University for Business and Technology in Kosovo

UBT Knowledge Center

Theses and Dissertations

Student Work

7-2018

PERFORMANCA E MAKINAVE KERKUESE DUKE PERDORUR CLOUD SHËRBIMET "ELASTICSEARCH" DHE "ALGOLIA"

Besnik Korça

University for Business and Technology, Besnik.korca1@gmail.com

Follow this and additional works at: <https://knowledgecenter.ubt-uni.net/etd>

Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Korça, Besnik, "PERFORMANCA E MAKINAVE KERKUESE DUKE PERDORUR CLOUD SHËRBIMET "ELASTICSEARCH" DHE "ALGOLIA" (2018). *Theses and Dissertations*. 4.
<https://knowledgecenter.ubt-uni.net/etd/4>

This Thesis is brought to you for free and open access by the Student Work at UBT Knowledge Center. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UBT Knowledge Center. For more information, please contact knowledge.center@ubt-uni.net.

Programi për Shkenca Kompjuterike dhe Inxhinierisë

**PERFORMANCA E MAKINAVE KERKUESE DUKE PERDORUR
CLOUD SHËRBIMET “ELASTICSEARCH” DHE “ALGOLIA”**

Shkalla Bachelor

Besnik KORÇA

Korrik / 2018
Prishtinë

Programi për Shkenca Kompjuterike dhe Inxhinierisë

Punim Diplome
Viti akademik 2015 – 2016

Besnik Korça

**PERFORMANCA E MAKINAVE KERKUESE DUKE PERDORUR
CLOUD SHËRBIMET “ELASTICSEARCH” DHE “ALGOLIA”**

Mentor: PhDc. Krenare Pireva

Korrik / 2018

Ky punim është përpiluar dhe dorëzuar në përmbushjen e kërkesave të
pjeshme për Shkallën Bachelor

ABSTRAKT

Ueb Aplikacionet me data-set të mëdha në kohën e tanishme janë shumë më të përhapura. Kërkimet e të dhënave në data-set të tilla për shkak të performances bëhen duke përdorur makinat kërkuese. Provider të ndryshëm ofrojnë makina kërkuese si Software as a Service(SaaS) shërbime të cilat kanë për avantazh konfigurimin minimal që duhet bërë makinave kërkuese për t'i vënë në funksion. Nëpërmjet këtij punimi synohet që së pari të analizohen shërbimet cloud computing që ofrohen nga provider të ndryshëm. Pastaj pasi që fokusi i makinave kërkuese ndodhet në “information retrieval” është bërë analizimi i disa teknikave të “information retrieval”. Makinat kërkuese kanë qvllim tw njejtë, megjithatë ato dallojnë nga njëra tjetra në mënyrën se si bëjnë kërkimin dhe kthimin e të dhënave si rezultat. Ky punim është i fokusuar mbi makinat kërkuese Elasticsearch dhe Algolia, mbi karakteristikat e tyre, mënyrat se si operojnë ato si dhe dallimet në performancë gjatë kërkimit të të dhënave. Gjatë integritit të makinave kërkuese në ueb aplikacione mund të hasim ende në vështirësi gjatë konfigurimit që duhet të kryejmë ne dhe gjatë përpunimit të dhënave që na kthehen si rezultate nga makinat kërkuese. Për t'i arritur objektivate këtij punimi është zhvilluar një “throw away prototype” tek i cili kemi integruar makinat kërkuese Elasticsearch dhe Algolia. Nëpërmjet këtij prototipi janë sqaruar mënyrat e integritit të makinave kërkuese dhe vështirësit që mund t'i hasim si dhe është bërë krahasimi i performancës ndërmjet makinave kërkuese. Në krahasimin e performancës në këtë studim ku është përdorur një data-set me 1.6 milion tweets makina kërkuese Algolia dukshëm ka shfaq performancë më të mirë. Nga objektiva e analizimit të makinave kërkuese është bërë e qartë dhe është pritur një rezultat i tillë pasi që Elasticsearch rezultatet më të mira i shfaq kur operon me Big Data.

PËRMBAJTJA

LISTA E FIGURAVE	IV
LISTA E TABELAVE	IV
1. HYRJE.....	1
2. SHFLETIMI I LITERATURËS	3
2.1 Resurset harduerike dhe softuerike brenda teknologjise Cloud Computing	4
2.2 Modelet e Cloud Sherbimeve	5
2.3 Infrastructure as a Service (IaaS).....	6
2.4 Platform as a Service (PaaS)	7
2.5 Software as a Service (SaaS)	7
2.5.1 Karakteristikat dhe tiparet kryesore të Software as a Service (SaaS).....	9
2.5.2 Llojet e klientëve të holle(thin clients)	10
3. TEKNIKAT E “INFORMATION RETRIEVAL (IR)”	13
3.1 Modeli boolean retrieval.....	13
3.2 Modeli vector space.....	13
3.3 Binary independance model	14
4. DEFINIMI I PROBLEMIT	15
5. METODOLOGJIA	16
6. DIZAJNIMI DHE IMPLEMENTIMI	17
6.1 Integrimi i Elasticsearch dhe Algolia makinës kërkuese ne es-Algolia-performance-test.....	19
6.2 Integrimi i Elasticsearch	20
6.3 Integrimi i Algolia	22
7. ELASTICSEARCH.....	24

7.1 Ruajtja e të dhënave.....	25
7.2 Analizimi në Elasticsearch	27
7.3 Scoring në Elasticsearch.....	28
7.4 Menaxhimi i dokumenteve	30
7.5 Kërkimi i të dhënave.....	34
7.6 Shkallëzimi (scaling) në Elasticsearch	38
7.7 Klluster në Elasticsearch	39
7.8 Shards në Elasticsearch	40
7.9 Trajtimi i problemeve në performancë nga Elasticsearch	41
8. ALGOLIA	44
8.1 Indeksiimi	45
8.2 Kërkimi i të dhënave.....	45
8.3 Infrastruktura	46
9. KRAHASUESHMËRIA NË MES MAKINAVE KËRKUESE ELASTICSEARCH DHE ALGOLIA	48
10. REZULTATET.....	51
11. KONKLUDIMI	53
12. REFERENCAT	55

LISTA E FIGURAVE

<i>Figura 1 Karakteristikat e Cloud Computing</i>	3
<i>Figura 2 Struktura e modeleve të servisit</i>	5
<i>Figura 3 Përgjegjësitë e klientit dhe të vendorit</i>	9
<i>Figura 4 Modeli i pjekurisë i SaaS aplikacioneve</i>	11
<i>Figura 5 Paterni Arkitekturor i es-Algolia-performance-test</i>	17
<i>Figura 6 User interface i aplikacionit es-Algolia-performance-test</i>	18
<i>Figura 7 Insertimi i data-set në Algolia nëpërmjet dashboard-it që ofron</i>	22
<i>Figura 8 Krijimi, indeksimi, fshirja e dokumenteve në Elasticsearch</i>	32
<i>Figura 9 Kërkimi nëpërmjet query</i>	36

LISTA E TABELAVE

<i>Tabela 1 Dallimet kryesore ndërmjet makinave kërkuese Algolia dhe Elasticsearch</i>	
<i>Tabela 2 Rezultatet e krahasimit të performancës të Algolia dhe Elasticsearch si SaaS servise</i>	

1. HYRJE

Suksesi i një aplikacioni varet nga vijueshmëria e klientëve, nga atraksioni që ofron ai aplikacion dhe nga eksperiencia e klientit në të. Në kohët moderne vërja e vëmendjes nga ana e klientëve është duke ra më shumë e sipër, gjë që si pasojë ka kohë më të shkurtër të vështirimit të një aplikacioni. Në shumicën e rasteve përveç se duhet të kemi user interface të mirë, instinktive për tu përdorur duhet edhe që t'i ofrohet klientit një mënyrë e shpejtë e kërkimit të dhënave. Një ndër format më të preferuara të kërkimit të dhënave është kërkimi në kohë reale që është popullarizuar nga ana e kompanive të mëdha. Deri vonë implementimi i kërkimeve të tilla në aplikacionet tona ka qenë gati se i pamundur sidomos në data-set të mëdha mirëpo gjithçka ndryshoi me shfaqjen e makinave kërkuese në treg. Kërkimi në kohë reale apo pothuajse reale është bërë i mundshëm edhe për ndërmarrjet e vogla nëpërmjet makinave kërkuese. Pasi që ekzistojnë një numër i madh i makinave kërkuese të cilat nuk mund të përfshihen të gjitha, në këtë punim diplome janë trajtuar Elasticsearch dhe Algolia. Makinat kërkuese janë të optimizuara në kërkim të dhënave dhe renditjen e rezultateve në bazë të relevancës. Secila makinë kërkuese përdor algoritme specifike të kostumizuara për të arritur këtë qëllim. Në kapitullin e shtatë dhe në kapitullin e tetë është bërë një sqarim i mirëfilltë i operimit të makinave kërkuese Elasticsearch dhe Algolia nga indeksimi e deri tek kërkimi i të dhënave ndërsa në kapitullin e tretë flitet mbi modelet më të mira për information retrieval të cilat përdoren nga ana e makinave kërkuese. Operimi i makinave kërkuese gjithashtu mund të paraqet probleme në konfigurim si dhe mund të shfaqen probleme në sigurimin e resurseve kompjuterike mbi të cilat do të operon makina kërkuese. Në raste të tilla si zgjidhje paraqiten serviset të cilat ofrohen nga provider të ndryshëm tek të cilët merret përsipër konfigurimi dhe gjetja e fuqive procesuese për të mbajtur në funksion makinën tonë kërkuese. Në kapitullin e dytë është analizuar më gjerësisht ofrimi i shërbimeve të tilla të cilat njihen si SaaS servise. SaaS serviset hyjnë në kuadër të cloud computing me ç'rast është trajtuar edhe kjo tematikë në kapitullin e dytë. Zgjidhja e makinës kërkuese që do të përdorim shpesh herë din të paraqitet si një problem. Për të bërë përzgjedhjen e makinës kërkuese duhet t'i krahasojmë ato ndërmjet veti duke marrë për bazë faktorë të ndryshëm që karakterizojnë

makinën kërkuese mirëpo edhe faktorë të performancës siç janë shpejtësia e kthimit të rezultateve si dhe fluksi i dokumenteve të kthyera. Karakteristikat e makinave kërkuese Algolia dhe Elasticsearch janë krahasuar në kapitullin e nëntë. Për të analizuar procedurën e integritit të makinave kërkuese në ueb aplikacione si dhe për të krahasuar performancën e makinave kërkuese si servise SaaS është zhvilluar aplikacioni es-Algolia-performance-test. Procesi i zhvillimit si dhe arkitektura softuerike e këtij aplikacioni është shtjelluar në kapitullin e gjashtë ku edhe janë prezantuar fotografi nga pamja përfundimtare e atij aplikacioni. Së fundmi është bërë një krahasim i performancës të makinave kërkuese Algolia dhe Elasticsearch rezultatet e së cilës janë prezantuar në kapitullin e 10.

2. SHFLETIMI I LITERATURËS

Cloud Computing është një qasje unike sa i përket llogaritjeve kompjuterike e cila është adoptuar nga shumë organizata për të arritur deri tek resurset e përbashkëta [1]. Cloud Computing mundëson çasje nëpërmjet rrjetit në momentin kur nevojiten resurset e përbashkëta që mund të sigurohen dhe të lancohen me përpjekje minimale sa i përket menaxhimit dhe konfigurimit [2]. Shërbimet që ofrohen nëpërmjet Cloud Computing ulin koston e inicimit, zhvillimit, mirëmbajtjes, zgjerimit, etj.

Figura 1 Karakteristikat e Cloud Computing

Inspirimi nga [3]

Nëse e vështrojmë nga një pikëpamje ekonomike atëhere një definicion adekuat për Cloud Computing është teknologjia e peste e llogaritjes e cila ka zhvendosur pjesën e procesimit nga ana e klientit tek ana e servereve. Tutje, me Cloud Computing i referohemi “shpërndarjes së resurseve IT në bazë të kërkesës dhe aplikacioneve nëpërmjet Internetit duke përdorur

principin çmimorë “pay-as-you-go” tek i cili bëhet pagesa për aq sa shpenzohet”. Kjo veçori ofron zgjedhje për kompanitë të cilave ju nevojitet të operojnë në analizim të dhënave në masë të madhe por të cilat nuk dëshirojnë të investojnë në themelimin dhe mirëmbajtjen e infrastrukturës së tyre kompjuterike [4].

2.1 Resurset harduerike dhe softuerike brenda teknologjise Cloud Computing

Pjesa atraktive tek teknologjia cloud computing është ofrimi i shërbimeve që janë të shkallëzueshme (scalable) dhe elastike në mënyrë të menjëhershme (instante). Resurset të cilat përdoren për këto servise janë të përbashkëta për të gjithë përdoruesit ku çdo përdorues faturohet vetëm për aq sa ka shfrytëzuar resurse [3]. Resurset e përbashkëta në cloud computing janë rrjetat, serverat, hapësira memorizuese, aplikacionet dhe serviset. Klienti nuk është në dijeni apo në kontroll se ku ndodhen saktësisht resurset të cilat është duke i shfrytëzuar mirëpo i ofrohet caktimi i lokacionit në mënyrë të gjeneralizuar pra mund të specifikohet si lokacion qyteti, shteti, etj [2]. Megjithatë, mos njohuria e plote për vendodhjen e shërbimeve, shumë kompani kanë qenë fillimisht mjaft skeptike për të përkrahur një teknologji të tillë, sidomos në atë vend apo kompani ku të dhënat janë pikat më të ndjeshme të bizneseve të tyre. Multiplexing mundëson shfrytëzim më të lartë të resurseve pasi që kur disa aplikacione ndodhen brenda të njëjtit sistem në kushte normale ato nuk do të jenë gjatë tërë kohës duke performuar me kërkesa maksimale të resurseve mirëpo do të ketë ndonjë lloj cikli që mundëson përdorim më të mirëfilltë të resurseve [3].

Lloji i Serverit Storage përdoret për të menaxhuar të dhëna digjitale, files dhe shërbime. Qëllimi i këtyre serverëve gjendet në ruajtjen e të dhënave dhe qasjes në një sasi të limituar të këtyre të dhënave nëpërmjet një rrjete të përbashkët apo nëpërmjet internetit. Zakonisht ky lloj i serverit është më i dobët në performancë në krahasim me serverët standard mirëpo e kompenson këtë mungesë performance duke ofruar¹ hapësira të mëdha memorizuese.

¹Server Storage, <https://www.techopedia.com/definition/9550/storage-server>

2.2 Modelet e Cloud Sherbimeve

Cloud Computing mund të kategorizohet në tri modele të cloud serviseve të cilat janë [2]:

- Infrastructure as a Service (IaaS),
- Platform as a Service (PaaS), dhe
- Software as a Service (SaaS).

Figura 2 Struktura e modeleve të servisit [3]

Në figurën 2 shihen kjo strukturë e modeleve të shërbimeve e cila është më e thjeshtë tek modeli SaaS dhe më tutje ndërlikohet tek modelet tjera. Është e rëndësishme që të kuptohen këto modele të shërbimeve pasi që do të vazhdojnë të bashkë ekzistojnë në të ardhmen e

parashikueshme. Servis modeli SaaS ka potencial që të jetë shumë i popullarizuar pasi që mundëson shërbimin edhe ndaj personave me njohuri të limituara ndërsa IaaS përdorimin më të madh do të gjejë tek personat me njohuri të mirëfillta në zhvillim softuerik. Klienti nuk menaxhon apo kontrollon cloud infrastrukturën, rrjetat, serverat. Servis modeli SaaS i vë në dispozicion mundësinë klientit për të përdor aplikacione të ofruara nga vetë provideri mirëpo nuk ka kontrollë mbi platformën apo infrastrukturën. PaaS mundëson krijimin dhe lancimin e aplikacioneve nëpërmjet gjuhëve programuese dhe veglave të cilat i ofron provideri. Në servis modelin IaaS klienti mundet të përdor çfarëdo softueri arbitrar duke përfshirë sisteme operative dhe aplikacione. Këto modele bazohen në model çmimor të shfrytëzimit të resurseve. Në qoftë se përpjekjet për standardizim arrijnë sukses atëherë do të mund të bëhej migrimi i PaaS nga një infrastrukturë në tjetrën duke evituar problemin e të qenit i lidhur për një vendor. Një seri e aktiviteteve janë të nevojshme për të mbështetur këto tre modele të servisit që përfshijnë menaxhimin e serviseve, provizionimi i resurseve, menaxhimi i sigurisë, shërbimi ndaj klientëve, shërbime integrimi [3].

2.3 Infrastructure as a Service (IaaS)

Modeli “Infrastructure as a Service (këtu e tutje IaaS)” në bazë të klasifikimit SPI është kategoria e tretë e shërbimeve që ofron resurse të infrastrukturës siç është CPU, memorie dhe hapësirë për ruajtjen e të dhënave [5]. NIST definon Infrastructure as a Service si “aftësia e ofruar ndaj konsumatorit për të vënë në dispozicion procesimin, ruajtjen, rrjetat, dhe resurse të tjera fundamentale kompjuterike ku konsumatori është në gjendje të bëjë lancimin dhe ekzekutimin e ndonjë softueri arbitrar, që mund të jenë aplikacione mirëpo edhe sisteme operative”. Konsumatori nuk menagjon e as nuk kontrollon infrastrukturën themelore të cloudit por ka kontroll mbi sistemin operativ, hapësirën memorizuese, dhe mbi lancimin e aplikacioneve; dhe në disa raste kontroll të limituar mbi komponentët e përzgjedhura të rrjetës si p.sh firewall i hostimit [2].

2.4 Platform as a Service (PaaS)

Platform as a Service (këtu e tutje PaaS) është një model i ofrimit të shërbimeve tek e cila provideri i ofron klientit mundësinë të krijojë aplikacione duke përdorur veglat e zhvillimit të cilat i ka vënë në dispozicion provideri. Klientët e kësaj kategorie të shërbimit zakonisht janë zhvillues të softuerit, ueb zhvillues apo biznese që përdorin API specifik të suportuar nga provideri për të ndërtuar, testuar dhe zhvilluar aplikacionet e tyre në cloud. Shërbimet e ofruara nga providerët përfshijnë ambiente të gjuhëve programuese, ueb server dhe baza të shënimeve. PaaS providerët abstrahojnë kompleksitetin e menagjimit të një serveri duke i dhënë klientit një ambient ku ata mund të fillojnë zhvillimin, testimin dhe lancimin e aplikacioneve mirëpo duke e marrë prej tyre ngarkesën e menagjimit të serverit. PaaS mund të shpërndahet si një shërbim publik i cloud-it nga provideri, ku klientët kontrollojnë aplikacionet dhe disa konfigurime minimale ndërsa provideri merret me sistemin operativë, bazën e shënimeve, middleware, rrjetat, serverët, etj [2].

Serviset e PaaS përfshijnë dizajnim, zhvillim, testim, lancim (deployment) dhe hostim. Shërbime të tjera përfshijnë kolaborim ekipor, ueb servise të integruara, integrim i bazës së shënimeve, siguri, shkallëzim (scalability), ruajtje të dhënave, versionim etj [6].

2.5 Software as a Service (SaaS)

Software as a Service (tash e tutje SaaS) është një model zbarkimi (delivery) në të cilin ofrohet një Software aplikacion si një servis i menaxhuar ku konsumatorët thjeshtë paguajnë për çasje në intervale kohore të caktuara. Tenant quajmë një grup të përdoruesve me privilegje të nejta që i qasen instancës të softuerit. Cloud aplikacionet SaaS mund të ndërtohen si multi-tenant aplikacione me c'rast ekziston një instancë e vetme të cilën e përdorin disa Tenant-s. Arkitektura softuerike multi-tenant përdoret për të zvogëluar shpenzimet duke qenë më kosto eficient në përdorimin e instancave të përbashkëta (shared). SaaS aplikacionet mund të paraqiten të dobishme edhe në gjenerimin e të dhënave analitike [7].

Software as a Service na mundëson të evitojmë instalimet lokale, mirëmbajtjet dhe azhurimet (update) e aplikacioneve [4].

Konsumatorët nuk menaxhojnë infrastrukturën e cloudit, komponentët e rrjetës, serverët, sistemet operative apo ruajtjen e të dhënave. SaaS eliminon shpenzimin e përvetësimit të harduerit. Thin client-s janë softuer të thjeshtë që shërbejnë për të shfaqur rezultatet nëpërmjet një user interface të thjeshtë. Sqarimi i mëtutjeshëm mbi thin clients bëhet në nënkapitullin në vijim 2.5.2 Llojet e klientëve të holle(thin clients). Qasja në servis bëhet e mundshme për klientin duke përdorur një klient të hollë (thin client) siç është zero client ose ueb client [2]. Nga kjo mund të konkludojmë se në modelin SaaS provider të palës së tretë hostojnë aplikacione dhe i bëjnë të qasshme për klientët nëpërmjet internetit. Ne kuader te kesaj teme te diplomes, fokusi kryesore do te jete ne kete model te ofrimit te sherbimeve, perkatesisht ne ofrimin e ElasticSearch sherbimit, i cili i takon SaaS kategorise, e qe ne realitet eshte sherbim i ofruar nga ofruesi Amazon, e qe per me shume do te elborohet ne kapitullin e gjate dhe shtate. Andaj, me tutje ne do te perqendrohemi me shume ne shtjellimin e karakteristikave, tipareve, llojeve te klientit te holle, si dhe modelit te pjekurise per ti paraprire tutje definimit te problemit, perkatesisht objektivave te kesaj teme.

Figura 3 Përgjegjësitë e klientit dhe të vendorit ²

2.5.1 Karakteristikat dhe tiparet kryesore të Software as a Service (SaaS)

SaaS ka aftësinë të servoj shërbimin tek disa përdorues paralelisht. Ofron mundësi të shumta të personalizimit dhe nevojitet që të aktualizohet vetëm në cloud, në mënyrë që klientët të mund të përdorin versionin e personalizuar. Shpenzimet në SaaS bazohen në Pay-per-use koncept ku paguan atë që është përdorur servisi. Për atë që servisi është në serverin e ofruesit të shërbimit klientët mund t'i çasen rregullisht [1]. Të dhënat ruhen në qendrën e të dhënave

²IaaS, PaaS dhe SaaS, <https://venturebeat.com/2011/11/14/cloud-IaaS-PaaS-SaaS>

të furnitorit të shërbimit ku janë edhe të sigurta në rast të ndonjë ndërprerje apo edhe në rast të dështimit total të makinës së klientit [3].

2.5.2 Llojet e klientëve të holle(thin clients)

Cloud aplikacionet bazohen në arkitekturën klient server ku nga ana e klientit përdoret një Software i thjeshtë i cili njihet si thin client. Thin clients operojnë pra në makinën e klientit ndërsa të gjitha veprimet kalkuluese kryhen në server [3].

Për të qenë alternativë e vlefshme në krahasim me aplikacionet lokale nevojitet që thin client të shfaq output audio-vizual të rrjedhshëm. Gjithashtu serveri tek i cili janë duke u performuar operacionet duhet të ketë resurse kompjuterike të mjaftueshme dhe të ndodhet në një lokacion të afërt ndaj pozitës së përdoruesit për të zvogëluar vonesat e rrjetës [8].

2.5.3 Modeli i pjekurisë (Maturity Model) tek Software as a Service.

Modelet e pjekurisë kanë fushëveprim më të gjerë se sistemet SaaS dhe kryesisht janë të koncentruara në dërgimin e shërbimeve nga ana e providerit. Ekzistojnë dy modele të pjekurisë (maturity) të cilët janë modeli i maturisë i Microsoft dhe modeli i maturisë Forrester. Për qëllime të këtij punimi do t'i sqarojmë vetëm nivelet e pjekurisë (maturity) të modelit të Microsoft. Në vitin 2006 Microsoft propozoi një model të pjekurisë i cili është i ndarë në katër nivele për sistemet SaaS. Autori na lë të kuptojmë se aplikacionet SaaS mund të klasifikohen duke marrë parasysh tre attribute arkitekturore [5]:

- configurability,
- efikasiteti multi-tenant dhe
- shkallëzimi (scalability).

Modeli i Pjekurisë (Maturity) i SaaS nga Microsoft

*Figura 4 Modeli i pjekurisë i SaaS aplikacioneve*³

Në figurën 4 mund të shihen dallimet ndërmjet niveleve të modelit të pjekurisë. Më tutje janë sqaruar nivelet e modelit të pjekurisë të cilat kanë gjithmonë ka një atribut më shumë sesa ai paraprak.

Niveli i parë është “Ad-hoc/custom” tek i cili çdonjëri nga klientët ka një instancë të izoluar të aplikacionit nga instancat e klientëve tjerë, kjo instancë e aplikacionit e personalizuar operon në serverët e hostuar nga ana e providerit. Përparësia e këtij niveli qëndron tek munda i vogël që nevojitet në zhvillim për të zhvendosur një aplikacion lokal apo client-server aplikacion në këtë nivel.

Niveli i dytë karakterizohet me atributin e të qenit i konfigurueshëm. Metadata quajmë të dhënat që definojnë sjelljen e një softueri. Ky nivel ofron fleksibilitet më të lartë të aplikacionit nëpërmjet metadatave të konfigurueshme. Ky është një avantazh i cili i mundëson ofruesit t’i përmbush nevojat për secilin klient nëpërmjet alternativave të detajuara të konfigurimit.

Niveli i tretë i modelit të pjekurisë SaaS përpos konfigurueshmërisë veçohet me atributin e radhës që është efikasiteti multi-tenant. Ofruesi vë në dispozicion një instancë të vetme të aplikacionit për çdo klient me metadata të konfigurueshme për të siguruar një eksperiencë

³ MODEL I PJEKURISE NE SAAS, <http://slideplayer.com/slide/11056942/>

unike dhe të personalizueshme për konsumatorin. Kjo instancë e vetme lejon përdorim efikas të resurseve kompjuterike gjë që si rezultat ka kosto më të ulët sesa niveli i dytë i modelit të pjekurisë të SaaS. Niveli i katërt i cili është më i larti në modelin e maturisë të Microsoft përpos se ofron konfigurueshmëri dhe efikasitet multi-tenant gjithashtu është i shkallëzueshëm (scalable). Shkallëzimi (scalability) është një avantazh për providerin që arrihet nëpërmjet një arkitekture disa shtresore. Në rastin e konsumatorëve të rinj, mund të shtohet një instancë e re e aplikacionit në pool në mënyrë dinamike për të ofruar mbështetje për ngritjen e ngarkesës. Ky model përdor një balancues të ngarkesës të tenants (round, robin, rule based) për të arritur shfrytëzimin e resurseve të hostimit (CPU, hapësirë memorizuese, etj) në maksimum gjë që si rezultat do të kishte distribuimin e ngarkesës mbi të gjithë infrastrukturën [5].

3. TEKNIKAT E “INFORMATION RETRIEVAL (IR)”

Me “information retrieval” nënkuptojmë gjetjen e materialeve që zakonisht janë dokumente (të dhëna) të pastrukturuara të cilat përmbushin ndonjë nevojë për informacione të cilën e kemi. Si të pastrukturuara i konsiderojmë të dhënat të cilat nuk kanë formë të qartë, nuk ndjekin ndonjë logjikë të organizimit dhe ato që janë vështirë për tu kuptuar nga kompjuteri [9]. Me tutje do të shpjegohen disa teknika të cilat përdoren kur kemi të bëjmë me marrjen e informatave, pasi ti kemi kërkuar ato.

3.1 Modeli boolean retrieval

Tek modeli set-theoric dokumentet paraqesin një seri të fjalëve dhe të frazave. Një ndër modelet më të përhapura të modelit set-theoric është boolean retrieval. Modeli boolean retrieval është një model që përdoret për kthimin e informacioneve duke përdorur “query” të cilat kanë shprehje boolean. Në këtë model query mund të kombinohen me operatorët:

- dhe,
- ose,
- asnjëherë.

Për një pjesë të madhe të boolean query duhet të llogaritet dhe të ruhet përkohësisht rezultatet për shprehje të mëtutjeshme. Mirëpo për shkak të natyrës të gjuhëve për query në shumicën e rasteve në vend se të bëhet bashkangjitja e dokumenteve preferohet të bëhet prerja e listave të dokumenteve me listën e dokumenteve që ndodhet në memorie [9].

3.2 Modeli vector space

Modelet tek të cilat dokumentet dhe queries reprezentohen si vektorë, matrica apo tulpa quhen modele algjebrike. Në këtë punim diplome do të fokusohemi në modelin vector space që bën pjesë në modelet algjebrike. Sot, materialet e listuara pasi që ti kemi kërkuar ato, shpesh herë nevojiten të rankohen dhe të listohen varësisht relevancës së tyre. Ne rastet e tilla,

kemi të bëjmë me modelin ranked retrieval, ku zakonisht dokumentet renditen në bazë të relevancës apo impaktit, me ç'rast i pari renditet dokumenti i cili ka relevancën më të madhe. Nje nder modelet me te shpeshta qe perdoret per te rankuar dokumentet varesisht relevances ne mes “query” dhe materialeve poseduese, eshte vektor space. Në modelin vektor space çdo dokument trajtohet si një vektor me ç'rast kalkullohet një peshë për çdo query dhe dokument. Modeli vektor space është fundamental për information retrieval për caktim të relevancës dhe klasifikim të dokumenteve. Variacionet e metodave të modelit vektor space dallojnë vetëm nga pesha e cila ju caktohet vektorëve [9].

3.3 Binary independance model

Ekziston edhe modeli probabilistik i information retrieval. Në këtë model versioni original dhe me më së shumti influencë është binary independance model. Teoria e probabilitetit ofron një zgjidhje për rastet e pasigurisë tek të cilat sistemit i ofrohet një query dhe një dokument me ç'rast sistemi duhet të hamendësoj se sa relevant është ai dokument për queryin. Tek binary independence model termi binary ka kuptimin boolean, nga kjo mund të kuptojmë se dokumentet dhe queries reprezentohen si vektorë binary i rastisjes të termit. Me këtë formë të reprezentimit ndodh që shumë dokumente të kenë të njëjtin vektor të prezantimit. Fjala independencë në emrin e Binary independance model qëndron për shkak se termet në dokumente konsiderohen të pavarura nga njëra tjetra. Pra nuk bëhet asnjë modelim i cili asocion termet që ndodhen në dokumente. Ky është një supozim i mangët mirëpo në raste reale ky model ka gjetur zbatim për shkak të rezultateve të mira që i ka shfaqur [9].

E gjithë literatura e shfletuar, do të shërbejë në definimin adekuat të problematikës së kësaj teme të diplomës, duke ofruar një njohuri bazike mbi konceptet dhe shërbimet të cilat do të përdorën më tutje. Andaj në kapitullin në vazhdim do të definojmë problemin të cilin do të shtjellojë kjo temë e diplomës, dhe objektivat e ndjekura për të përbushur kërkesat e problematikës.

4. DEFINIMI I PROBLEMIT

Deri më tani kërkimi në kohë reale kanë qenë shërbim i rezervuar për kompanitë e mëdha, mirëpo kjo ka ndryshuar me shfaqjen e makinave kërkuese që përdorin cloud shërbimet e ofruar nga ofrues të ndryshëm, që në këtë rast mund të përmendim shërbimet Elasticsearch dhe Algolia. Elasticsearch ofrohet si SaaS cloud shërbim nga Amazon ndërsa Algolia ofrohet vetëm si cloud shërbim nga kompania që e ka zhvilluar atë. Ka raste kur konfigurimi, lancimi dhe integrimi i makinave kërkuese mund të komplikohet. Andaj në kuadër të kësaj teme, është tentuar të sqarohen mënyrat e operimit të makinave kërkuese përmes Elasticsearch dhe Algolia. Tutje është tentuar të evidentohen sfidat të cilat paraqiten në fazat e integritit të makinave kërkuese si SaaS shërbime në ueb aplikacione dhe të krahasohen performancat e makinave kërkuese duke përdorur një data-set të ofruar nga sentiment140 që është një shërbim që ofron të dhëna nga produktet e kompanisë “Twitter”. Tutje, në kuadër të kësaj teme është prezantuar eksperiencia jonë gjatë integritit të makinave kërkuese.

Objektivat e këtij hulumtimi janë:

- Analizimi i operimit të makinave kërkuese
- Integrimi i SaaS shërbimeve në makinat kërkuese duke përdorur teknologjinë ueb.
- Krahasimi i performancës të makinave kërkuese varësisht përdorimit të SaaS shërbimeve.

5. METODOLOGJIA

Në kuadër të kësaj teme, për të trajtuar problemin e definuar në kapitullin e katërt, ne kemi përdorur një metodologji të miksuar, duke tentuar të gërshetojmë metoda të ndryshme hulumtuese për objektiva të ndryshme. Fillimisht për të arritur rezultatet e dëshiruara në objektivën e parë, ne kemi përdorur metodën e shfletimit të literaturës, përmes të cilës së pari janë hulumtuar modelet e shërbimeve të cloud computing me fokus të veçant mbi SaaS shërbimet dhe teknikat e “information retrieval” për të fituar një dije gjithëpërfshirëse mbi teknikat që janë në dispozicion sa i përket kërkimit dhe kthimit të rezultateve nga një burimi i të dhënave. Në vijim është bërë hulumtimi i operimit të makinave kërkuese elasticsearch dhe algolia. Në kapitujt sqarues të operimit të makinave kërkuese është përfshirë edhe konfigurimi i makinave kërkuese të lartëpërmendura gjë që do të jetë e dobishme edhe për plotësimin e objektivës së dytë të kësaj teme. Gjithashtu është sqaruar se si bëhet indeksimi i të dhënave, manipulimi me të dhëna(futje, fshirje, azhurim) dhe kërkimi në të dhëna duke përdorur “query” të ndryshëm. Më pastaj, në kuadër të kësaj teme karshi shfletimit të literaturës dhe gjetjeve më të reja në këtë sferë, ne kemi krijuar një prototip hulumtues përmes të cilit kemi krijuar një makinë kërkuese duke përdorur SaaS shërbimet elasticsearch që ofrohet nga Amazon Web Service (AWS) dhe Algolia që ofrohet si SaaS shërbim nga vetë kompania Algolia. Për zhvillimin e këtij prototipi si gjuhë programuese në “server-side” është përdorur php, për të lehtësuar punën e zhvillimit është përdorur framework-u më i përhapur i php që quhet “laravel”. Si “front-end framework” për dizajnimin e user interfaces është përdorur vue, ndërsa për të lehtësuar shkruarjen e css është përdorur një preprocesor i quajtur scss. Tutje, ne kemi përdorur metodën krahasuese, duke krahasuar shërbimet e ofruara nga elasticsearch dhe algolia ne kushte identike(duke përdorur të njëjtin dataset).

6. DIZAJNIMI DHE IMPLEMENTIMI

Paterni i arkitekturës softuerike i cili është përdorur në këtë projekt është arkitektura tri shtresore. Paterna 3 shtresore e prototipit tonë mund të shihet në figurën 5 ku në shtresën user interface bëhet prezantimi i të dhënave, në shtresën e biznes logjikës ku ndodhet logjika funksionale e proceseve dhe shtresa e qasjes në të dhëna. Arkitektura softuerike e këtij programi mund të shihet në diagramin të paraqitur në vijim.

Figura 5 Paterni Arkitekturor i es-Algolia-performance-test

Ueb aplikacioni es-Algolia-performance-test është zhvilluar duke përdorur metodologjinë agjile scrum. Zhvillimi i këtij aplikacioni është ndarë në tre sprinte (iterime) ku në sprintin e parë është bërë krijimi i projektit integrimi i veglave dhe librarive tjera që do të përdorën në këtë projekt siç janë vue.js, Elasticsearch.js, algoliasearch.js . Në sprintin e dytë janë

përgatitur user interfaces të cilat do të përdoren për shfaqjen e të dhënave të cilat kërkohen në makinat kërkuese. Këto user interfaces janë zhvilluar duke përdorur vue.js gjë që lehtëson zhvillimin e mëtutjeshëm pasi që për shkak të vetive react të vue.js bëhen rerender rezultatet në mënyrë automatike sa herë që ndryshojnë ato. Pas ndryshimit të fjalëve me të cilat është duke u kërkuar ekzekutohet kërkimi në makina kërkuese. Dhe në sprintin e tretë është bërë konfigurimi dhe testimi i makinave kërkuese që do të përdoren për të bërë kërkimin dhe kthimin e dokumenteve.

Performance Test | Algolia & Elasticsearch

z

Stats

Search Engine	Processing Time	Number of Results
ElasticSearch	106 ms	1705
Algolia	1 ms	22487

elasticsearch

algolia

id	polarity	username	text	date
1564565965	4	teamftskaty	@Zackalltimelow Happy birthday Zack.	Mon Apr 20 04:04:12 PDT 2009
1573368498	4	wtfxtina	@zackalltimelow happy birthday zack!!!	Tue Apr 21 00:51:11 PDT 2009
2011823111	4	noblonkat	@zackalltimelow anreed wise words Zack	Wed Jun 03 00:00:00 PDT 2009

id	polarity	username	text	date
1882993637	4	zeezo0000	now i feel better than yesterday thankyou my bestFI	Fri May 22 07:49:59 PDT 2009
1882954352	4	zr_l	@mnowluck @chhama me @ home n u know how the internet connection is like out here (keep on tweeting ...will b back soon	Fri May 22 07:46:05 PDT 2009
1882954357	4	7arrth	TOIE Off to enjoy the weekend	Fri May 22 07:46:05 PDT 2009

Figura 6 User interface i aplikacionit es-Algolia-performance-test

Aplikacioni es-Algolia-performance-test shërben për demonstrimin dhe krahasimin e kërkimit të dhënave në makinat kërkuese Elasticsearch dhe Algolia. User interface i prototipit mund të shihet në figurën 6 ku në anën e majtë listohen rezultatet e kërkimit në elasticsearch, në anën e djathtë ndodhen rezultatet e kërkimit në algolia, ndërsa në pjesën qendrore mund të shihet koha e kthimit të rezultateve dhe numri i rezultateve. Si data-set janë përdorur 1.6 milion tweets të cilat janë ngarkuar në të dy makinat kërkuese që të bëhet kërkimi në të dhëna

të njëjta. Në të ndodhet një fushë për kërkim tek e cila mund të shkruhen fjalët kyçe. Për shkak të reaktivitetit të vue.js me ndryshimin e përmbajtjes në fushën për kërkim iniciohet ekzekutimi i kodit që ka të enkapsuluar libraritë Elasticsearch.js dhe algoliasearch.js nëpërmjet të cilave bëhet kërkimi në makina kërkuese. Pasi që të mirët përgjigjja nga makina kërkuese, rezultatet bëhen render nëpërmjet vue.js . Si rezultat shfaqën kolonat e tweets që janë kthyer si rezultat. Përpos kësaj të dy makinat kërkuese kthejnë kohën që ju është nevojitur për të kryer kërkimin të shprehur në milisekonda. Në aplikacion gjithashtu është shfaqur numri i dokumenteve të cilat ju janë përshtatur fjalëve kyçe në bazë të cilave kemi kërkuar. Ky aplikacion mund të përdoret për krahasimin e makinave kërkuese në performancë në bazë të dy elementeve që janë shpejtësia me të cilën kthehen rezultatet dhe numri i rezultateve.

6.1 Integrimi i Elasticsearch dhe Algolia makinës kërkuese ne es-Algolia-performance-test

Aplikacioni es-Algolia-performance-test është zhvilluar duke përdorur gjuhën programuese php në backend. Më saktësisht është përdor framework Laravel i cili është më i përdoruri kur kemi të bëjmë me teknologjinë php. Laravel lehtëson procesin e zhvillimit duke ofruar zgjidhje të gatshme ose të thjeshtësuara për shumë procese si p.sh autentifikim, routing, sesione, cacheing etj ⁴. Në front end si teknologji për menaxhim të user interfaces të aplikacionit es-Algolia-performance-test është përdorur vue.js . Vue është një framework progresivë, libraria bazë e së cilës është e fokusuar në shtresën e prezantimit të dhënave (view layer). Në rastin tonë vue.js është përdorur për një single-page aplikacion e cila është e aftë të fuqizojë edhe single-page aplikacione të sofistikuar sidomos kur mbështetet nga ndonjë librari komplementare. Në zhvillimin e këtij aplikacioni gjithashtu është përdor Syntatically Awesome Style Sheet(tani e tutje saas) që është një gjuhë skriptuese e cila përdoret për të optimizuar dhe për të lehtësuar shkrimin e Cascade Style Sheet filave që

⁴Vue.js <https://vuejs.org/v2/guide>

⁴Laravel <https://laravel.com/docs/4.2/introduction>

përdoren për të caktuar mënyrën se si do të prezantohet përmbajtja e ueb faqeve. SaaS lejon përdorimin e variablave, nested rules, mixins etj.

Pasi që shfletuesit (Browsers) nuk mbështesin saas (syntactically awesome style sheets) duhet që të konvertohet në css të thjeshtë. Shumica e shfletuesve gjithashtu nuk mbështet vue.js me ç'rast ato gjithashtu duhet të kompilohen në javascript files të thjeshtë për të hasur në mbështetje nga të gjithë shfletuesit ⁶. Për këtë është përdorur webpack i cili në thelb është një statik module bundler. Webpack përdoret në aplikacione për të gjeneruar modulet dhe për të ndërtuar një graf të varshmërive që i mapon të gjitha modulet e aplikacionit që nevojiten⁷.

Objektivë e këtij punimi ka qenë gjithashtu krahasimi i performancave të makinave kërkuese që përdorin Elasticsearch dhe Algolia shërbimet. Si burim i të dhënave për krahasimin e performancave është përzgjedhur një data-set i ofruar nga Sentiment140 i cili përbëhet nga 1.6 milion tweets (mesazheve në rrjete sociale) të cilave ju janë larguar të gjithë karakteret special, siç është rasti me karakterat që shprehin emocione (emoji). Ky data-set është shkarkuar në formatin csv nëpërmjet linkut të ofruar në faqen e sentiment140⁸.

Për të lehtësuar implementimin e makinave kërkuese janë përdor algoliasearch.js dhe Elasticsearch.js si librari komplementare mbi të cilat do të flitet më shumë në vijim.

6.2 Integrimi i Elasticsearch

Libraria Elasticsearch.js është implementuar në aplikacionin es-Algolia-performance-test për të lehtësuar komunikimin me shërbimin SaaS të Elasticsearch të cilin e ofron Amazon ueb services (AWS). Elasticsearch është ndërtuar mbi Lucene e cila është një search engine librari, open source që është e shkruar tërësisht në java. Lucene është libraria bazë e Elasticsearch e cila i mundëson implementimin e funksionalitetit të kërkimit. Elasticsearch ofron një API më të thjeshtë dhe ofron funksionalitet më të gjerë në ruajtje, indeksim dhe

⁵Syntactically awesome style sheet(saas), http://sass-lang.com/documentation/file.SASS_REFERENCE.html

⁶Vue.js, <https://vuejs.org/v2/guide/index.html>

⁷Webpack, <https://webpack.js.org/concepts>

⁸Sentiment140, <http://help.sentiment140.com/for-students>

kërkim më efektiv sesa Lucene në vete. Një ndër avantazhet më të mëdha të Elasticsearch është kompatibiliteti me çfarëdo gjuhë programuese pasi që ofron HTTP API të vetin nëpërmjet të cilit të gjitha kërkesat ndaj saj mund të bëhen nëpërmjet protokollit http në formatin json [10].

Pra komunikimi me Elasticsearch mund të bëhet nëpërmjet ndonjë aplikacioni që përdor RESTful API, shfletuesve apo edhe prej terminalit. Elasticsearch në kushte normale pasi të shkarkohet në një pajisje mund të fillojë punën menjëherë pasi që ka konfigurime default kështu që të gjitha detyrat administrative vetëm se janë duke u kryer në prapavijë, duke përdorur parametra default [11].

Elasticsearch është efikas në kërkim për shkak se bën indeksimin e të gjitha të dhënave automatikisht, indeksimi në Elasticsearch bëhet nëpërmjet inverted indexing e cila krijon një strukturë të dhënave e cila bën memorizimin se në cilat të dhëna ndodhet dhe sa herë përsëriten termet që I përmban ai dokument. Inverted indexing përdoret kryesisht për makina kërkuese pasi që mundëson të dihet se sa herë është paraqitur një fjalë dhe në bazë të kësaj gjatë kërkimit mund të vlerësohet se çfarë peshe do të qojë ajo fjalë në kërkim. Kjo strukturë e të dhënave optimizon kërkimin në të dhëna pasi që mundëson kthimin e dokumenteve të cilat kanë relevancën më të madhe në bazë të fjalëve kyqe të cilat janë dhënë gjatë kërkesës për kërkim të dhënave [10].

Për të shkarkuar Elasticsearch.js në aplikacion është përdorur NPM pastaj është bërë konfigurimi i kllusterit të Elasticsearchit dhe indeksit në të cilin do të kërkohet. Për indeksimin e data-set sentiment140 është përdorur logstash që është në gjendje të dërgojë edhe sasi më të mëdha të të dhënave në klluster të Elasticsearch pa hasur në vështirësi. Janë krijuar metoda gjenerike të kërkimit dhe pastaj janë implementuar forma të ndryshme të “query” që mund të përdoren për kërkim të dhënave. Për qëllime të punimit tonë është përzgjedhur të bëhet krahasimi performancës duke marrë parasysh vetëm një fushë që është teksti i tweets kështu që query janë të implementuara asisoj që e konsiderojnë vetëm tekstin e tweets si fushë për kërkim. Rezultatet të cilat kthehen nga kllusteri i Elasticsearch prezantohen nëpërmjet user interface të menaxhuar nga vue. Në vijim i është dedikuar kapitulli ELASTICSEARCH sqarimit të mekanikave të brendshme të Elasticsearch.

6.3 Integrimi i Algolia

Algolia është një search servis plotësisht i hostuar të cilit mundemi t'i qasemi nëpërmjet REST API (Representational State Transfer API). Vegla për komunikim me API të Algolias janë të ofruara në të gjitha frameworks, platforma dhe gjuhë programuese të cilat kanë rëndësi të lartë [12].

Libraria algoliasearch.js është implementuar në aplikacionin es-Algolia-performance-test për të lehtësuar komunikimin me shërbimin SaaS Algolia. Libraria algoliasearch.js njejt sikurse elasticsearch.js është shkarkuar në aplikacion nëpërmjet NPM. Është krijuar një file Algolia_config.js i cili është përdorur për të enkapsuluar librarinë algoliasearch.js në bazë të nevojave të aplikacionit tonë. Në këtë file është vendosur aplikacion key dhe search-only API key i shërbimit tonë në Algolia. Një indeks në algolia shërben për ruajtjen e të dhënave të përpunuara. Aplicacion key është vlera unike që e dallon llogarinë përkatësisht shërbimin që e marrim në Algolia ndërsa search-only API key është vlera e cila na lejon që të bëjmë kërkime vetëm në indek⁹set e llogarisë tonë.

Figura 7 Insertimi i data-set në Algolia nëpërmjet dashboard-it që ofron¹⁰

⁹Algolia api keys, <https://www.algolia.com/doc/guides/security/api-keys>

¹⁰Algolia, <https://www.algolia.com/>

Indeksimi i data-set sentiment¹⁴⁰ është bërë nëpërmjet dash-board të cilin e ofron Algolia. Nëpërmjet këtij dashboard që shihet në figurën 7 mund të shkarkohen vetëm files me madhësi maksimale 50mb me ç'rast është dashtë të bëhet ndarja e data-set në 5 files pasi që madhësia totale e data-set është 233mb. Pasi që është përfunduar indeksimi në kushte normale mund të fillojë përdorimi i API të Algolia me ç'rast do të kërkohet mbi të gjitha fushat e kërkueshme të dokumenteve. Siç është cekur paraprakisht për qëllime të punimit tonë është përzgjedhur të bëhet krahasimi performancës duke marrë parasysh vetëm një fushë që është teksti i tweets. Për të lejuar kërkimin në vetëm një kolonë duhet që të përzgjedhim fushat mbi të cilat mund të bëhet kërkimi ku në rastin tonë është përzgjedhur teksti i tweets. Në file Algolia_config.js janë implementuar forma të ndryshme për kërkimin e të dhënave. Rezultatet të cilat pranohen nga algoliasearch.js gjithashtu prezantohen nga user interface i cili është i menaxhuar nga vue. Në vijim i është dedikuar kapitulli ALGOLIA sqarimit të operimit të Algolia, mirëpo është shumë më i limituar në krahasim me atë të Elasticsearch pasi që Algolia është një closed source projekt.

7. ELASTICSEARCH

Elasticsearch është një open-source search engine e ndërtuar mbi Apache Lucene. Lucene është një full-text search-engine librari e shkruar në java. Në përdorimin e Luceneit paraqiten probleme për shkak se së fundmi është vetëm një librari. Për të punuar me të duhet të zhvillohet programi në gjuhën programuese Java dhe përpos kësaj ka kompleksitet të lartë me ç'rast nevojiten njohuri të mirëfillta në data retrieval për të punuar me të. Për t'i evituar këto probleme si zgjidhje është shfaqur Elasticsearch e cila përdor Lucene si bazë për indeksim dhe kërkim, mirëpo e bën këtë duke fshehur kompleksitetin e Lucene pas një RESTful API të thjeshtë me të cilin mund të komunikojmë nëpërmjet protokollit HTTP duke dërguar HTTP request dhe gjithashtu mund të dërgohen të dhëna shtesë në formatin JSON [11].

Që të kthehen rezultate më të sakta çdo dokumenti i caktohet një vlerë relevance në bazë të kriterit të kërimit dhe indekseve. Në kushte normale për të kalkuluar vlerën e relevancës të një dokumenti përdoret algoritmi TF-IDF. Algoritmi TF-IDF cakton vlerën e relevancës në bazë të numrit të paraqitjeve të fjalëve kyqe në dokument ndërsa i jep vlera më të larta në qoftë se ajo fjalë është më e rrallë në dokumente tjera. Elasticsearch është i personalizueshëm në bazë të nevojave të klientit si p.sh mund t'i caktohet një vlerë relevance më e lartë një dokumenti tek i cili fjalët kyqe paraqiten në titull ose mund të caktohet që përshtatja e plotë e fjalëve kyqe të vlerësohet më shumë sesa përshtatja e pjesshme, dokumentet më të reja të marrin vlerësime më të larta, të merren parasysh gabimet drejtshkimore dhe të paraqiten sugjerime etj. Në Elasticsearch mund të përdoren edhe query speciale që kthejnë rezultatet duke përdorur parashtesa, wild cards, ose regular expressions [10].

Një dallim tjetër ndërmjet Elasticsearch dhe bazave të shënimeve relacionare është kalkulimi i relevancës dhe shfaqja e dokumenteve më relevante nga Elasticsearch. Një funksionalitet tjetër që posedon është aftësia e gjenerimit të raporteve analitike të sofistikuar mbi të dhënat që janë të ruajtura në indekse nëpërmjet agregimeve [11].

Elasticsearch ofron agregime të tipit metrics dhe bucket. Metrics aggregations mundësojnë gjenerimin e statistikave të ndryshme mbi të dhënat që i mbajmë në Elasticsearch ndërsa bucket agregimet mundësojnë ndarjen e dokumenteve në përmbajtëse të ndryshme (bucket

) dhe si rezultat na I kthen të gjitha buckets dhe njëkohësisht të gjitha dokumentet që ndodhen në to [10].

Elasticsearch përdoret gjithashtu në procesimin e të dhënave që njihen si Big-data. Big-data është term që përdoret për të përshkruar një volum të madh dhe kompleks të dhënave, aplikacionet tradicionale nuk janë të përshtatshme për procesimin e këtyre të dhënave [13].

Elasticsearch mund të definohet si makinë kërkuese e shpërndarë (distributed) me real-time analitikë. Fjala elastik vjen nga aftësia e shkallëzimit (scaling) deri në qindra serverë dhe petabyte të të dhënave të strukturuar dhe të pastrukturuar.

Një ndër arsyet e popullaritetit të Elasticsearch është gjithashtu fakti që shpërndahet me Apache 2 licenca është një ndër licencat open-source më fleksibile [11].

7.1 Ruajtja e të dhënave

Për dallim prej bazave të shënimeve relacionare në Elasticsearch të dhënat nuk ruhen në tabela mirëpo në dokumente. Dokumentet ruajnë të gjitha të dhënat që kanë të bëjnë me një entitet, gjë që ndikon pozitivisht në shpejtësinë e kthimit të rezultateve pasi që nuk është nevoja të bëhet “join” për të ju qasur të gjitha të dhënave të një entiteti. Dokumentet grupohen në tipe të cilat mund t’i konsiderojmë si ekuivalentja e tabelave në databaza relacionare dhe një apo më shumë tipe mund të ndodhen në indeks ku indeksi është ekuivalentja e databazës në bazat e shënimeve relacionare.

Në rast se bëjmë një http request për të shtuar një dokument në një indeks me mapim që nuk ekziston, Elasticsearch I krijon indeksin, mapon dhe së fundmi e shton dokumentin. [10]

Për të shtuar të dhëna në Elasticsearch domosdoshmërisht nevojitet të ekzistojë indeksi në të cilin do të vendoset dokumenti. Në realitet indeksi është një namespace logjik që drejton (points) drejt një apo më shumë shards fizik [11].

Në prapavijë Elasticsearch I ndan këto indekse në shards të cilat mund të migrojnë nga serveri në server [10].

Tipet kryesore të fushave në Elasticsearch janë Core ku bëjnë pjesë numrat, strings, arrays dhe multi-fields të cilat na mundësojnë të ruajmë shumë vlera të tipit core të njëjtë në të njëjtën fushë. Gjithashtu ekziston edhe tipi I fushave predefined të cilat janë të dhëna që

plotësohen nga Elasticsearch si p.sh timestamp. Tipet nuk ekzistojnë në Lucene mirëpo vetëm janë një abstraksion të cilin Elasticsearch e përdor. Elasticsearch ofron concurrency control duke përdorur një numër të versionit për çdo dokument që të mos vijë deri tek mbishkrimi aksidental i të dhënave nga kërkesat e ndryshme të njëkohshme [10].

Veprimi i ruajtjes së të dhënave në Elasticsearch njihet si indeksim. Një dokument është ekuivalentja e një tulpe në databazë relacionare. Një dokument ka një tip të caktuar dhe ruhet brenda një indeksi. Një klluster i Elasticsearchit që zakonisht përbëhet nga një apo më shumë instanca të Elasticsearchit mund të përmbajë një apo edhe më shumë indekse që janë ekuivalentet e databazës në sistemet e menaxhimit të databazave relacionare (RDBMS) [11]. Indeksi i dokumenteve bëhet nëpërmjet procesit të analizimit ku gjatë analizimit fjalët kyqe nga teksti që po indeksohet bëhen terme të kërkimit të cilat ruhen dhe më vonë mund të përdoren gjatë kërkimit për të identifikuar dokumentet që i përmbajnë ato në ndonjë fushë specifike apo në më shumë fusha dhe gjithashtu në ndonjë renditje specifike apo arbitrare.

Në kushte normale përdoret analizuesi i cili fjalët i ndanë në terme kërkuese në bazë të ndonjë ndarësi të fjalëve siç mund të jetë hapësira e bardhë apo presja. Pastaj i kthen të gjitha shkronjat në shkronja të vogla. Ekzistojnë edhe lloje të tjera të analizuesve që ofrohen nga Elasticsearch dhe përveç asaj ekziston mundësia e ndërtimit të ndonjë analizuesi specifik të personalizuar. Procesi i indeksimit të një dokumenti fillon me dërgimin e dokumentit në një shard primare që zgjidhet në bazë të një hashi të ID-së së dokumentit. Pas kësaj dërgohet edhe në shartet replikë që ato të jenë të sinkronizuara me shartet primare dhe në rast se ndonjëra prej tyre bie nga funksioni atëherë të bëhet promovimi i një shard kopje në një shard primare. Për të kërkuar të dhëna brenda një indeksi Elasticsearch duhet të bëj atë search në të gjitha shards në të cilat është ruajtur ai indeks. Kërkimi në më shumë shards lehtëson kërkimin pasi që bëhet ndarja e ngarkesës në ato shards. Kërkimi mund të bëhet edhe në kopje pasi që ato përmbajnë të dhënat e njëjta si shards primare gjë që i bën kopjet efektive jo vetëm për evitimin e gabimeve por edhe në përmirësimin e kërkimit të dhënave [10].

Dokumentet në Elasticsearch karakterizohen me faktin që një dokument përmban emrat e fushave dhe vlerat e tyre, një dokument mund të jetë hierarkik. Nga kjo nënkuptojmë se vlera e një fushe mund të jetë një string mirëpo në raste tjera mund të përmbajë një dokument tjetër (emrat e fushave dhe vlerat e atij dokumenti) dhe ka strukturë fleksibile apo me fjalë tjera

skema e saj nuk është e predefinuar. Pra në qoftë se në të ardhmen paraqitet nevoja të ruhen edhe vlera të reja atëherë vetëm duhet të dërgohet emri i fushës së re dhe vlera e saj. Në tipe bëhet definimi i kolonave dhe ky proces njihet si mapim. Dokumentet nuk janë të obliguara që t'i kenë të gjitha kolonat dhe në rast se në ndonjë dokument shtohet ndonjë kolonë e cila nuk është e definuar paraprakisht në tip atëherë edhe për atë kolonë automatikisht bëhet mapim në atë tip. Tipi I të dhënave që ruhen në këtë kolonë automatikisht caktohet nga Elasticsearch varësisht nga vlera që kemi dhënë mirëpo kjo mund të qojë deri tek disa gabime ku nëpërmjet mapimit automatik në disa raste Elasticsearch gabon tipin e të dhënës. Indekset janë kontejner për mapimet e tipeve. Një indeks në Elasticsearch është një grup i dokumenteve që ruhet në disk. Aty ruhen të gjitha fushat nga të gjitha tipet e mapuara dhe settings të saja. Indekset e rinjë që krijohen në Elasticsearch përbëhen nga pesë shards dhe pesë kopje të tyre. Shard është një folder ku ndodhen fajlla në të cilat janë ruajtur të dhënat e indekseve. Një shard është njësia më e vogël të cilën Elasticsearch mund të bart nga node në node [10].

7.2 Analizimi në Elasticsearch

Analiza është procesi të cilin Elasticsearch e performon mbi përmbajtjen e një dokumenti para se të shtohet në inverted indeks. Procesi i analizimit fillon me filtrimin e karaktereve ku bëhet transformimi i karaktereve në karaktere më të përshtatshëm për kërkim si p.sh & shndërrohet në and. Pas kësaj bëhet ndarja e përmbajtjes në një apo më shumë pjesë më të vogla që quhen token. Lucene është përgjegjës për krijimin e tokenëve duke bërë ndarjen e tekstit në një numër të tokenëve që mund të jetë edhe zero. Këta token transformohen duke përdorur token filter ku një ndër më të famshmit dhe njëkohësisht më të përdorurit është lowercase filter që i transformon tokenat dhe së fundmi tokenat e përfituara dërgohen në Lucene për tu indeksuar për atë dokument.

Edhe gjatë kërkimit në Elasticsearch duke përdorur disa query specifike vjen deri tek analizimi i përmbajtjes që kërkohet. Elasticsearch mundëson edhe përdorimin e analizuesve të personalizuar. Analyzer API lehtëson krijimin e analizuesve të personalizuar duke na dhënë mundësi për të testuar analizuesit që po krijojmë. Token analizuesi i thjeshtë bën ndarjen e

përmbajtjes në tokena dhe automatikisht e shndërron çdo shkronjë të shkruar të madhe në shkronjë të shkruar të vogël. Token analizuesi hapësirë e zbrazët bën ndarjen e tekstit në token në karaktere që janë hapësirë e zbrazët. Stop analizuesi I largon fjalët që nuk kanë ndonjë vlerë (p.sh fjalët lidhëse) nga përmbajtja e tekstit. Analizuesi patern mundëson ndarjen e tekstit në një patern të cilin ne ja caktojmë. Elasticsearch ofron edhe analizues për gjuhë specifike. Përveç analizuesve Elasticsearch ofron edhe tokenizues për shërbime të ndryshme. Standard tokenizuesi bën segmentimin e tekstit unicode ku gjatësia maksimale e tokenit mund të jetë 255 karaktere pa përfshirë shenja të pikësimit pasi që ato largohen nga ky tokenizues. Tokenizuesi fjalë bën ndarjen e tekstit në çdo karakter jo-shkronjë. Tokenizuesi hapësirë e zbrazët ndan tekstin në tokene kur has në karakter të hapësirës së zbrazët. Ekziston mundësia që të bëhet tokenizimi në bazë të ndonjë paterni të caktuar nëpërmjet tokenizuesit patern. Elasticsearch gjithashtu ofron disa token filter të gatshëm ku ndër më të përdorurit janë Token filteri standard, shkronja të vogla, gjatësi, stop etj. Token filteri standard nuk bën asnjë ndryshim në karakteret e atij tokeni. Token filteri shkronja të vogla bën shndërrimin e të gjitha shkronjave të mëdha në një tekst në shkronja të vogla. Nëpërmjet token filterit gjatësi mund të lejohen të futen vetëm tokenat tek të cilët numri I karaktereve është në mes të një vlere minimale dhe maksimale të karaktereve [10].

7.3 Scoring në Elasticsearch

Procesi I përcaktimit të relevancës së një dokumenti ndaj një query të caktuar quhet scoring. Elasticsearch bën caktimin e score për një dokument duke përdorur frekuencën e paraqitjes të fjalëve kyqe të kërkuara në dokument. Pra sa më shumë paraqitet një fjalë kyqe e kërkuar në dokument aq më e madhe është relevanca e atij dokumenti mirëpo Elasticsearch analizon edhe se sa e shpeshtë është kjo fjalë në dokumente tjera me ç'rast në qoftë se është e përhapur shumë në dokumente tjera atëherë kjo e ulë relevancën e asaj fjale. Kjo quhet TF-IDF ku TF qëndron për term-frequency dhe paraqet shpeshtësinë e një termi brenda atij dokumenti ndërsa IDF që qëndron për inverted document frequency tregon shpeshtësinë e atij termi në dokumentet tjera. Termi invers në IDF rrjedh nga fakti që frekuenca e termit shumëzohet me

vlerën inverse të frekuencës që ai term është paraqitur në dokumente tjera. Score I një dokumenti përcaktohet në Lucene përkatësisht në Elasticsearch nëpërmjet formulës:

$$score_{query,document} = \sum_t^q \sqrt{TF_{t,d}} * IDF_{t,d}^2 * norm(d, field) * boost(t) ..(1)$$

$score_{query,document}$ – paraqet vlerën score për një dokument të caktuar për një query specifik.

$TF_{t,d}$ – tregon se sa shpesh është paraqitur një term i caktuar në dokument.

$IDF_{t,d}^2$ – paraqet vlerën inverse të numrit të paraqitjeve të termit në dokumentet tjera në fuqi katrore.

$norm(d, field)$ – paraqet faktorin normalizues të një fushe të dokumentit

$boost(t)$ – paraqet shtyerjen për atë term

Score I një dokumenti d për një query q është shuma (për secilin term) e rrënjës katrore të frekuencës të termit në dokumentin d shumëzuar me frekuencën inverse të dokumentit në fuqi katrore shumëzuar me faktorin normalizues dhe shtytjen për atë term. Ekzistojnë edhe modele tjera të përcaktimit të score të një dokumenti siç janë:

- Okapi BM25
- Divergjenca nga rastësia DFR(Divergence from randomness)
- Bazuar në informacione
- Ngjashmëritë LM Dirichlet
- Ngjashmëritë LM Jelinek Mercer

Shtyrja(Boosting) është proces gjatë të cilit modifikohet relevanca e një dokumenti. Elasticsearch ofron mundësinë e shtyrjes të një dokumenti ose gjatë indeksimit ose gjatë kërkimit. Në përgjithësi preferohet që të aplikohet shtyrja gjatë një query (gjatë kërkimit) pasi që është më fleksibil sesa shtyrja gjatë indeksimit e cila na obligon të bëjmë ri-indeksim të dokumentit sa herë që dëshirojmë t'i ndryshojmë shtyrjen dokumenteve. Shtyrja të cilën e marrin fushat në të cilat kërkohet në fund normalizohet nga Elasticsearch, për këtë arsye

duhet të jepen vlera të ndryshme për fusha të ndryshme që të ketë efekt shtyrja e të mos vijë deri tek normalizimi i shtyrjes për të gjitha fushat në vlerën një. Kalkulimi i score të një dokumenti mund të analizohet në Elasticsearch nëpërmjet parametrin `explain` ku në qoftë se dërgohet edhe ky parametër si rezultat kthehet mënyra e kalkulimit të score të dokumenteve apo mund ta përdorim për të analizuar pse ndonjë dokument specifik nuk është kthyer si rezultat nga një query. Nëpërmjet përdorimit të `function_score` query mund të përcillen së bashku me kërkesën e kërimit disa funksione në bazë të cilave do të modifikohet score i dokumenteve rezultuese. Modifikimi i scoring të dokumenteve mund të bëhet edhe nëpërmjet përdorimit të skripteve të cilat shkruhen në gjuhën Groovy. Elasticsearch ofron `random_score` funksionin të cilit i jepet një parametër `seed` në bazë të cilit gjenerohen score të rëndomta për dokumentet gjë që na mundëson të kthejmë rezultate të ndryshme gjatë kërkimeve përderisa ndryshojmë vlerën `seed`. Decay Functions mundësojnë të kthehen rezultate të cilat janë më aktuale apo mund të përdoren edhe në geolocation data për të kthyer rezultatet të cilat janë më afër pozitës nga bëhet kërkesa [10].

7.4 Menaxhimi i dokumenteve

Kur krijojmë një dokument të ri ai ruhet në një shard primarë. Na nevojitet të dimë se në cilin shard do të ruhet ai dokument për ta përdorur në të ardhmen për këtë arsye shard primarë përzgjidhet duket përdorur formulën:

$$shard = hash(routing) \% number_of_primary_shards \dots\dots\dots (2)$$

shard – shard tek i cili do të ruhet dokumenti.

hash – funksion i cili i mapon të dhënat me madhësi arbitrare në të dhëna me madhësi të fiksuar.

routing – vlera që përdoret për të definuar se në cilin shard duhet të ndodhet dokumenti.

number_of_primary_shards – paraqet shumën e shards primare.

Vlera routing është një string arbitrar e cila në kushte normale është vlera e `_id` mirëpo mundet edhe të personalizohet nga ana jonë t’i korrespondojë ndonjë vlere tjetër. Ky string i jepet si parametër një funksioni i cili do ta hashirat për të gjeneruar një numër, i cili modulohet me numrin e shardeve primarë dhe si rezultat na jep numrin e shardit primarë tek i cili do të ruhet ky dokument. Për këtë arsye caktimi i numrit të shardeve primarë duhet të bëhet gjatë krijimit të indeksit sepse po të ndryshonim numrin e shardave primarë atëherë të gjitha vlerat e routing paraprak nuk do të ishin më valide për gjetjen e dokumenteve. Në qoftë se kemi disa nodes në një klluster kërkesa mund të bëhet në cilën do prej tyre pasi që shards kanë dijeni se cilat dokumente në cilat nodes ndodhen, atëherë ato e përcjellin kërkesën tek node i cili i ka të dhënat adekuate. Në rastet e dërgimit të kërkesës për krijim, indeksim apo fshirje të një dokumenti në shard primar dhe replikat e tij, së pari nodei tek i cili bëhet kërkesa përdorë `_id` e dokumentit për të identifikuar se në cilin shard primarë ndodhet ai dokument. Në qoftë se nuk e përmban vetë atë dokument atëherë e përcjell (forward) kërkesën tek node i cili përmban shardin primarë në të cilin ndodhet ky dokument. Në momentin kur kryhen veprimet me dokumentin në shard primarë nodei tek i cili ndodhet shardi primarë e përcjell kërkesën tek të gjitha nodes që përmbajnë replika të këtij shardi. Pasi që të përfundohet veprimi në shards kopje ato e përcjellin një mesazh suksesi tek node i cili i ka bërë kërkesën dhe në fund kthehet një përgjigje tek klienti mbi rezultatin e kërkesës së bërë nëpërmjet nodeit të cilit klienti e ka bërë kërkesën. Ekziston mundësia të bëjmë ndryshime në konfigurimet e Elasticsearchit për të rritur performancën duke ulur sigurinë e të dhënave si p.sh mund të bëjmë kërkesën për përcjelljen e ndryshimit të dhënave të ekzekutohet në mënyrë asinkrone në shard replika ku pra nuk pritet të përfundohet dhe të konfirmohet ndryshimi në shard replika mirëpo në momentin kur ndryshimi aplikohet në shard primarë menjëherë informohet klienti mbi suksesin e ndryshimit [11].

Figura 8 Krijimi, indeksimi, fshirja e dokumenteve në Elasticsearch

Në figurën 8 shihet procesi i krijimit, indeksimit dhe fshirjes të dokumenteve në elasticsearch të cilat janë përshkruar më detajisht në vijim. Azhurimi (update) i pjesshëm i një dokumenti fillon me kërkesën të cilën e dërgon klienti në ndonjërin prej nodave të kllusterit pas kësaj në qoftë se ai node nuk e përmban shardin i cili përmbanë atë dokument atëherë e përcjell kërkesën në nodein i cili e përmban shardin primarë që e përmban atë dokument. Ajo node pastaj e gjen atë dokument dhe e ndryshon përmbajtjen në fushën `_source` që është në formatin JSON dhe provon ta riindeksojë atë dokument në shard primarë. Në qoftë se dokumenti është ndërruar nga ndonjë proces tjetër atëherë provon përsëri për ta ndryshuar deri sa të përfundohet azhurimi apo deri sa të tejkalohet limiti i tentimeve që varet nga vlera e fushës `retry_on_conflict` me ç'rast heq dorë nga ai ndryshim. Në qoftë se node ka arritur të bëjë azhurimin (update) në dokument me sukses atëherë e përcjellë (forward) paralelisht versionin e ri të dokumentit tek të gjitha nodat që përmbajnë shards replikë në të cilat ndodhet ky dokument që të ri indeksohen. Pasi që të gjitha shards të raportojnë sukses tek node paraprak rezultati përcjelllet në node tek e cila është bërë kërkesa fillestare dhe së fundmi kthehet rezultati i ndryshimit tek klienti. Në rastet kur shardi primarë e përcjell (forward) ndryshimin tek shard replikë, nuk e përsërit kërkesën mirëpo e përcjell versionin e ri të atij dokumenti. Në rastet kur bëjmë kërkesë nëpërmjet mget dhe bulk API në ndonjë node ajo node e ndërton nga një multi-get kërkesë për secilin node e cila përmban shards ku ndodhen dokumente që po kërkohen. Së fundmi kthehen rezultatet nga të gjitha nodes tek nodei fillestar i cili pastaj i përmbledh dhe i kthen si përgjigjje ndaj bërësit të kërkesës [11].

Fshirja e dokumenteve mund të bëhet nëpërmjet ID të tyre. Mund të fshihen më shumë dokumente me një kërkesë duke bërë një bulk request ku shpejtësia e fshirjes është më e madhe sesa fshirja e dokumenteve me kërkesa të njëpasnjëshme. Elasticsearch ofron mundësinë e fshirjes së një tipi të mapimit dhe të gjitha dokumentet që I takojnë atij tipi dhe gjithashtu mund të bëjmë fshirjen e të gjitha dokumenteve që kthehen si rezultat I një query. Fshirja e indekseve është më e shpejtë sesa fshirja e dokumenteve individuale pasi që dokumentet në fakt markohen si të fshira dhe vetëm kur ndodh procesi I gërshetimit(merging) ato fshihen. Gërshetimi (merging) është procesi I kombinimit të disa Lucene segmente të vogla në një segment më të madh. Një segment është një copë e një Lucene Indeksi që krijohet kur indeksojmë. Me indeksimin e dokumenteve të reja krijohen edhe segmente të

reja. Që të mos dal jashtë kontrollit numri I segmenteve Lucene bën ngjitjen (merging) të segmenteve duke përjashtuar dokumentet që janë fshirë [10].

7.5 Kërkimi i të dhënave

Kërkimet në Elasticsearch mund të bëhen nëpërmjet RESTful API duke i dërguar HTTP request. Elasticsearch vë në dispozicion një query language që quhet domain specific language. DSL është gjuhë fleksibile e cila në mundëson ndërtimin e queryve të komplikuar nëpërmjet request body duke përdorur formatin JSON [11].

Elasticsearch ofron funksionalitet më të lartë se Lucene nga cacheing e deri te rezultate analitike në kohë reale. Kërkimi mund të bëhet në një indeks të vetëm, në disa indekse apo të kërkohet në të gjithë indekset së bashku. Field data cache është një cache në memorie që Elasticsearch e përdor për të ofruar kërkim më të shpejtë. Në kushte normale kjo ndërtohet herën e parë që bëhet ndonjë kërkim gjë që në raste kur kemi shumë të dhëna mund të ngadalësojë kërkimin e parë për këtë arsye ekzistojnë warmers të cilët ekzekutohen automatikisht dhe sigurohen që është bërë ngarkimi i field data cache me të dhëna të cilat do të përdoren në query. Kërkimi në Elasticsearch mund të bëhet si URI request ku të gjitha parametrat së bashku me fjalët që po kërkohen shkruhen si URL, mirëpo mund të bëjmë edhe kërkime duke dërguar disa parametra si json objekte. Gjatë kërkimit mund të përdoren edhe filter të cilët nuk marrin parasysh relevancën mirëpo vetëm a përmbush kriteret e kërkimit. Nëpërmjet përdorimit të agregimeve në search mund të gjenerohen statistika të ndryshme mbi të dhënat tona. Gjatë kërkimit duhet të kemi parasysh tre aspekte thelbësorë që janë ku të kërkojmë, çfarë përmbajtje do të kthehet si rezultat dhe si të kërkojmë. Kërkimi mund të bëhet në një tip apo më shumë tipe të një apo më shumë indekseve. Për të kërkuar në më shumë se një tip duhet që të shkruhen emrat e tipave dhe të ndahen me presje. E njëjta vlen për kërkimin në më shumë se një indeks ku shënohen emrat e indekseve dhe ndahen me presje ose përdoret si vlerë indeksi “_all” që e bën kërkimin në të gjitha indekset. Në të dhënat që kthehen si rezultat në kushte normale ndodhen të dhëna mbi kohën e procesimit të kërkesës, numrin e shards që janë përfshirë në këtë kërkim dhe dokumentet që I përshtaten kërkimit së bashku me të dhëna deskriptive mbi to siç është relevanca. Elasticsearch

mundëson edhe përdorimin e filterave për të kërkuar të dhënat. Dallimi kryesor në mes të query dhe filterave është që në rast të kërimit duke përdorur filterat kthehen dokumentet që e përmbushin kushtin pa u bërë llogaritja e relevancës së tyre gjë që kursen shumë resurse. Rastet më të shpeshta të përdorimit të filterave janë në Filtered Query e cila përbëhet nga dy komponentë Query dhe Filter. Në prapavijë Elasticsearch e krijon një bitset që tregon se cilat prej dokumenteve e kalojnë kushtin apo kushtet e filterit. Me ç'rast Elasticsearch përdor këtë bitset për të ditur se cilat dokumente t'i përjashtojë nga kërkitimi me query. Në këtë rast eliminohen një pjesë e dokumenteve për të cilat më nuk duhet të bëhen llogaritje të mëtutjeshme siç është relevanca. Elasticsearch ofron edhe mundësinë e cashiratjës të bitset të filterit për përdorim të mëtutjeshëm që të mos llogaritet bitseti çdo herë që bëhet një kërkesë e re [10].

Nëpërmjet përdorimit të query “match_all” mund të kërkojmë që të kthehen si rezultat të gjitha dokumentet. Kërkimet mund të bëhen edhe nëpërmjet term queries dhe term filters. Tek kërkitimi nëpërmjet term queries specifikohet një fushë e caktuar në të cilën do të kërkohet termi specifik. Pasi që termet që kërkohen nuk janë të analizuara duhet të ketë përputhje ekzakte në mes të termit të kërkuar dhe termit në atë fushë në dokument. Term filters mund të përdoren për kërkitim siç përdoren edhe term queries mirëpo avantazhi i term filters është që mund të përdoren pa afektuar vlerësimin. Ekziston edhe terms query që mundëson kërkitimin e më shumë termeve në një dokument. Match query mundëson kërkitimin e dokumenteve tek të cilat përmbahet teksti i kërkuar në një fushë të caktuar apo kërkitimi mund të bëhet edhe në të gjitha fushat. Phrase_prefix query mund të përdoren për kërkitimin e dokumenteve që përmbajnë një tekst që fillon me një prefiks specifik. Multi_match queries mundësojnë kërkitimin e termave në më shumë fusha. Query bool mundëson kombinimin e disa query në një query të vetme e cila cakton se cilat kushte duhet të përmbushen nëpërmjet pjesës “must”, cilat kushte duhen të përmbushen pjesërisht nëpërmjet pjesës “should” dhe cilat kushte nuk duhet të përmbushen nëpërmjet pjesës “must_not”. Ekzistojnë edhe bool filters që veprojnë identik me bool queries mirëpo për dallim nga bool queries nëpërmjet bool filters mund të bëjmë kombinimin e disa filterave në një filter të vetëm. Range query dhe filter shërbejnë për gjetjen e dokumenteve të cilat ndodhen në mes një intervali të caktuar.

Prefix query dhe filter mundëson kërkimin e dokumenteve ku fusha e kërkuar fillon me prefix të dhënë. Wildcard query mundësojnë kërkimin e dokumenteve duke përdorur wildcards. Exists Filter shërbejnë për kërkim të dokumenteve të cilat kanë ndonjë vlerë në një fushë specifike. Missing Filter mund të përdoret për të gjetur të gjitha dokumentet tek të cilat mungon vlera apo ku vlera është vlera default e cila është paracaktuar në mapim [10].

Figura 9 Kërkimi nëpërmjet query

Për dallim nga operacionet CRUD (create, read, update, delete) kërkimi është më kompleks pasi që nuk e dimë cilat dokumente do t'i përshtaten kërkimit me ç'rast nuk mundemi të dimë se në cilat shards specifike do të gjejmë dokumente që do të përshtaten. Kërkesat e kërkimit duhet që të përcillen në secilën shard të indeksit apo indekseve në të cilat po kërkojmë dokumente për të siguruar që të gjitha dokumentet që i përshtaten kërkesës së kërkimit të

kthehen si rezultat. Kërkimi është një proces me dy faza të cilat janë query dhe fetch ku në fazën e parë bëhet kërkimi për dokumente në të gjitha shards të indeksit apo indekseve ndërsa në fazën e dytë bëhet përmbledhja e rezultateve nga të gjitha shards dhe kthehen si rezultat nga nodei fillestar. Procesi i kërkimit nëpërmjet një query në elasticsearch që shihet në figurën 9 është sqaruar në vijim. Gjatë fazës query, kërkesa transmetohet tek të gjitha shards të një indeksi. Secila shard e ekzekuton kërkesën lokalisht dhe e ndërton një queue prioriteti me dokumente që ju kanë përgjigjur kërkesës. Sikurse get requests edhe search requests mund të kryhen nëpërmjet shards primare apo shards replika. Node koordinues përdor algoritmin round robin për të përzgjedhur shardin i cili do të përpunojë kërkesën për të shpërndarë ngarkesën në mes të nodes. Kur këto shards e bëjnë kërkimin lokalisht secila prej tyre e ndërton një priority queue i cili mund të përmbajë aq elemente saqë të jetë në gjendje të përmbush kërkesën vetë. Tek nodei koordinues kthen si rezultat një list me pak të dhëna siç janë id e dokumentit dhe ndonjë vlerë që nevojitet për sortim të tyre si `_score`. Node koordinues i bashkon këto rezultate në priority queue të vetin që paraqet rezultatin e kërkimit. Gjatë fazës query pra u bë identifikimi i dokumenteve të cilat i plotësojnë kushtet e kërkesës, mirëpo duhet edhe të kthehen dokumentet si rezultat. Kjo arrihet nëpërmjet fazës fetch ku node koordinuese i identifikon dokumentet që duhet të kthehen si rezultat dhe pastaj e bën një multi-get kërkesë tek shards të cilat i përmbajnë ato dokumente. Secila shard i ngarkon dokumentet dhe pastaj i kthen ato tek node koordinues. Pasi të arrijnë të gjitha dokumentet te node koordinues ato përcillen te klienti nga pikërisht kjo node. Marrja e dokumenteve si rezultat kërkimi mund të bëhet prej shards primare apo replikave. Një kërkesë për marrjen e dokumenteve si rezultat kërkimi mbi një node që ndodhet në klluster të Elasticsearch së pari do të fillonte me kërkesën që e bën klienti mbi cilëndo node me ç'rast node e përdorë `_id` e atij dokumenti për të përcaktuar se në cilin shard ndodhet ai dokument. Pasi të përcaktohen shartet primare dhe replikat në të cilat ndodhet ai dokument duke përdorur algoritmin round robin bëhet përzgjedhja se prej cilit shard kopje do të mirëtohet rezultati. Kjo mënyrë e përzgjedhjes së shardit mundëson një shpërndarje të mirëfilltë të ngarkesës. Pas kësaj rezultati përcjellet nëpër nodes nga ka ardhur kërkesa dhe së fundmi i kthehet si rezultat klientit [11].

Algoritmi round-robin mund të shkaktojë probleme në qoftë se nuk kanë performancë të njejtë të gjitha shards, në raste të tilla duhet të bëhet konfigurimi që të evitohen shards me performancë më të vogël [10].

7.6 Shkallëzimi (scaling) në Elasticsearch

Në rast se nevojitet rritja e resurseve që përdoren për t'i përmbushur të gjitha kërkesat e klientëve atëherë Elasticsearch ofron zgjidhje të lehtë nëpërmjet zgjerimit horizontal nga edhe rrjedh fjala elastik në emrin e Elasticsearch [10].

Elasticsearch është e dizajnuar që gjithmonë të mund t'i përballojë kërkesat nëpërmjet vertical scaling apo horizontal scaling. Vertical scaling arrihet kur serverit në të cilin ndodhet instanca e Elasticsearchit i shtohen resurset kompjuterike ndërsa horizontal scaling arrihet kur e shtojmë numrin e serverëve në të cilët po operon Elasticsearch ku të gjitha këto instanca së bashku ndodhen dhe formojnë të njëjtin klluster. Scaling Horizontal në shumicën e aplikacioneve din të paraqitet problem pasi që aplikacionet duhet të modifikohen që të mund të operojnë dhe t'i shfrytëzojnë resurset në mënyre efektive prej më shumë se një serveri. Për dallim nga këto aplikacione Elasticsearch është e distribuuar (distributed) në kushte normale pra din se si t'i shfrytëzojë këto instanca të reja të Elasticsearch duke i përdorur resurset e serverëve në të cilët operojnë këto instanca të reja [11].

Kllusteri më i vogël përmban një node (Instance të Elasticsearch) e cila i menaxhon të gjitha shards. Me shtimin e nodave të reja në atë klluster shartet shpërndahen në mënyrë të barabartë edhe në nodes tjera. Shtimi i nodave të reja në një klluster njihet si scaling(shkallëzim) horizontal. Në rastin kur shtojmë resurse siç janë CPU apo RAM te ndonjë node ekzistues për të akomoduar numrin më të madh të kërkesave atëherë kemi bërë scaling(shkallëzim) vertical. Aplikim praktik i Elasticsearch mund të jetë si bazë e shënimeve primare dhe e vetme ku do të shkruhen të gjitha të dhënat mirëpo në raste kur kemi nevoja tjera specifike që nuk mund të përmbushen vetëm nga Elasticsearch atëherë mund ta përdorim së bashku me baza të shënimeve tjera [10].

7.7 Klluster në Elasticsearch

Një Instance të Elasticsearch I referohemi si node. Në server mund të niset një apo më shumë instanca të Elasticsearch pra një apo më shumë nodes. Kushtet minimale për të formuar një klluster janë të krijohen Instanca të cilat kanë emër të kllusterit të njejtë dhe settings default. Efektiviteti I përdorimit të kllusterave qëndron në shpërndarjen e të dhënave në disa servera gjë që I ofron më shumë resurse Elasticsearch-it gjatë kërkimit. Konfigurimi mbi emrin e kllusterit ndodhet në Elasticsearch.yml . Një node I ri I zbulon kllusterat ekzistues nëpërmjet multicast [10].

Klluster përbëhet prej një apo më shumë nodes me emër të kllusterit të njejtë. Të gjitha këto nodes që ndodhen brenda kllusterit të njejtë shpërndajnë të dhënat dhe ngarkesën e tyre ndërmjet veti. Me shtimin dhe largimin e nodes kllusteri e riorganizon vetën për të siguruar shpërndarjen e të dhënave. Në një klluster përzgjidhet një node të jetë master node e cila është përgjegjëse për menaxhimin e eventeve që përfshijnë të gjitha nodes në klluster siç është krijimi apo fshirja e një indeksi, shtimi apo largimi i një node nga kllusteri, mirëpo nuk nevojitet të përfshihet me menaxhimin e kërkesave për kërkim apo ndryshime të dokumenteve. Requestat për kërkim mund të bëhen ndaj cilës do node pasi që ato janë në dijeni se në cilën node ndodhen dokumentet dhe pastaj ato mund të përcjellin kërkesën tek node i cili përmban ato të dhëna. Mbledhjen e të dhënave dhe shndërrimin e tyre në një response e bën node tek i cili kemi bërë kërkesën. Në rast se brenda kllusterit nodei master bie nga funksioni atëherë menjëherë bëhet ngritja në detyrë e një nodei të thjeshtë në node master. Në qoftë se me rënien nga funksioni të nodeit janë humbur edhe disa shards primare atëherë veprimi i parë që do të bëjë node master në qoftë se ekzistojnë replika të shardave të humbura është t'i promovojë shartet replikë në shards primare [11].

Për të vendosur node-in e dytë në klluster mjafton që të fillojë ekzekutimi I Elasticsearch në një server tjetër që komunikon me serverin ku ndodhet node I parë. Instanca e dytë e detekton instancën e parë nëpërmjet multicast dhe I bashkangjitet kllusterit. Node I parë është master I kllusterit me ç'rast mban të dhëna mbi nodat të cilat ndodhen në klluster dhe shards të tyre. Këto informacione quhen gjendja e kllusterit (cluster state) dhe replikohen në nodes tjera.

7.8 Shards në Elasticsearch

Një shard është një njësi punuese e cila përmban një pjesë të dhënave të një indeksi. Një shard është një instancë e Lucene dhe është një search engine në vete. Dokumentet ruhen dhe kërkohen nëpër shards mirëpo aplikacionet tona nuk komunikojnë me shards direkt mirëpo komunikojnë me indekse. Në rast se ndryshon numri i nodes në klluster, në mënyrë automatike bëhet migrimi i shards në nodes për të arritur balancimin e kllusterit. Një shard mund të jetë shard primarë dhe replikë. Secili dokument i takon dhe ruhet në një shard primarë. Një shard replikë është kopje e shardit primar të cilat përdoren për redundancën të cilën e ofron në rast të dështimit në hardware dhe për të rritur numrin e kërkesave të njëkohshme që mund t'i shërbejë Elasticsearch. Numri i shards primare caktohet në momentin kur krijohet një indeks ndërsa numri i shards replikë mund të ndryshohet gjatë gjithë kohës [11].

Njësia më e vogël e të dhënave me të cilën mirëpo Elasticsearch është një shard që në Lucene njihet si indeks. Një indeks i Lucene është një folder me files që përmban një inverted indeks. Inverted indeks është strukturë e të dhënave e cila na mundëson të bëjmë kërkime se në cilat dokumente ndodhen fjalët kyçe që po i kërkojmë pa i shfletuar dokumentet. Lucene indeksi ruan të gjitha të dhënat e dokumenteve dhe informacione ekstra si fjalorin e termeve dhe shpeshtësinë e termeve. Fjalori i termave përdoret për të gjetur se në cilat dokumente ndodhet ndonjë fjalë mirëpo pa i shfletuar ato dokumente ndërsa shpeshtësia e termave përdoret për caktimin e vlerës së relevancës të një dokumentit varësisht nga shpeshtësia e paraqitjes të fjalëve kyçe në dokument [10].

Në rast se kemi një indeks që është ndarë në 4 shards. Shpejtësia e procesimit të kërkesës në atë indeks rritet me shtimin e nodave të rinjë përderisa nuk tejkalohet numri i shards në të cilat është ndarë ai indeks. Kur vendosim 4 nodes në klluster secila prej tyre merr një shard të atij indeksi, mirëpo me vendosjen e nodesave tjera nuk kemi ndonjë ndryshim pasi që nuk mund të zhvendosen shards tjera në to. Me ç'rast në runtime mund të rrisim numrin e shards kopje mirëpo kjo vetëm do të rriste numrin e kërkesave të njëkohshme që mund të procesohen nga Elasticsearch.

Për të bërë përmirësime të mëtutjeshme në procesimin e kërkesave të kërkimit duhet që të ndërprehet ekzekutimi i Elasticsearch dhe të rritet numri i shards primare për atë indeks, gjë që nuk mund të bëhet në runtime. Me ndarjen e indeksit në më shumë shards vjen deri tek zvogëlimi i numrit të dokumenteve që ndodhen në shard. Pasi që shard është njësia më e vogël me të cilën punon një node. Krijimi i më shumë shards mundëson ndarjen e atyre të dhënave në grupe më të vogla të cilat mund të shpërndahen në më shumë nodes [10].

7.9 Trajtimi i problemeve në performancë nga Elasticsearch

Për kërkim full-text databazat tradicionale nuk kanë demonstruar performanca të mira pasi që në to ruhet një vlerë e vetme për një fushë. Për të bërë kërkime full-text në kohe reale nevojitet që secili term i cili e përbën tekstin të jetë i indeksuar. Struktura e të dhënave më e përshtatshme për qëllime të tilla është inverted index i cili përmban të dhëna të sortuara mbi vlerat unike të termeve të cilat shfaqen në dokumente dhe për secilin term e përmban një listë të dokumenteve në të cilat ndodhet ai term. Inverted indexes përmbajnë edhe informata të tjera mbi dokumentet siç janë numri i paraqitjeve të një termi në një dokument, renditja e termeve në një dokument, gjatësia e secilit dokument, gjatësia mesatare e dokumenteve etj. Këto të dhëna mund të përdoren nga Elasticsearch për të përcaktuar se cilat dokumente janë më relevante gjatë një request të kërkimit. Që Elasticsearch të jetë plotësisht funksional duhet që inverted indeksi t'i përmbajë të gjitha dokumentet të cilat dëshirojmë që të shfaqën si rezultate të kërkimit. Në kohën fillestare të kërkimeve full-text, një inverted indeks është ndërtuar për të gjitha dokumentet dhe është shkruar në disk prej nga është përdorur derisa nuk është formuar ndonjë indeks i ri për shkak të ndryshimeve që janë bërë në dokumente me ç'rast indeksi i ri e ka zëvendësuar indeksin e vjetër. Inverted indekset janë të pandryshueshme (immutable) dhe ky fakt i bën që të kenë disa avantazhe. Pasi që inverted indekset janë të pandryshueshme ato nuk duhet të mbyllen deri sa ndonjë proces të përfundojë ndryshimin që ka pasur për të bërë. Në momentin kur indeksi lexohet në filesystem të kernelit vendoset në cache. Përderisa ka hapësirë në cache ajo gjithmonë qëndron aty dhe qasja në të dhëna shpejtohet pasi që nuk është nevoja të mirën të dhëna nga disku mirëpo direkt nga cache. Në cache gjithashtu ruhen edhe lista të tjera që janë gjeneruar gjatë kërkimeve

specifike pasi që inverted indeksi nuk ndryshon, atëherë edhe këto kërkime duhet të kthejnë rezultatin e njejtë. Fakti që inverted indeksi i përmban të gjitha të dhënat, e bën të madh në hapësirë memorike, gjë që mundëson kompresimin e të dhënave duke reduktuar dukshëm resurset kompjuterike që shpenzohen. Anët negative të një inverted indeks i cili është i pandryshueshëm janë pamundësia e shtimit të dokumenteve të reja pa rindërtuar indeksin. Në qoftë se azhurimi i indeksit është shumë i shpeshtë kjo mund të ndikojë negativisht në resurset kompjuterike që shpenzohen. Për të zgjidhur këtë problem përdoren indice të cilat mund të azhurohen (update) në mënyrë dinamike. Në vend se të rishkruhet inverted indeksi, thjesht shtohen indekse shtesë të cilat përmbajnë ndryshimet më aktuale. Kërkimi fillon nga indekset më të vjetra e vazhdon në ato më të reja dhe së fundmi rezultatet kombinohen. Lucene ofron konceptin e kërkimit per-segment. Një segment është një inverted indeks. Fjala indeks në Lucene paraqet një koleksion të segmenteve dhe një commit point që është një file që liston të gjitha ato segmente. Dokumentet e reja së pari shtohen në një buffer indeksues në memorie para se të shkruhen në ndonjë segment në disk. Pra një pre-segment search fillon me mbledhjen e dokumenteve të reja në buffer indeksues në memorie. Kur të ndodh commit i buferit një segment i ri plotësues ruhet në disk ku së bashku me atë shkruhet një commit point në disk i cili referencon emrin e segmentit plotësues të ri. Së fundmi kur ndodh fsync të gjitha shkrimet që po presin në cache të filesystemit shkruhen në disk për të siguruar që ato janë shkruar fizikisht. Segmenti i ri hapet dhe dokumentet e reja që i përmban tani janë të kërkueshme ndërsa buferi pasi që është boshatisur është i gatshëm të pranojë dokumente të reja. Në këtë mënyrë dokumentet e reja mund të shtohen pa pasur shpenzime të panevojshme të resurseve kompjuterike. Deletes dhe updates funksionojnë ngjashëm pasi që përsëri inverted indeksi është i pandryshueshëm dhe gjithashtu inverted indekset plotësuese janë të pandryshueshme për këtë arsye nuk mund të ndryshohen apo të fshihen dokumentet direkt në to. Në rastin e fshirjes dokumenti në të vërtetë vetëm markohet si i fshirë në file-in .del . Dokumentet e tilla të markuara mund të i plotësojnë kushtet që të përzgjidhen si rezultat i ndonjë kërkese mirëpo ato largohen nga rezultati para se të kthehet te klienti. Azhurimi (update) i ndonjë dokumenti funksionon ngjashëm ku versioni i vjetër i dokumentit thjesht markohet si i fshirë ndërsa versioni i ri i dokumentit shtohet po sikur të ishte dokument i ri me ç'rast dokumenti i vjetër i cili është i markuar si i fshirë nuk kthehet si rezultat ndërsa

versioni i ri mund të kthehet si rezultat i ndonjë query. Me integrimin e pre-segment search në Lucene është zvogëluar vonesa e indeksimit të një dokumenti dhe vënies në dispozicion për kërkim të atij dokumenti me ç'rast dokumentet e reja mund të bëheshin të kërkueshme brenda disa minutave. Megjithatë kjo përsëri ishte një vonesë e madhe dhe arsye e kësaj vonese ishte shkrimi në disk i segmenteve të reja. Për këtë arsye u bë ndryshimi në bazë të cilit segmentet e reja përsëri shkruhen në disk mirëpo para kësaj ato shkruhen në cache të filesystemit gjë që mundëson që këto dokumente të jenë të kërkueshme dhe pastaj më vonë të shkruhen në disk nëpërmjet fsync. Me këtë ndryshim Lucene na mundëson që segmentet e reja të shkruhen dhe të jenë të qasshme pa bërë segmentin commit në disk. Në Elasticsearch procesi i shkrimit dhe hapjes (vënies në dispozicion për kërkim) të një segmenti njihet si refresh. Në kushte normale çdo shard kryen procesin e refresh automatikisht çdo një sekondë. Kjo është arsyeja pse Elasticsearch ofron kërkime gati se në kohë reale (near real-time) pasi që ndryshimet në dokumente nuk mund të kërkohen menjëherë mirëpo shfaqen brenda një sekonde [11].

JVM në të cilin ekzekutohet Elasticsearch e ka një hapësirë të dedikuar të memories që përdor. Shumica e memories të cilën e përdor Elasticsearch quhet heap. Elasticsearch në kushte normale përdor 256 MB RAM për heap dhe pastaj e zgjeron deri në 1 GB. Kërkesat të cilave ju nevojitet më shumë se 1 GB memorie dështojnë me ç'rast do të shënojnë një out-of-memory error në logs [10].

8. ALGOLIA

Algolia është një search engine i aftë që të kthejë rezultate në kohë reale. Në bazë të autorit shërbimi i tyre ju shërben me mijëra kompanive çdo muaj me ç'rast asnjë rezultat nuk vonohet më shumë se 100 ms. Për dallim nga Elasticsearch, Algolia nuk është e bazuar në Lucene, Solr apo ndonjë zgjidhje tjetër për kërkim. Algolia na mundëson kërkim të mirëfilltë duke përfshirë të gjitha elementet që kanë të bëjnë me kërkim si p.sh kërkimi me prefix, gabimet shkrimore, highlight i rezultateve etj. Algolia për dallim nga Elasticsearch dhe makinat kërkuese tjera nuk përdor algoritmin TF-IDF që është defacto për llogaritje të relevancës mirëpo përdor një algoritm të veçantë të shkruar nga vetë ekipa e Algolia që quhet tie-breaking algorithm dhe që i përmban gjashtë faktorë kyç që tregojnë se si të përcaktohet relevanca e dokumenteve. Pra ky algoritm do të i jap relevancë më të madhe një dokumenti që përputhet me fjalët që janë kërkuar pra që nuk përmban gabime drejtshkrimore, sa më shumë fjalë përputhen aq më të madh do të ketë rankimin ky dokument, vlerëson më shumë fjalët të cilat janë në distancë më të afërt ndaj njëra tjetrës, do të vlerësojë më shumë rezultatin që përputhet në fusha më të rëndësishme si p.sh titulli, përshkrimi etj, rankimi do të jetë më i lartë gjithashtu në rastet kur fjala ndodhet në fillim të dokumentit, do të rankojë më sipër fjalët të cilat përshtatën pa ndonjë prapashetë. Gjithashtu konfigurimi i përcaktimit të relevancës në Algolia është i thjeshtë¹¹.

Nëpërmjet tie-breaker algoritmit përfundimisht fitohen scores, këto scores grupohen në buckets të rezultateve. Kur përmbushet kriteri i parë, përdoret kriteri i dytë për të bërë ndryshimin e kështu me radhë vazhdon me kriteret tjera. Algolia është fleksibile në këtë aspekt pasi që mundëson që të vendosen edhe rregulla të tjera të cilat do të optimizionin relevancën për ndonjë data-set specifik. Në bazë të zhvilluesve të Algolia kjo është vetë mënyra si truri i njeriut bën sortimin e të dhënave duke i krahasuar me një kriter pas tjetrit deri sa të mbetën vetëm rezultatet që i nevojiten, gjë që është më intuitave sesa algoritmet komplekse matematikore¹².

¹¹ Algolia relevanca, <https://blog.algolia.com/how-algolia-tackled-the-relevance-problem-of-search-engines>

¹² Algolia, <https://www.algolia.com/doc/guides/getting-started/what-is-algolia>

8.1 Indeksimi

Algolia gjithashtu përdor indekse të cilat definojnë si entitete tek të cilat vendosen të dhënat të cilat dëshirojmë t'i indeksojmë dhe më vonë të kërkojmë në këto indekse duke ekzekutuar query ndaj tyre. Për të i indeksuar të dhënat për herë të parë duhet që të dhënat vetëm se ekzistuese të iterohen dhe të dërgohen në serverët e Algolia. Në rast se nevojiten modifikime apo transformime të dhënave atëherë këto veprime duhet të përfundohen para dërgimit të dhënave në Algolia. Në rast se objektet kanë ndonjë id unike dhe në qoftë se do të dëshironim të përdorim këtë id për të gjetur më lehtë dokumentet. Në raste të indeksimit të shumë objekteve është mirë të dërgohen objektet në batches. Në bazë të autorit preferohet që një batch të jetë i madhësisë 10 MB që reprezenton nga 1000 deri në 10000 objekte varësisht nga madhësia mesatare e një objekti. Popullimi i indekseve me të dhëna mund të bëhet nëpërmjet dashboard ku mund të shkruhen objektet në formatin json manualisht apo mund të bëhet ngarkimi i një file me madhësi më të vogël se 50 MB që mund të jetë në formatin json, csv, apo tsv nga i cili importohen objektet. Gjithashtu ekzistojnë disa extensions të zhvilluara nga ekipa e Algolia të cilat lehtësojnë procesin e indeksimit kur jemi duke punuar me disa platforma të palës së tretë. Deri më tani ekzistojnë ekstension për Magento, Wordpress, Zendesk, Shopify dhe Je¹³kyll.

8.2 Kërkimi i të dhënave

Për të marrë të dhëna nga Algolia duhet që të nxjerrët maksimumi nga metoda e kërkimit në indekse të Algolia. Për të bërë kërkime në Algolia duhet të definohet indeksi në të cilin do të bëhet kërkimi, një query e caktuar dhe parametra opcional të tjerë. Nëpërmjet parametrave opcional mundemi të ndryshojmë relevancën tekstuale, të mbishkruajmë settings në nivel të indeksit etj. Autori preferon që kërkimet në Algolia të bëhen direkt nga browseri i klientit në serverat e Algolia me ç'rast na mundëson evitimin e ngarkimit të serverëve tanë. Në disa raste të kërkimit nuk na nevojitet të kërkojmë në bazë të ndonjë teksti. Algolia ofron marrjen

¹³Algolia, <https://www.algolia.com/doc/guides/indexing/indexing-overview>

e të dhënave vetëm duke specifikuar indeksin apo ndonjë parametër tjetër opcional duke formuar një filtër në këtë rast. Rezultati i kërkimit në Algolia kthehet në formatin json. Ky rezultat i kërkimit gjithashtu përmban edhe të dhëna tjera të rëndësishme mbi rezultatet, shpejtësinë e kthimit të rezultateve, highlights etj. Relevanca tekstuale e Algolia në kushte normale për shumicën e data set nuk ka nevojë për modifikime të mëtutjeshme. Nga ana e klientit nevojitet që të specifikohen vetëm dy parametra që janë atributet e kërkueshme dhe rankimi i kostumizuar. Atributet e kërkueshme janë atributet të cilat janë në dispozicion për kërkim tekstual mbi to ndërsa rankimi i kostumizuar mundëson që të vendosim kritere që do të ndryshojnë mënyrën e rankimit. Nëpërmjet browse Algolia na mundëson që të iterojmë të gjithë objektet brenda një indeksi. Metoda browse është e optimizuar për shpejtësi dhe rezultatet e browse kthehen të rankuara në bazë të attributeve dhe rankimeve të kostumizuara. Browse mbështet shumicën e parametrave që i ofron search, me ç'rast mund të përdoren gjithashtu filterët për të marrë ndonjë rezultat¹⁴.

8.3 Infrastruktura

Algolia fillimisht kanë përdorë serverë me 4 bërthama dhe 8 threda mirëpo pasi që kanë hasur në probleme gjatë menagjimit të kërkesave të ndryshme paralelisht kanë kaluar në përdorimin e serverëve me 6 bërthama dhe 12 threads. Ekipa e Algolia kanë eksperimentuar me lloje të ndryshme të SSD RAID-0 deri sa së fundmi janë pajtuar me SSD S3500 series. Gjithashtu është kaluar kohë e gjatë përgjatë përzgjedhjes së RAM dhe hapësirës memorike pasi që nevojitet balancimi i tyre. Rëndësia e balancimit të RAM, me numrin e bërthamave dhe threads qëndron tek indeksat e mëdhenjë. Algolia mban indeksat e klientëve në RAM. I ndan indeksat e mëdhenjë në copa më të vogla ku secilës copë apo shard i jipet një thread në të njëjtin server. Virtualizimi është zgjedhja e shumicës të SaaS shërbimeve, mirëpo në rastin e Algolia është vendosur të përdoren bare metal servers. Bare metal servers ju referohemi serverëve të cilët kanë vetëm një tenant pra ato të cilat nuk janë të përbashkëta në mes të klientëve. Avantazhi i përdorimit të bare metal servers është qasja direkte e

¹⁴Algolia, <https://www.algolia.com/doc/guides/searching/searching-overview>

aplikacioneve në resurset fizike dhe softuerike për dallim nga makinat e virtualizuara ku duhet gjithmonë të kalohet nëpërmjet një apo më shumë shtresa të makinës virtuale për të arritur deri tek shërbimet e serverit fizik. Në rastin e Algolia bare metal servers janë zgjidhja optimale pasi që me komponentë harduerike të fuqishme, një server i vetëm mund të shërbejë shumë klientëve. Një klluster në Algolia përbëhet nga 3 serverë të bërë bashkë për të trajtuar kërkesat. Secili serverë është i njëjtë me të tjerët në klluster kështu që secili prej tyre është në gjendje të reagojë ndaj kërkesave. Për të qenë e mundshme që secili server të reagojë ndaj kërkesave duhet që ato të kenë të dhënat e njëjta, konfigurimet sistimore dhe settings të njëjta për tu sjell si një server i vetëm. E gjithë kjo bëhet për shkak të redundancës në rast se bien deri në dy serverë nga funksionimi mbetet serveri i tretë që vazhdon operimin. Në bazë të autorit kjo është mënyra se si Algo¹⁵lia garanton SLA prej 99.99% të qasjes.

¹⁵Infrastruktura e Algolia, <https://www.algolia.com/doc/guides/infrastructure/clusters>

9. KRAHASUESHMËRIA NË MES MAKINAVE KËRKUESE ELASTICSEARCH DHE ALGOLIA

Performanca e shërbimit të kërkimit të cilin e marrim n'varet nga përzgjedhja e search engine, hardware, konfigurimit, regjionit të data qendrave etj. Në aplikacione të tilla një rol të rëndësishëm luan edhe shkallëzimi (scaling) pasi që vonesat paraqiten çdokund, gjatë kërkimit mirëpo edhe në çdo hap tjetër ndërmjet kërkimit për këtë arsye preferohet që në aplikacione të tilla shkallëzimi të bëhet në mënyrë vertikale që të mos ketë burime të tjera të vonesave. Kërkimi në të dhëna është sfidues edhe për zhvillues me eksperiencë pasi që klientët gjithmonë bëjnë krahasime me kompanitë që ofrojnë kërkimin më të shpejtë siç janë google dhe Amazon me ç'rast duhet të ofrojmë së paku një eksperience të përafërt të kërkimit për të mbajtur bounce rate sa më të vogël që mundemi. Elasticsearch është më e përshtatshme se Algolia për streaming logs, mapim/reduktim të query, agregime komplekse dhe operime në data set gjigande me miliarda dokumente. Mangësi e Elasticsearch është nevoja e njohurive inxhinierike në front-end dhe back-end për të ndërtuar një search i cili mund të sfidojë shërbimet tjera të kërkimit të botës reale. Kërkimet instante fillojnë të japin ndjenjën e vonesës tek klienti tek vonesat më të mëdha se 100 milisekonda. Në bazë të Algolia preferohet që kërkimet të jenë më të shpejta sesa 50 milisekonda. Për të evituar vonesat që shkaktohen nga distanca ndërmjet pajisjes së klientit dhe data-qendrës tek e cila bëhet kërkimi nevojitet infrastruktura e data-qendrave e shpërndarë në regjione të ndryshme. Në rast se nevojitet të zhvendosen të dhënat nga një data-qendër në tjetrën Algolia mundëson replikimin automatik të indekseve në cilindo regjion nëpërmjet Distributed Search Network (DSN) të tyre. Në Elasticsearch edhe pse është e mundshme që të formohen klluster të instancave në data qendra të ndryshme një gjë e tillë nuk preferohet. Rekomandohet që në raste të tilla replikimi të bëhet nëpërmjet një messaging queue tek kllusterët në data-qendra tjera. Sa i përket RAM memories Indekset e Algolies janë të ruajtura në RAM dhe mapohen në nginx proces. Në Elasticsearch me konfigurimet e duhura mund të mbahen gjithashtu të gjitha indekset në RAM memorie. Elasticsearch mund të operojë në sisteme të virtualizuara mirëpo edhe në bare metal servers ku Algolia operon vetëm me bare metal servers. Algoritmi tie-breaker i Algolia bën sortimin në kohë të indeksimit dhe është një hap minimal ndërsa në

Elasticsearch sortimi mund të bëhet gjatë indeksimit apo në fund të çdo query. Sortimi në fund të query në Elasticsearch mund të shkaktoj vonesa në kthimin e rezultatit. Algolia përdor dy provider të DNS gjë që ofron siguri më të lartë në rast se bie njëri ndërsa Elasticsearch nuk ofron redundancë të DNS me ç'rast duhet të implementohet vetë. Sa i përket indeksimit dhe kërkimit Algolia i bën këto nëpërmjet proceseve të ndryshme të cilat kanë prioritet të ndryshme në procesor ndërsa në Elasticsearch duhet të konfigurohen nodes të ndryshme që të mirën me operate të ndryshme¹⁶.

Tabela 1 Dallimet kryesore ndërmjet makinave kërkuese Algolia dhe Elasticsearch

	Algolia	Elasticsearch
Transferimi i të dhënave ndërmjet data-qendrave	Algolia mundëson replikimin të indekseve në cilindo regjion nëpërmjet Distributed Search Network (DSN) të tyre.	Në Elasticsearch nuk preferohet që instancat nga data-qendra të ndryshme t'i takojnë një kllusteri.
RAM memoria	Indekset mbahen në RAM	Me konfigurimin e duhur Indekset mbahen në RAM
Operon	Vetëm në bare metal servers	Në bare metals servers dhe në sisteme të virtualizuara
Sortimi	Sortim në kohë indeksimi	Sortim në kohë indeksimi dhe në kohë kërkimi. Vonesë nga sortimin në kohë kërkimi
DNS	Dy provider të DNS. Ka redundancë për shkaqe sigurie	Nuk ofron DNS redundant. Duhet të implementohet manualisht

¹⁶Algolia Elasticsearch latency <https://blog.algolia.com/algolia-v-elasticsearch-latency>

Indeksi dhe Kërkimi	Kërkimi dhe indeksimi janë processe të ndryshme me prioritete të ndryshme	Duhet të konfigurohet për të përdor nodes të ndryshme për kërkim dhe indeksim
----------------------------	---	---

10. REZULTATET

Objektivë e këtij punimi është krahasimi i performancës të makinave kërkuese Elasticsearch dhe Algolia. Këto dy search engine dëshirojmë t'i krahasojmë si SaaS aplikacione ku në rastin e Algolia është e vetmja formë në të cilën mund të mirët ky shërbim ndërsa search engine Elasticsearch ofrohet si SaaS shërbim nga Amazon ueb service (AWS). Sa i përket resurseve kompjuterike të cilat ofrohen nga Algolia mund t'i referoheni kapitullit të 8-të të nënkapitulli i infrastrukturës ku sqarohet se çfarë resurse kompjuterike përdoren, mirëpo këto resurse janë të përbashkëta për disa klient dhe deri më tani nuk kemi hasur në ndonjë mënyrë për të gjetur resurset e sakta kompjuterike mbi të cilat kemi qasje. Në Amazon ueb service është përzgjedhur t2.small.Elasticsearch si virtual computing enviroment. Ky krahasim merr për bazë vetëm dy faktorë që janë koha e cila ju nevojitet makinave kërkuese për të kthyer rezultatin dhe numrin e dokumenteve të cilat i janë përshtatur kërkimit tonë. Ky krahasimi i performancave është zhvilluar duke përdorur data-set të ofruar nga sentiment140 me 1.6 milionë dokumente. Ky data-set edhe pse nuk bën pjesë në big-data ndodhet në prag të këtij definicioni dhe është optimal për studimin tonë për shkak të limitimeve në resurse kompjuterike. Gjatë këtij krahasimi kërkimi në search engine është bërë vetëm në fushën e përmbajtjes së tweets. Në tabelën vijuese mund të shihen rezultatet përfundimtare të krahasimit të bërë.

Tabela 2 Rezultatet e krahasimit të performancës të Algolia dhe Elasticsearch si SaaS servise

Query	Koha ekzekutimit Elasticsearch	Numri rezultateve Elasticsearch	Koha ekzekutimit Algolia	Numri rezultateve Algolia
presidential	3ms	18	6ms	37
Extreme	135ms	1374	1ms	1447
Manager	58ms	516	7ms	2429
Freeze	37ms	476	5ms	2125

Drawing	18ms	647	5ms	681
Annual	14ms	212	4ms	585
Punch	32ms	671	4ms	7291
License	61ms	475	3ms	618
transmision	19ms	57	10ms	65
Empire	13ms	201	5ms	713

11. KONKLUDIMI

Makinat kërkuese kanë lehtësuar punën tonë si zhvillues duke na ofruar shërbime të gatshme për kërkim në kohë reale apo gati se në kohë reale dhe duke fshehur kompleksitetin e menaxhimit të dhënave. Edhe pse është fshehur kompleksiteti i menaxhimit të dhënave, probleme mund të paraqiten edhe në integrimin e këtyre serviseve në ueb aplikacione. Nëpërmjet literaturës e cila është shfletuar në këtë temë diplome është sqaruar se si operojnë makinat kërkuese nga indeksimi i dokumenteve e deri tek kërkimi në to. Gjithashtu është vërejtur se ekzistojnë forma të ndryshme të implementimit të makinave kërkuese të cilat përdorin algoritme të ndryshme të kërkimit. Këto makina kërkuese mund të shfrytëzohen si servise SaaS të cilat ofrohen nga provider të ndryshëm. Nga literatura e shfletuar kemi fituar një kuptim më të thellë të mekanikave të makinave kërkuese. Megjithatë edhe pse ekzistojnë lloje të ndryshme të makinave kërkuese ende ka hapësirë për përmirësime të mëtutjeshme, për versione të reja të cilat përdorin algoritme më të optimizuara apo edhe makina kërkuese krejtësisht të reja të cilat kanë një arkitekturë unike në krahasim me ato parapraket. Pjesë e kësaj teme të diplome është edhe integrimi i makinave kërkuese në ueb aplikacionin es-
Algolia-performance-test. Në rast se i përdorim makinat kërkuese si SaaS shërbime atëherë një pjesë e madhe e përgjegjësiave mirën përsipër nga cloud computing provider. Integrimi është lehtësuar nëpërmjet përdorimit të librarive të dedikuara për këto makina kërkuese. Në shumicën e rasteve mjafton që në këto librari vetëm të bëjmë konfigurimin e të dhënave që e karakterizojnë makinën kërkuese si dhe të japim të dhënat për qasjen në atë makinë kërkuese. Pas kësaj mbetet të bëhet vetëm prezantimi i të dhënave nëpërmjet ndonjë user interface. Në rast se kemi nevoja të veçanta të cilat nuk përmbushen nga libraritë e ofruara atëherë mund të bëjmë modifikime në librarit që jemi duke i përdor për komunikimi apo edhe të krijojmë librarinë tonë për qasje në makinën kërkuese. Nga krahasimi i bërë mbi performancën e makinave kërkuese mund të konkludojmë se në shpejtësinë e kthimit të rezultateve Algolia ka qenë dukshëm më e shpejtë sesa Elasticsearch në data-setin e dhënë. A është Algolia makina kërkuese më e mirë atëherë? Jo, fillimisht data-set i dhënë ishte relativisht i vogël me 1.6 milionë dokumente ku këto makina kërkuese mund të përballojnë sasi më të mëdha të

dokumenteve e sidomos Elasticsearch e cila shkëlqen edhe në big-data. Përpos kësaj makinat kërkuese kanë specializimet e tyre për ku ato janë menduar të përdoren.

12.REFERENCAT

- [1] H. S. Saini, R. Sayal and S. S. Rawat, Innovations in Computer Science and Engineering, 1 ed., Springer Singapore, 2016.
- [2] P. Mell dhe T. Grance, The NIST Definition of Cloud Computing, 2009.
- [3] D. C. Marinescu, Cloud Computing Theory and Practice, Morgan Kaufmann, 2013.
- [4] J. Sen, Cloud Computing: Architecture and Applications, ExLi4EvA, 2017.
- [5] S. A. Ahson dhe M. Ilyas, Cloud Computing and Software Services Theory and Techniques, CRC Press, 2010.
- [6] T. Velte, A. Velte dhe R. C. Elsenpeter, Cloud Computing: A Practical Approach, McGraw-Hill Education, 2009.
- [7] B. Wilder, Cloud Architecture Patterns, O'Reilly Media, 2012, p. 182.
- [8] L. Deboosere, Cloud-based desktop services for thin clients IEEE Internet Computing, 2012, pp. 60-67.
- [9] C. D. Manning, P. Raghavan dhe H. Schütze, An Introduction To Information Retrieval, Cambridge, England: Cambridge University Press, 2009.
- [10] R. Gheorghe, M. L. Hinman dhe R. Russo, Elasticsearch in Action, Manning Publications, 2015.
- [11] C. Gormley dhe Z. Tong, Elasticsearch: The Definitive Guide, O'Reilly Media, 2015.
- [12] P. Dašić, J. Dašić dhe B. Crvenković, Applications of the search as a service (SaaS), Bulletin of the Transilvania University of Brasov, 2016.
- [13] P. Pandey, M. Kumar dhe P. Srivastava, Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on, New Delhi, India: IEEE, 2016.