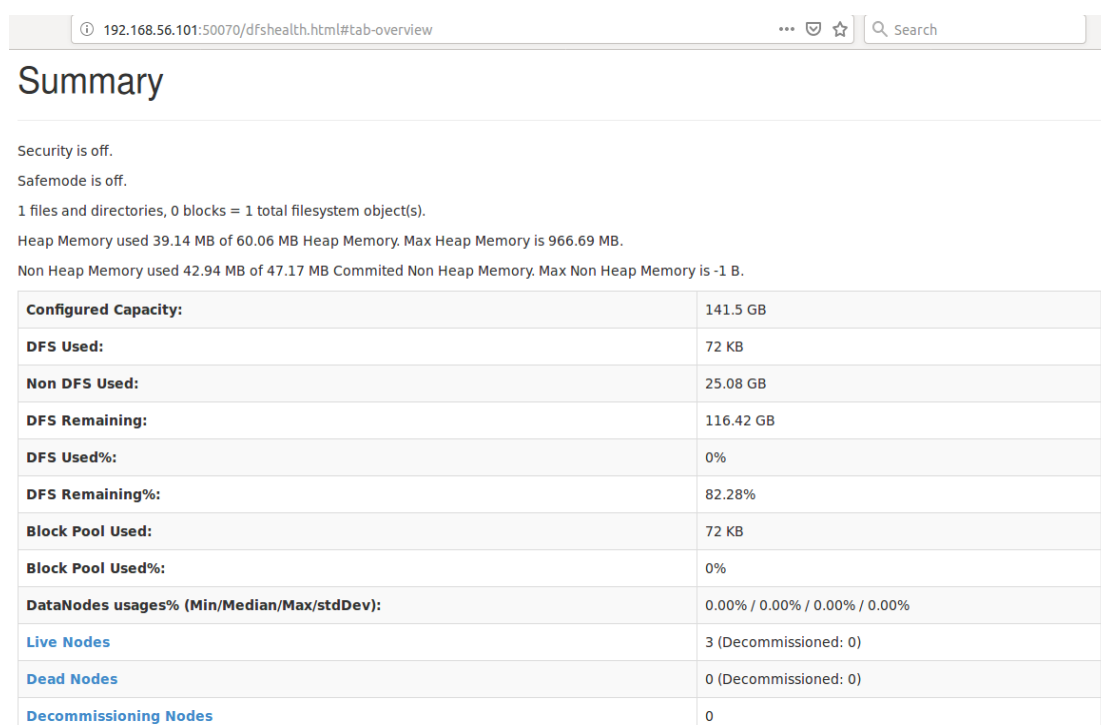
Fig 44. Përmbledhje për të dhënat për klasterin Hadoop e ngritur

Në pamjen e parë të paraqitur më sipër mund të shohim disa detaje për Hadoop klasterin. Brenda ueb-formës së paraqitur për Hadoop mund të shohim një menu e cila përmban disa faqe për klasterin.

Këto përbëjnë një pasqyrim të përgjithshëm për Hadoop, për të dhënat për nyjet punese (ang. DataNode), të dhënat për file përkatës në Hadoop, historikun e qasjes brenda Hadoop (ang. logs) dhe të tjera.

Te faqja e pëmbledhjës (ang. Overview) së të dhënave për Hadoop shihen të dhënat për startimin të tij, versionin për Hadoop i cili është përdor, ID e klasterit si dhe të dhënat e tjera.

Në vazhdim është paraqitur pjesa e përmbledhëse e sistemit të file-ve për Hadoop (ang. HDFS):



The screenshot shows the 'Summary' section of the Hadoop DFS Health Overview page. It includes status information like 'Security is off' and 'Safemode is off', followed by file and memory statistics. A table provides detailed metrics on capacity, usage, and node health.

Configured Capacity:	141.5 GB
DFS Used:	72 KB
Non DFS Used:	25.08 GB
DFS Remaining:	116.42 GB
DFS Used%:	0%
DFS Remaining%:	82.28%
Block Pool Used:	72 KB
Block Pool Used%:	0%
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	3 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0

Fig 45. Të dhënat për sistemin e file-ve të Hadoop-së dhe nyjet punuese

Te pjesa e mësipërme paraqiten të dhënat për konfigurimin e kapacitetit për ruajtjen e të dhënave brenda sistemit të Hadoop-së. Gjithashtu, në vijim të përmbledhjes për Hadoop-në paraqiten të dhënat për nyjet të cilat janë aktive te linja 'Live Nodes'. Në rastin tonë paraqiten gjithësej tri nyje aktive të cilat paraqesin të gjitha nyjet e konfiguruar brenda klasterit Hadoop.

Për të parë detajet rreth nyjeve punuese (ang. slave) klikojmë te pjesa 'Datanodes' si në vijim:

Hadoop Overview Datanodes Snapshot Startup Progress Utilities -										
Datanode Information										
In operation										
Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
test4-VirtualBox (192.168.56.104:50010)	0	In Service	47.17 GB	24 KB	8.35 GB	38.81 GB	0	24 KB (0%)	0	2.6.5
test2-VirtualBox (192.168.56.103:50010)	0	In Service	47.17 GB	24 KB	8.38 GB	38.79 GB	0	24 KB (0%)	0	2.6.5
test3-VirtualBox (192.168.56.102:50010)	0	In Service	47.17 GB	24 KB	8.35 GB	38.82 GB	0	24 KB (0%)	0	2.6.5
Decomissioning										
Node	Last contact	Under replicated blocks		Blocks with no live replicas		Under Replicated Blocks in files under construction				

Fig 46. Nyjet punuese (ang. slave) brenda Hadoop klasterit

Nga foto më sipër shohim se të gjitha nyjet e konfiguruar për klasterin Hadoop janë duke funksionuar. Kjo faqe paraqet të dhënat për nyjet punuese (ang. DataNodes) të cilat janë duke operuar. Të dhënat kanë të bëjnë me statusin e funksionimit të secilës nyje, kapacitetin e tyre, blloqet e replikimit të të dhënave (ang. Blocks) si dhe versionin e Hadoop-së që përdoret.

Një bllok brenda nyjeve punuese (ang. DataNode) mund të përmbajë një ose më shumë kopje të të dhënave. Kopjet e ndryshme të cilat gjenden në të njëjtin bllok mund të jenë ruajtur në nyje punuese (ang. DataNodes) të ndryshme [67]. Duke pasur parasysh këtë arkitekturë të ruajtjes dhe replikimit të të dhënave, Hadoop-ja ofron vetinë e disponueshmërisë së të dhënave në një nivel të lartë.

Në fund të paraqitjes së dokumentuar të instalimit dhe konfigurimit të platformës Hadoop, mund të konkludojmë se ndërtimi i klasterit të katër kompjuterëve ‘virtual’ është kryer me sukses. Të gjitha nyjet (serverët) të cilët përmbajnë sistemin operativ Linux Ubuntu janë instaluar brenda paketës Virtuale si VirtualBox dhe konfiguruar për të punuar së bashku. Secila nga nyjet punuese (ang. server) përmban një hapsirë të caktuar të diskut për të mbajtur paketa të ndryshme softuerike, si dhe për të ruajtur të dhënat e gjeneruara nga klateri. Klasteri i katër nyjeve punuese paraqet një rrethinë të shpërndarë e cila ofron: përshkallëzueshmërinë, disponueshmërinë si dhe koston e ruajtjes së të dhënave.