UDC: 004.65 Info M str. 11-19

ANALIZA APACHE HADOOP-A I MOGUĆNOSTI UPOTREBE
APACHE HADOOP ANALYSIS AND USAGE POSSIBILITIES

dipl. ing. Dragan Šušak,
prof. dr Zoran Đurić

REZIME: U ovom radu urađena je analiza i ispitane mogućnosti upotrebe *Apache Hadoop* aplikativnog *framework-a*, zajedno sa njegovim propratnim alatima. Osim toga, data je i detaljnija analiza alternativnog *framework-a* za procesiranje podataka u distribuiranom okruženju, poznatog i kao *Apache Spark*. Takođe, opisani su hardverski, softverski i sigurnosni zahtjevi neophodni za uspostavljanje tipičnog *Hadoop* klastera, prema *Cloudera*-inoj referentnoj specifikaciji. Kao praktični dio, implementirana je relativno kompleksna aplikacija za kategorizaciju *Web* vijesti, korištenjem *Apache Spark* aplikativnog *framework-a* i principa mašinskog učenja. Specifikacija implementirane aplikacije data je odgovarajućim UML dijagramima, a navedeni su i ilustrativni primjeri kategorizacije novih *Web* vijesti.

KLJUČNE REČI: Apache Hadoop, Apache Spark, mašinsko učenje, kategorizacija vijesti

ABSTRACT: This paper analyzes Apache Hadoop application framework, some of its accompanying tools, and their usage possibilities. Research on alternative data processing framework in distributed environments, known as Apache Spark, is done in more detail. Description of hardware, software and security requirements, necessary for setting up a typical Hadoop cluster, according to Cloudera referent specification, is in this paper as well. As a practical example, relatively complex application for Web news classification is implemented, using Apache Spark application framework and machine learning principles. This paper also contains appropriate UML diagrams, used during the implementation process. As an illustration, few classification examples of unclassified Web news can also be found below.

KEY WORDS: Apache Hadoop, Apache Spark, machine learning, news classification

1. UVOD

Količina podataka svakim danom je sve veća. To se posebno odnosi na podatke u elektronskom obliku. Prema istraživanjima Međunarodne korporacije za podatke (IDC) količina elektronskih podataka se svake dvije godine približno udvostručava [1]. Prema ovim istraživanjima, 2013. godine veličina tzv. "digitalnog univerzuma" je bila 4,4 zetabajta, da bi 2020. godine ona trebala iznositi oko 44 zetabajta. Ovakav eksponencijalni rast će se nastaviti i u narednim godinama, a očekuje se da do 2025. godine, ukupna količina podataka bude veća od 175 zetabajta [2]. Generalno posmatrano, evidentan je porast količine podataka po osobi, ali isto tako, i količina podataka generisana od strane uređaja (npr. GPS uređaji, RFID čitači, senzori različitih namjena) je u stalnom porastu, i vrlo je izvjesno da će po obimu prevazići one generisane od strane ljudi [2]. Logična pitanja koja se nameću su pitanja skladištenja i obrade ovako velikih količina podataka. Pojam velike količine podataka u kontekstu *Hadoop*-a se odnosi na skupove podataka čija veličina se kreće u opsegu od nekoliko stotina gigabajta do nekoliko petabajta. S obzirom da standardni načini obrade i skladištenja nisu adekvatni za posao ovakvih razmjera, pronalaženje novih rješenja, orijentisanih ka intenzivnim obradama velikih količina podataka, je postao jedan od prioriteta velikih kompanija.

Kompanija *Google* je napravila prvi korak u ovom smjeru, kada je 2004. godine objavio novi programski model za paralelnu obradu podataka, pod nazivom *MapReduce.* Nekoliko godina kasnije, pojavljuje se *open source* rješenje, pod nazivom *Apache Hadoop*, zasnovano na *MapReduce* modelu i *Google* distribuiranom fajl sistemu. *Apache Hadoop* posjeduje i treću komponentu, *Apache* YARN, koji je zadužen za poboljšanje i pojednostavljivanje upravljanja resursima u distribuiranim sistemima obrade podataka kao što je *MapReduce.* Jedan od važnijih datuma u istoriji razvoja *Apache Hadoop*-a i kompletnog ekosistema je 19. februar 2008. godine, kada je jedan od najvećih svjetskih pretraživača *Yahoo*! objavio da je svoju kompletnu infrastrukturu za indeksiranje *Web* stranica prebacio na *Hadoop* klaster sa 10 000 računara. Od tog datuma pa sve do danas, brojne kompanije, poput *Amazon*-a, IBM-a, *Intel*-a i *Microsoft*-a su pronašle primjenu za *Apache Hadoop* u situacijama gdje količina podataka i potrebna snaga procesiranja prevazilaze do tada poznata konvencionalna rješenja.

U sekciji 2 ovog rada data je detaljna analiza *Apache Hadoop* aplikativnog *framework*-a i njegovih osnovnih gradivnih elemenata, gdje je naglasak stavljen na njihove dobre i loše osobine. Sekcija 3 sadrži opis pomoćnih *Hadoop* alata, koji se jednim imenom nazivaju *Hadoop* ekosistem, uz poseban osvrt na *Apache Spark* kao alternativni aplikativni framework za obradu podataka u distriburianim sistemima. U 4. sekciji data je analiza hardverskih, softwerskih i sigurnosnih zahtjeva, neophodnih za uspostavljanje *Hadoop* klastera, prema referentnoj specifikaciji kompanije *Cloudera*. Sekcija 5 sadrži detalje implementacije relativno kompleksne aplikacije za kategorizaciju *Web* vijesti, korištenjem *Apache Spark*-a i principa mašinskog učenja. Na kraju rada je dat i zaključak, gdje su sumirani razultati istraživanja.

2. OSNOVNI ELEMENTI HADOOP-A

Iako su tokom posljednjih nekoliko godina razvijeni brojni alati koji upotpunjavaju i proširuju mogućnosti *Hadoop*-a, te pojednostavljuju njegovo korištenje u svakodnevnom radu, tri osnovne komponente koje su prisutne od samog njegovog začetka su: MapReduce, HDFS (*Hadoop Distributed File System*) i YARN (*Yet Another Resource Negotiator*).

Slika 1. Osnovni Hadoop elementi

Pojednostavljena šema osnovnih gradivnih elemenata *Apache Hadoop*-a, na vrlo visokom nivou apstrakcije, koja ujedno oslikava i ulogu svake od njih u cjelokupnom sistemu, se može vidjeti na Slici 1.

HDFS

Kada količina podataka preraste mogućnosti skladištenja jednog računara, potrebno je izvršiti njihovo particionisanje i distribuciju na više mrežno povezanih mašina, računara, koji su uglavnom klasterizovani u jednom *data centru*. Fajl sistemi koji upravljaju podacima u ovakvim okruženjima se nazivaju distribuiranim. Pošto su ovakvi fajl sistemi bazirani na računarskim mrežama, jedan od najvećih problema je obezbijediti stabilnost i redundantnost sistema u situacijama otkaza pojedinih čvorova mreže, kako bi se spriječio eventualni gubitak podataka. *Hadoop*-ov distribuirani fajl sistem, upravo nudi visok nivo otpornosti na otkaze kroz replikaciju podataka na pojedinim čvorovima *Hadoop* klastera. HDFS je osmišljen kao fajl sistem na kojem se mogu skladištiti velike količine podataka, nad kojima se u najvećem broju slučajeva izvršavaju operacije čitanja. To znači da je u cilju što boljih performansi HDFS-a, poželjno podatke jednom upisati na fajl sistem, i naknadno ih po potrebi čitati proizvoljan broj puta prilikom obrade. Svaka obrada bi trebala uključivati što veću količinu podataka, u idealnom slučaju sve upisane podatke koje jedna *Hadoop* aplikacija treba, kako bi se minimizovalo uticaj vremena traženja prvog zapisa podataka na performanse sistema. Imajući ova ograničenja na umu, lako se mogu prepoznati slučajevi upotrebe u kojima HDFS nije poželjno koristiti. To su slučajevi gdje aplikacije zahtijevaju veoma brz pristup podacima, u vremenskom opsegu od nekoliko milisekundi. HDFS je kao i većina sličnih distribuiranih fajl sistema optimizovan za čitanje velikih količina podataka, dok je brzina pristupa stavljena u drugi plan.

Hadoop distribuirani fajl sistem se obično nalazi u klasterizovanom okruženju, gdje računari koji sačinjavaju klaster takođe predstavljaju i čvorove HDFS-a. HDFS posjeduje dva tipa čvorova, koji rade u *master-slave* režimu rada. *Namenode* ili *master* čvor ima odgovornost upravljanja cjelokupnim *Hadoop* fajl sistemom. Ovaj čvor sadrži kompletnu strukturu fajl sistema i metapodatke za sve datoteke i direktorijume u jednom HDFS-u. Ova struktura je obično smještena na nekom od diskova računara koji ima ulogu *master* čvora. *Master* čvor takođe zna i lokaciju svih blokova po pojedinim čvorovima sistema. Klijentska aplikacija koja treba pristup nekom od fajlova HDFS-a, komunicira sa *master* i *slave* čvorovima sistema potpuno transparentno, ne znajući ništa o internoj strukturi samog sistema. *Datanode* ili *slave* čvorovi su odgovorni za skladištenje podataka na fajl sistem. Oni upisuju i čitaju podatke kad god za to postoji potreba, izražena od strane aplikacije ili *master* čvora sistema. Takođe, *slave* čvorovi šalju *master* čvoru periodično podatke o tome koje blokove podataka skladište. Često korišteni blokovi se mogu keširati u memoriji ovih čvorova ako za to postoji potreba, a u cilju poboljšanja performansi. Korisnici fajl sistema mogu pomoću odgovarajućih komandi *namenode-u* instruisati koji fajlovi i na koji period vremena trebaju biti keširani.

Po pitanju interakcije sa HDFS-om, isti je poprilično fleksibilan i nudi različite vidove pristupa datotekama i strukturi direktorijuma. Svakako jedan od najkorištenijih je interfejs komandne linije, a same komande su po obliku vrlo slične odgovarajućim *Linux* komandama za manipulaciju datetekama i direktorijumima. Na Slici 2. se nalaze neke od najkorištenih HDFS komandi.

```
// komanda za prikaz sadržaja datoteke
% hdfs dfs cat file.txt

// komanda za brisanje datoteke
% hdfs dfs -rm file.txt

// komanda za kreiranje novog direktorijuma
% hdfs dfs -mkdir some_hdfs_folder

// komanda za prikaz sadržaja trenutnog direktorijuma
% hdfs dfs -ls .
```

Slika 2. Primjeri HDFS komandi

MapReduce

MapReduce je jednostavan programski model za paralelno procesiranje podataka, koji omogućava tačno definisanje koraka u procesu obrade tih podataka, pomoću odgovarajućeg programskog interfejsa na nekoliko programskih jezika. Kao i većina procesnih modela, *MapReduce* takođe ima više faza ili koraka u procesu obrade, koje se simbolično nazivaju faze mapiranja i redukovanja. Ono po čemu se *MapReduce* izdvaja od ostalih sličnih modela za procesiranje je mogućnost skaliranja, gdje se problem performansi u procesno zahtjevnim aplikacijama jednostavno rješava dodavanjem novih čvorova u klaster na kojem se aplikacija izvršava. Iako uveliko zavisi od konkretnog problema koji se rješava kao i strukture ulaznih podataka, većina *MapReduce* programa radi sa jednostavnim tipovima podataka. Najčešće su to liste i ključ-vrijednost (eng. *key/value*) parovi, gdje se za tip ključa i tip vrijednosti uglavnom koriste cijeli brojevi (eng. *integer*) i nizovi alfanumeričkih karaktera (eng. *string*) [3].

U fazi mapiranja, *MapReduce* program filtrira i transformiše ulazne podatke na način pogodan za daljnju obradu u fazi redukovanja. Kada je riječ o aplikativnom programskom interfejsu, implementacija ove faze u većini programskih jezika se ostvaruje kroz programiranje odgovarajućih funkcija mapiranja. Primjer jedne takve funkcije za brojanje različitih riječi u nekom fajlu se može vidjeti na Slici 3. Ulaz faze ma-

piranja mora biti strukturiran kao lista *key/value* parova, gdje se nad svakim parom poziva funkcija mapiranja. Vrijednost *map* funkcije se dalje transformiše u novi *key/value* par koji se kao međurezultat prosljeđuje fazi redukovanja. Način na koji se ova transformacija vrši uveliko zavisi od logike samog *MapReduce* programa.

```
map(String key, String value) {
      List<String> T = tokenize(value);
      for each token in T {
            emit((String)token, (Integer)1);
      }
}
```

Slika 3. Pseudokod map funkcije za primjer brojanja riječi

Faza redukovanja je zadužena za sumiranje međurezultata i formiranje konačnog izlaza programa. Kao i faza mapiranja, obično se implementira kao *reduce* funkcija, koja kao ulazne parametre ima agregirane rezultate iz prethodne faze. Na Slici 4. se može vidjeti pseudokod jedne *reduce* funkcije, takođe za slučaj prebrojavanja različitih riječi u nekoj datoteci.

```
reduce(String token, List<Integer> values) {
      Integer sum = 0;
            for each value in values {
                  sum = sum + value;
      }
      emit ((String)token, (Integer)sum);
}
```

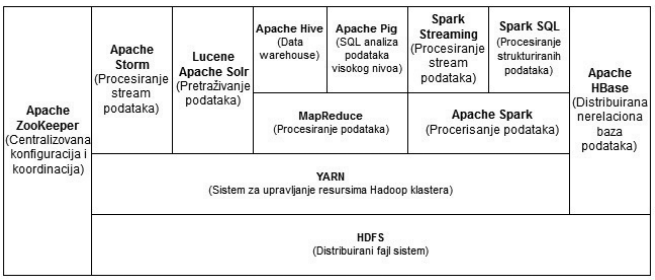
Slika 4. Pseudokod reduce funkcije za primjer brojanja riječi

Yet Another Resource Negotiator (YARN)

Apache YARN je sistem za upravljanje resursima unutar *Hadoop* klastera. Dostupan je od verzije 2.0 *Apache Hadoop*-a, a glavni cilj mu je poboljšanje i pojednostavljenje upravljanja resursima u distribuiranim sistemima. U ranijim verzijama, ovi poslovi su bili dio *MapReduce* aplikativnog *famework*-a. Jedan od razloga za uvođenje ovog dodatnog sloja u arhitekturu *Hadoop*-a je podjela odgovornosti između slojeva, te povećanje skalabilnosti i dostupnosti. Takođe, povećanje fleksibilnosti je jedan od efekata ove podjele, koji se ogleda u mogućnosti izvršavanja drugih tipova distribuiranih aplikacija na *Hadoop* klasteru, poput *Spark* aplikacija. Iako YARN posjeduje API za rad sa resursima u klasterskom okruženju, ovaj API nije direktno korišten od strane korisnika. U normalnim slučajevima, korisnik definiše potrebnu programsku logiku pomoću API-ja nekog od distribuiranih *framework*-a za obradu podataka, kao što su *MapReduce* ili *Spark*. Dva najbitnija pojma kada je u pitanju YARN i aplikacije zasnovane na njemu su upravljač resursima (eng. *resource manager)* i upravljač čvora (eng. *node manager).* U normalnom *Hadoop* klasteru, obično postoji jedan *resource manager* koji je odgovoran za upravljanje resursima u čitavom klasteru. S druge strane, odgovornost *node manager*-a je pokretanje i nadgledanje kontejnera u kojima se izvršava aplikacija, na pojedinim čvorovima klastera. Zahtijevanje resursa se može izvršiti na jedan od dva načina. Prvi je zahtijevanje svih potrebnih resursa odmah na početku izvršavanja aplikacije, što je slučaj kada je u pitanju *Spark* aplikativni *framework*. Drugi način, koji inače koristi *MapReduce*, je dinamički način, gdje se na početku izvršavanja aplikacije zahtijevaju samo resursi potrebni za izvršavanje svih zadataka faze mapiranja, dok se ostatak kontejnera potrebnih za izvršavanje zadataka *reduce* faze zahtijeva po potrebi.

3. HADOOP EKOSISTEM

Iako veoma robustan i skalabilan, i u stanju da riješi većinu problema, sam *Hadoop* u osnovnom okruženju postaje vrlo brzo neodrživ u slučaju velikih aplikacija. Programski kod potreban za definisanje faze mapiranja i redukovanja u ovakvim slučajevima postaje veoma kompleksan, nerazumljiv i nepregledan, pogotovo kada je riječ o aplikacijama koje se izvršavaju i održavaju duži niz godina. Pristup podacima na HDFS-u pomoću osnovnih operacija distribuiranog fajl sistema je takođe vrlo ograničen i nepraktičan u slučaju kompleksnijih slučajeva upotrebe. Iz tog razloga, tokom nekoliko godina nakon nastanka *Hadoop*-a, razvijeni su brojni pomoćni alati i podsistemi, zasnovani na *Hadoop*-u, koji u velikoj mjeri pojednostavljuju i ubrzavaju razvoj kompleksnijih aplikacija, a i dodatno olakšavaju proces održavanja istih. Svi ovi alati se u većoj ili manjoj mjeri oslanjaju na *Hadoop* i jednim imenom se nazivaju *Hadoop* ekosistem. Na Slici 5. se može vidjeti i grafički prikaz ovog ekosistema sa najčešće korištenim alatima.



Slika 5. Hadoop ekosistem

Apache HBase

Apache HBase je kolonski orijentisana nerelaciona baza podataka koja se izvršava na *Hadoop* distribuiranom fajl sistemu i omogućava čitanje i upisivanje velikih količina podataka na HDFS u realnom vremenu. *HBase* ima mogućnost linearnog skaliranja u slučaju velikih kolekcija podataka sa fleksibilnom strukturnom i šemom. Jedan od zahtjeva za linearno skaliranje je taj da svaka tabela ima primarni ključ. Neke od najbitnijih osobina *HBase*-a su: otpornost na greške, brzina i mogućnosti primjene. Otpornost na greške se ostvaruje kroz replikaciju podataka unutar jednog *data centra*, te kroz atomičnost i konzistetnost operacija niskog nivoa. Takođe, visoka dostupnost i balansiranje opterećenja na nivou pojedinih tabela, se mogu svakako navesti kao pozitivne osobine. Brzina čitanja i pretraživanja se ostvaruje približno u realnom vremenu zahvaljujući mehanizmima memorijskog keširanja te pro-

cesiranja na serverskoj strani. Mogućnosti primjene su brojne, prije svega zahvaljujući fleksibilnom modelu podataka, te jednostavnom *Java* i REST interfejsu za komunikaciju [12]. *HBase* klaster je takođe organizovan po *master-slave* principu i sastoji se od jednog *master* čvora koji upravlja jednim ili više *region* servera. U slučaju distribuiranih baza podataka, koje se inherentno karakterišu velikom količinom podataka i lakoćom horizontalnog skaliranja, ispunjavanje ACID (*Atomicity, Consistency, Isolation, Durability*) principa nije uopšte jednostavno, ako se žele zadržati zadovoljvajuće performanse i dobre osobine distribuiranih sistema. Iz tog razloga, većina distribuiranih baza podataka, uključujući i *HBase*, nastoji da obezbijedi neku vrstu manje restriktivnog ACID principa, iz već navedenih razloga. Tako u slučaju *HBase*-a, atomičnost je osigurana na nivou reda i ćelije neke tabele na jednom serveru regiona. Što se tiče konzistentnosti, *HBase* trenutno ne nudi opciju poništavanje transakcije (eng. *rollback*) u slučaju greške. Izolacija je ispunjena do neke mjere, ali i dalje ne na istom nivou kao kod relacionih baza podataka. Trajnost je osigurana u slučaju kad je transakcija uspješna.

Apache Pig

Jedan od glavnih problema *MapReduce*-a je kompleksnost koja dolazi do izražaja u situacijama nešto kompleksnijih aplikativnih zahtjeva. Formulisati i podijeliti ovakve zahtjeve na fazu mapiranja i redukovanja nije nimalo jednostavan zadatak, koji se dodatno usložnjava kada postoji potreba za povezivanjem više *MapReduce* poslova (eng. *job*). Određeni konstrukti, kao što je na primjer združivanje (eng. *join*), koji su kod većine jezika visokog nivoa za obradu podataka podržani i jednostavni, kod *MapReduce*-a nisu trivijalni za upotrebu. *Apache Pig* je jedno od proširenja *Hadoop*-a, koje nastoji riješiti ovaj problem pomoću još jednog jezika za procesiranje podataka, nazvanog *Pig Latin*, a koji omogućava izražavanje kompleksnih operacija nad *Hadoop* podacima na relativno jednostavan i intuitivan način. Pomoću ovog jezika, kompleksne *MapReduce* transformacije se mogu predstaviti u obliku upita, sličnih SQL upitima. S obzirom da je zasnovan na *MapReduce*-u i HDFS-u, visoka skalabilnost i pouzdanost se podrazumijevaju i u slučaju *Apache Pig* aplikacija. Interakcija se najčešće ostvaruje putem ugrađenog *shell* interfejsa zvanog *grunt* [3]. *Pig Latin* je po sintaksi vrlo sličan SQL-u, a uporedan primjer upita za izlistavanje sadržaja se može vidjeti na Slici 6.

```
//SQL upit
SELECT * FROM proizvod;

//Pig Latin upit
proizvodi = LOAD proizvod.txt AS (naziv, opis, cijena);
DUMP proizvodi;
```

Slika 6. Upredni primjer SQL i Pig Latin upita

Za razliku od SQL-a, *Pig Latin* zahtijeva i definisanje strukture podataka koji se čitaju, te njihovo smještanje u privremenu varijablu ili relaciju, nad kojom se kasnije mogu izvršavati dodatne komande. Takođe bitna razlika je i način izvršavanja, jer *Pig* komanda *LOAD* se ne izvršava automatski nakon poziva iste, već tek nakon upotrebe komande *DUMP ili STORE,* koje vrše ispisivanje učitane relacije na standardni izlaz ili smještanje na HDFS. Definisanje šeme prilikom pisanja *Pig job*-ova nije obavezno i striktno, te u slučaju HDFS fajla, čija struktura ne zadovoljava definisanu šemu, višak ili manjak podataka u tom fajlu će biti ili ignorisan ili popunjen *NULL* vrijednostima [3].

Apache Hive

Slično kao i *Apache Pig*, *Apache Hive* je nastao iz potrebe za pojednostavljivanjem pisanja *MapReduce job*-ova. *Hive* se može definisati kao *data warehousing* alat za procesiranje velikih količina podataka, reda nekoliko petabajta, zasnovan na *Apache Hadoop*-u. Prvi puta je korišten od strane kompanije *Facebook*, za procesiranje velikog broja serverskih logova. Zahvaljujući jednostavnoj sintaksi, koja je skoro identična SQL-u, *Hive* se nametnuo kao jedan od glavnih alata korištenih od strane analitičara, za analizu podataka smještenih na *Hadoop*-u. Dizajniran je u svrhu procesiranja i pretraživanja strukturiranih podataka, uz ugrađenu automatsku optimizaciju SQL upita, što dodatno olakšava rad krajnjem korisniku. Takođe, od svog nastanka 2008. pa sve do danas *Hive* se smatra *de facto* standardom za SQL procesiranje na *Apache Hadoop*-u. *Apache Hive*, slično kao i većina konvencionalnih relacionih baza podataka, takođe posjeduje koncepte tabele, kolona i redova. Poboljšanje performansi se može ostvariti kroz dodatno particionisanje *Hive* tabela, a s obzirom da se iste smještaju kao fajlovi na HDFS, particionisanje se ostvaruje kroz kreiranje odgovarajuće hijerarhije direktorijuma i poddirektorijuma. Za smještanje ovakvih i sličnih metapodataka, *Hive* koristi relacionu bazu podataka zbog što boljih performansi pristupa metapodacima. Ono po čemu se *Hive* razlikuje od konvencionalnih baza podataka je nepostojanje koncepta primarnog i stranog ključa, samim tim ne postoji ni koncept relacije. Interakcija sa *Hive*-om se može ostvariti na više načina, a najčešće putem JDBC-a *(Java Database Connectivity)* i interfejsa komandne linije [4]. Nekoliko primjera *HiveQL* upita dato je na Slici 7, gdje se vidi njegova sličnost sa standardnim upitnim jezikom.

```
SELECT COUNT(*) FROM proizvod WHERE cijena = 10;

SELECT MAX(cijena) FROM proizvod;

SELECT cijena, COUNT(*) FROM proizvod GROUP BY cijena;

SELECT * FROM proizvod LEFT JOIN racun ON proizvod.id = racun.proizvod_id;
```

Slika 7. Primjeri HiveQL upita

Apache Spark

Apache Spark se može definisati kao *framework* za procesiranje velikih količina podataka u klaster okruženju. Za razliku od drugih alata, poput *Apache Pig*-a i *Apache Hive*-a, koji su zasnovani na *MapReduce*-u, *Spark* posjeduje sopstveni *engine* za procesiranje podataka u distribuiranim sistemima.

Uprkos tome, *MapReduce* i *Spark* imaju i dosta dodirnih tačaka, posebno po pitanju API-ja, izvršavanja u YARN okruženju i korištenja HDFS-a kao sloja za perzistenciju. Ono po čemu je *Spark* još razlikuje od *MapReduce*-a je mogućnost skladištenja velikih količina radnih podataka u memoriji, i to uglavnom između različitih *Spark* jobova. Ovakav način rada omogućava *Spark*-u značajno poboljšanje performansi u odnosu na *MapReduce*, koji inherentno smješta sve podatke na *Hadoop* fajl sistem. Zahvaljujući ovakvom načinu procesiranja, mogu se identifikovati dva tipa korisničkih aplikacija, pogodnih za implementaciju pomoću *Apache Spark*-a. Jedan tip su aplikacije zasnovane na iterativnim algoritmima, gdje se određene funkcije izvršavaju proizvoljan broj puta nad podacima, sve dok se ne postigne zadovljavajući rezultat. Drugi tip aplikacija su one koje zahtijevaju interaktivnu analizu nad određenim skupom podataka, gdje krajnji korisnik izvršava niz upita nad podacima od interesa [5]. Fleksibilnost procesiranja koju *Apache Spark* nudi, a koja se ogleda u mogućnosti definisanja proizvoljnih faza izvršavanja aplikacije, je još jedna od prednosti u odnosu na *MapReduce.* Zahvaljujući svojoj fleksibilnosti i dobrim performansama, *Spark* se pokazao kao dobra platforma i osnova za razvoj analitičkih alata, poput onih za mašinsko učenje, procesiranje grafova i tokova podataka, te pisanje i izvršavanje SQL upita [6].

Slično kao i *MapReduce*, *Spark* poznaje pojam poslova (eng. *job*), s tim da su ovi poslovi dosta univerzalniji i sastoje se od proizvoljnog skupa faza, organizovanih u obliku usmjerenih acikličnih grafova, (eng. *Directed Acyclic Graph* - DAG). Svaka od ovih faza je približno jednaka već pomenutim fazama mapiranja ili redukovanja kod *MapReduce*-a. *Spark* faze su po pravilu podijeljene u zadatke (eng. *task*), koji se izvršavaju u paraleli nad distribuiranim skupom podataka otpornih na gubitak informacija (eng. *Resilient Distributed Dataset* - RDD). U slučaju gubitka neke od particija RDD-a, *Spark* može izvršiti rekonstrukciju iste kroz ponovnu rekalkulaciju. *Spark job*-ovi se uvijek izvršavaju u kontekstu aplikacije, koja je krajnjem korisniku, programeru, dostupna putem *SparkContext* instance. Spark aplikacija po pravilu može da izvršava više jobova istovremeno, serijski ili paralelno, pri tom obezbjeđujući jednostavan i intuitivan način pristupa keširanim RDD-ovima, kreiranim od strane drugih *job*-ova u istoj aplikaciji. Primjer jednostavne *Spark* aplikacije, za brojanje riječi u nekon fajlu, napisane u *Java* programskom jeziku, dat je na Slici 8.

```
JavaRDD<String> textFile = sparkContext.textFile("ulaz.txt");

JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);

counts.saveAsTextFile("izlaz.txt");
```

Slika 8. Primjer jednostavne Apache Spark aplikacije za brojanje riječi

Iz ugla *Spark job*-a, na najvišem nivou razlikuju se dva nezavisna koncepta, koji čine i osnovne komponente *job*-a. Prvi je upravljač (eng. *driver*) koji je odgovoran za upravljanje *Spark* aplikacijom i raspoređivanje zadataka. Drugi koncept je koncept izvršioca (eng. *executors*) koji postoje jedino u kontekstu neke aplikacije i koji izvršavaju taskove te aplikacije na pojedinim čvorovima klastera. Izvršavanje *Spark* posla počinje automatski, nakon poziva neke od akcija nad RDD-om, koje za posljedicu ima poziv odgovarajuće funkcije za početak izvršavanja *job*-a nad *SparkContext* instancom. Nakon toga, *Spark job* prolazi kroz nekoliko faza, u kojima različiti raspoređivači (eng. *schedulers*) vrše podjelu istog na proizvoljan broj faza i zadataka, koji se kasnije izvršavaju na čvorovima klastera. Raspoređivač zadataka vrši raspodjelu pojedinih zadataka na proizvoljan broj izvršioca, koji su prethodno alocirani aplikaciji. Naravno, da bi se neki od izvršioca mogao koristiti, isti mora da ima na raspolaganju slobodne procesorske resurse. U toku izvršavanja nekog *Spark job*-a, može se desiti da ne postoji slobodan izvršilac za neki od neizvršenih zadataka. U tom slučaju taj zadatak će biti pauziran sve dok neki drugi zadatak ne završi sa izvršavanjem i oslobodi potrebne resurse.

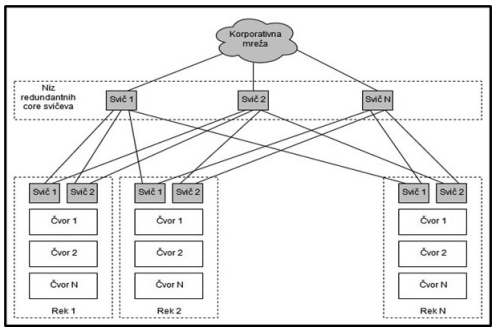
Apache Zookeeper

Prilikom razvoja distriburanih aplikacija, upravljanje i koordinacija pojedinih dijelova sistema su zasigurno među najtežim zadacima u ovom procesu, i to prvenstveno zbog faktora na koje krajnji korisnik, nema uticaja. Najčešći problemi su svakako nepouzdanost i nepredviđeni otkazi pojedinih dijelova infrastrukture, koji po pravilu dovode do kašnjenja ili potpunog prekida rada informacionog sistema. Iako se ovakvi problemi po pravilu ne mogu potpuno riješiti na aplikativnom nivou, već na infrastrukturnom, i to najčešće kroz obezbjeđivanje određenog stepena redundancije, *Apache Zookeeper* nastoji minimizovati uticaj ovih problema na distribuirane sisteme zasnovane na *Hadoop*-u. U ovakvim sistemima, vrlo često postoji i potreba za komunikacijom između pojedinih dijelova ili komponenti sistema iako ti dijelovi nisu u direktnoj vezi ili čak i ne znaju za postojanje drugih komponenti. *Zookeeper* nastoji da riješi i ovaj problem kroz odgovarajuće mehanizme koordinacije i otkrivanja servisa, a sve u cilju pojednostavljivanja razvoja distribuiranih sistema, i olakšavanja posla programeru. U najužem smislu, *Zookeeper* se može definisati kao sistem za pouzdanu koordinaciju, konfiguraciju i sinhronizaciju distribuiranih informacionih sistema. Jedna od najvažnijih osobina *Apache Zookeeper*-a, kao i većine ostalih *Hadoop* zasnovanih *framework*-a, je visoka pouzdanost i nepostojanje jedne, izolovane tačke otkaza. *Zookeeper* se automatski replicira na više lokacija unutar *Hadoop* klastera, što omogućava nesmetano funkcionisanje sistema i u slučaju otkaza pojedinih čvorova. Takođe, dvije vrlo bitne prednosti *Zookeeper*-a su svakako jednostavnost i brzina, i to u situacijama kada je čitanje podataka mnogo češće od upisivanja [7].

4. REFERENTNA HADOOP ARHITEKTURA

Pokretanje i izvršavanje osnovnih *Hadoop* komponenti, *MapReduce*-a, HDFS-a i YARN-a, na jednom računaru, može biti vrlo korisno u početnoj fazi, fazi upoznavanja i testiranja *Hadoop*-a i njegovih komponenti. Ipak, za nešto ozbiljnije slučajeve upotrebe, izvršavanje u klaster okruženju, gdje se

prednosti *Hadoop* sistema mogu u potpunosti iskoristiti, je podrazumijevani i jedini pravi način upotrebe. Upravo iz tog razloga, potrebno je posvetiti više pažnje procesu uspostavljanja jednog *Hadoop* klastera u produktivnom okruženju, a jedan od mogućih pristupa definisan je *Cloudera*-inom referentnom arhitekturom.



Slika 9. Referentna arhitektura klastera sa fizičkim uređajima

Na Slici 9. prikazan je fizički ramještaj pojedinih hardverskih komponenti *Hadoop* klastera. Svaki od računara klastera je povezan na dva mrežna sviča (eng. *switch*), koji se nalaze u istom reku (eng. *rack*) kao i računari. U zavisnosti od konkretnog slučaja upotrebe, svaki od ovih svičeva je povezan sa dva ili više *core* sviča kako bi se obezbijedila redundantnost u slučaju otkaza. Na kraju, svaki od *core* svičeva je povezan na internu mrežu kompanije, koja u zavisnosti od potrebe može dozvoliti javni pristup samom klasteru. Ovakva mrežna topologija osigurava visok stepen redundantnosti, jer u slučaju otkaza bilo kojeg od mrežnih uređaja, njegovu funkciju preuzima drugi uređaj, jednakih ili sličnih karakteristika. Maksimalan propusni opseg, visok stepen skalabilnosti kao i minimalna mrežna kašnjenja su takođe osigurani kroz ovakvu mrežnu postavku. Prestanak funkcionisanja nekog od *rack*-ova je jedino moguć u slučaju otkaza oba *switch*-a dodijeljena tom *rack*-u. Čak i u tom slučaju, dostupnost cjelokupnog *Hadoop* kalstera nije ugrožena, ukoliko su redundatni *master* čvorovi raspoređeni u drugim *rack*-ovima. Pristup Internetu iz samog klastera, iako nije obavezan je jako poželjan u slučaju dugovječnih klastera, prvenstveno zbog ažuriranja postojećih softverskih alata što uveliko olakšava administraciju i održavanje [8].

Cloudera referentna specifikacija za svaki od čvorova klastera preporučuje minimalno dvije procesne jedinice, sa 8 do 14 jezgara i više od 2 GHz radnog takta. Po pitanju memorijske konfiguracije, preporučuje se 128 GB radne memorije po čvoru, od čega bi 4 GB trebalo biti rezervisano za operativni sistem i ostale servise, koji ne pripadaju *Hadoop* ekosistemu. Naravno u slučaju konkretnog *Hadoop* klastera, ove vrijednosti mogu da variraju, u zavisnosti od konretnog slučaja upotrebe i aplikacija koje se izvršavaju u tom klasteru. Kada je riječ o mrežnom interfejsu računara, svaki bi trebao da posjeduje minimalno jednu *Ethernet* mrežnu kartu sa 10 Gbps propusnog opsega. Slično kao i kod računarskih resursa, *Cloudera* daje i određene preporuke po pitanju mrežnih uređaja, u ovom slučaju svičeva, kako bi se obezbijedio maksimalan propusni opseg, visok stepen skalabilnosti uz minimalna mrežna kašnjenja. Tako bi svaki od svičeva trebao da ima 10 Gbps propusni opseg, sa dovoljnim brojem portova da omogući nesmetano funkcionisanje svih dijelova klastera, a u isto vrijeme da podržava eventualno dodavanje novih čvorova. Preporučuju se po dva sviča na nivou jednog *rack*-a, dok broj *core* svičeva zavisi od broja *rack*-ova klastera, kao i od željenog stepena redundantnosti. Kada je riječ o *master* čvorovima klastera, *Cloudera* preporučuje dva SSD (eng. *Solid State Drive*) diska, povezana u RAID 1 *(Redundant Array of Independent Disks)*, kapaciteta 500 GB, za smještanje operativnog sistema čvora i *Hadoop* logova. Takođe, master čvor bi trebao imati i dodatna dva diska, kapaciteta 1 TB, povezanih RAID 1 konfiguracijom, za smještanje metapodataka HDFS *namenode*-a. Preporučuje se i zaseban HDD *(Hard Disk Drive)* disk, kapaciteta 1 TB, dodijeljen *Zookeeper* procesu master čvora. Konfiguracija radnih čvorova bi takođe trebala da sadrži dva SSD diska, u RAID 1 konfiguraciji, za smještanje operativnog sistema čvorova i log fajlova. Sto se tiče diskova na kojima se smještaju *Hadoop* podaci, preporučuju se između 16 i 24 magnetna diska, kapaciteta do 4 TB, povezani u RAID 0 konfiguraciju.

Kako bi cjelokupan *Hadoop* sistem funkcionisao na zadovoljavajućem nivou, određena razmatranja po pitanju korištenog softvera takođe moraju biti urađena. Prema *Cloudera*-inoj referentnoj arhitekturi, nekoliko distribucija *Linux* operativnog sistema su podržane i u najnovijim verzijama *Clouder*-ine *Hadoop* distibucije. Najpopularnije su svakako RHEL *(Red Hat Enterprise Linux)*, *CentOS*, *Oracle*, *Ubuntu*, te SLES *(SUSE Linux Enterprise Server) Linux* distribucije. S obzirom da se *Hadoop* klaster sastoji od velikog broja računara, preporučuje se i određena konvencija imenovanja istih, u cilju lakše identifikacije, na osnovu funkcije i lokacije računara u klasteru. Takođe, korištenje DNS *(Domain Name System)* servera za rezoluciju imena čvorova je dobra ideja, s obzirom da *hosts* fajl vrlo brzo postaje neodrživ, kada veličina klastera pređe određeni broj čvorova. Većina administrativnih poslova na nivou klastera se može prilično jednostavno odraditi korištenjem *Cloudera Manager* alata. Nekoliko tipova fajl sistema se smatra optimalnim za upotrebu u *Hadoop* klasteru, a prema *Clouder*-inoj specifikaciji preporučuje se upotreba *extent* baziranih fajl sistema, od kojih su najkorišteniji *xfs*, *ext3* i *ext4* [8].

Kada se uzmu u obzir veličina i ekonomski troškovi jednog Hadoop klastera, sigurnosni aspekti sistema dodatno dobijaju na značaju. Takođe, kompanije odgovorne za razvoj i održavanje sistema koji sakupljaju i obrađuju lične podatke krajnjih korisnika, ukoliko žele da zadovolje sve zakonske procedure o zaštiti tih podataka, moraju da ulože dodatne napore kako bi sigurnost svojih sistema podigle na viši nivo. Glavni zahtjevi u slučaju *Hadoop* informacionih sistema, kao i drugih sličnih sistema za upravljanje velikim količinama podataka, su svakako povjerljivost informacija, njihov integritet ali i dostupnost. Izolacija računarske mreže u kojoj se klaster nalazi, kroz odgovarajuće *firewall* konfiguracije, te konfiguracije mrežnih uređaja, je preferiran način postizanja zadovoljavajućeg stepena mrežene sigurnosti. Obavezna autentikacija svih korisnika i aplikacija koje imaju pristup *Hadoop* klasteru, prije samog pristupa istom, je još jedan efikasan način ostvarivanja pomenutih zahtjeva. Podaci smješteni u klasteru bi takođe trebali biti za-

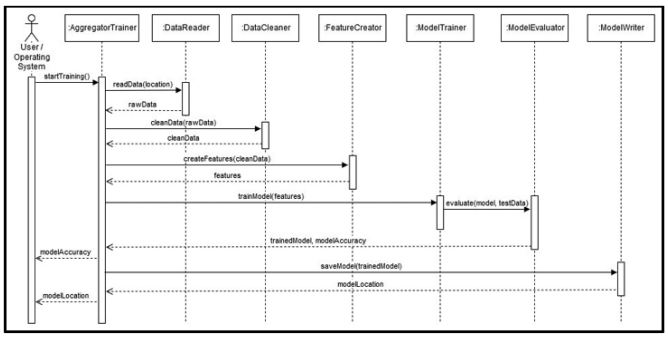
štićeni od pristupa koji nisu adekvatno autorizovani. Isto tako, komunikacija između čvorova klastera bi trebala biti zaštićena. Enkripcija se nameće kao rješenje ovih problema, gdje su podaci na diskovima kriptovani, što osigurava visok stepen sigurnosti i u slučaju nedozvoljenog pristupa računarskoj mreži ali i u slučaju nedozvoljenog pristupa samom hardveru klastera. Pristup određenim servisima *Hadoop*-a, kao i drugim komponentama ekosistema, bi jedino trebao biti dozvoljen korisnicima koji su uspješno autentikovani, i koji imaju odgovarajuća prava pristupa podacima i procesima. Istorija promjena klaster podataka, ostvariva kroz odgovarajuće mehanizme revizije (eng. *auditing*), bi trebala biti transparentna i dobro dokumentovana, kako bi zadovoljila striktne zakonske regulative [9].

5. PRIMJER UPOTREBE

Klase problema koje se mogu riješiti primjenom *Hadoop*-a, ili nekog drugog alata iz zaista velikog ekosistema, su poprilično brojne. Ipak, ono što im je svima zajedničko je činjenica da u jednoj ili više faza razvoja aplikativnih rješenja, potreba za skladištenjem i obradom velikih količina podataka prevazilazi mogućnosti konvencionalnih računarskih sistema. U nastavku se govori o jednom takvom rješenju, koje prvenstveno upotrebnom alata i biblioteka iz *Hadoop* ekosistema nastoji da riješi problem kategorizacije *Web* vijesti, korištenjem nekoliko principa mašinskog učenja (eng. *Machine Learning - ML*). Treba naglasiti da iscrpna analiza principa i algoritama mašinskog učenja nije u fokusu, i da je naglasak stavljen na ulogu *Hadoop*-a i njegovih podsistema u jednom ovakvom rješenju. Aplikativno rješenje ovog tipa bi bilo pogodno za automatizaciju procesa klasifikovanja *Web* vijesti, i kao takvo može biti sastavni dio nekog većeg informacionog sistema, npr. za agregiranje vijesti u *Web* okruženju.

Funkcionalni opis aplikacije

Aplikacija je zamišljena tako da na osnovu metapodataka o nekoj *Web* vijesti, pronađe odgovarajuću kategoriju, iz konačnog skupa dostupnih kategorija. Konkretno u ovom slučaju, postoje četiri kategorije vijesti: poslovne vijesti, označene slovom *b* (eng. *business*), vijesti iz oblasti nauke i tehnologije, označene slovom *t (*eng. *technology)*, vijesti iz svijeta zabave, označene slovom *e* (eng. *entertainment*), kao i vijesti iz oblasti medicine i zdravlja, označene slovom *m* (eng. *medicine*). Metapodaci o vijestima su brojni, ipak aplikacija koristi one najpogodnije za utvrđivanje odgovarajuće kategorije. Inforamcije o naslovu vijesti te objavljivaču su svakako bitni metapodaci koji utiču na samu kategoriju vijesti. Osim toga, informacija o *Web* portalu koji je objavio vijest kao i informacija o temi ili priči na koju se vijest odnosi su takođe vrlo bitni u cjelokupnom procesu kategorizacije. Sam proces implementacije aplikacije zasnovane na mašinskom učenju se uveliko razlikuje od procesa razvoja konvencionalnih korisničkih aplikacija. I dok na jednoj strani imamo, u manjoj ili većoj mjeri, jasno utvrđene korisničke zahtjeve, jasne procese realizacije istih kao i prethodno utvrđene načine testiranja rezultata, u slučaju ML aplikacija veliki dio razvoja se odrađuje empirijskim i eksperimentalnim postupcima. Kao podaci za treniranje i evaluaciju modela u najvećem broju slučajeva se koriste istorijski podaci, sakupljani tokome određenog perioda. Na osnovu ovih podataka, uz varijaciju različitih tehnika treniranja, nastoji se ostvariti što veća tačnost predviđanja (eng. *prediction*) kroz proces evaluacije modela. Tako treniran model se kasnije koristi za stvarna predviđanja, u konkretnom informacionom sistemu. Za potrebe ove aplikacije, trening podaci, kao i podaci za evaluaciju i testiranje modela, su sakupljani na već postojećoj platformi za agregaciju *Web* vijesti, tokom 2014. godine [10]. Veličina skupa podataka za evaluaciju i treniranje je oko 100 MB, i sadrži preko 400 000 unosa u CSV *(Comma-separated Values)* formatu. Ovaj skup podataka je tokom procesa implementacije podijeljen na dva skupa: skup trening podataka i skup testnih podataka. Uobičajena praksa je da trening podaci imaju veći udio, a koliki bi on tačno trebao biti, uglavnom se utvrđuje eksperimentalno, kroz više iteracija treniranja i evaluacije. Aplikacija za klasifikaciju vijesti se ugrubo može podijeliti na dva glavna dijela, od kojih je prvi dio namijenjen treniranju prediktivnog modela, *AggregatorTrainer*, pomoću pomenutog skupa podataka. Drugi dio, *AggregatorPredictor*, koristi rezultate prvog dijela, istreniran model, kako bi izvršio potrebnu klasifikaciju *Web* vijesti.

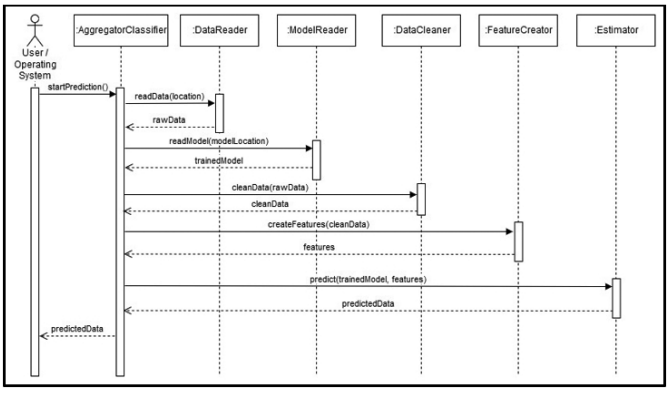


Slika 10. - Dijagram sekvence za proces treniranja prediktivnog modela

Na Slici 10. nalazi se dijagram sekvence, na kojem se vidi cjelokupan proces treniranja prediktivnog modela, kroz razmjenu poruka između pojedinih objekata i komponenti sistema. Zahvaljujući vremenskoj komponenti, kao sastavnom dijelu ovog tipa UML dijagrama, lako se uočava i redoslijed izvršavanja pojedinih dijelova aplikativne logike. Naravno, s obzirom na to da je za uspješno treniranja modela, potrebna što veća količina istorijskih podataka na raspolaganju, jasno se nameće i uloga *Hadoop*-a i HDFS-a u jednoj ovakvoj postavci. Trening podaci se čuvaju na HDFS-u, u CSV ili nekom drugom tekstualnom formatu, i po potrebi pomoću *MapReduce*-a ili nekog sličnog sistema, kao što je *Apache Spark*, transformišu i dovode u oblik pogodan za treniranje prediktivnog modela. Takođe je jasno da ova funkcionalnost sistema nije pogodna za izvršavanje u realnom vremenu, prvenstveno zbog vremenskih kašnjenja, prouzrokovanih obradom enormnih količina podataka. Osim toga, kod velikog broja ML aplikacija, proces treniranja modela nije kontinualan, dinamičan, i ne izvršava se stalno, već u određenim vremenskim trenucima kad za to postoji potreba. Koliko često i koliko dugo, uveliko zavi-

si od dostupnih podataka za treniranje, njihove količine kao i učestanosti njihove promjene. Kao što se može vidjeti sa Slike 10. jedan od izlaza procesa treniranja je istreniran prediktivni model, koji je sačuvan u trajnu menoriju, HDFS ili *Hive* tabelu, i koji može kasnije biti korišten u procesu kategorizacije vijesti. Drugi izlaz je tačnost modela, izražena u procentima, na osnovu koje se može utvrditi da li je model dovoljno dobar. Ukoliko zadovoljavajuća tačnost treniranog modela nije postignuta, proces obuke prediktivnog modela, sa korigovanim parametrima treniranja može biti ponovljen sve dok se željena tačnost predviđanja ne postigne.

Na Slici 11. je dat dijagram sekvence na kojem se vidi proces estimacije, korištenjem modela iz prethodne faze. Jasno se može vidjeti redoslijed izvršavanja pojedinih koraka u procesu klasifikovanja do tad neklasifikovanih *Web* vijesti. Po završetku postupka klasifikovanja, rezultati se mogu sačuvati na fajl sistem, prikazati korisniku na interfejsu komandne linije, ili što je i najčešći slučaj sačuvati u relacionu bazu podataka, odakle jednostavno mogu biti dalje konzumirani od strane drugih podsistema.



Slika 11. - Dijagram sekvence za proces estimacije (kategorizacije) vijesti

Primjer kategorizacije novih vijesti

Iako su ulazni podaci takođe smješteni u CSV formatu, zbog jednostavnosti prikaza i bolje preglednosti, ovdje su navedeni u tabelarnom formatu. Na Slici 12. nalazi se primjer 5 novih *Web* vijesti za koje je potrebno odrediti kategoriju, upotrebnom razvijene aplikacije za kategorizaciju vijesti. Treba napomenuti da sirovi, neprečišćeni, CSV ulazni podaci po pravilu sadrže više informacija o vijestima, i da je ovdje navedeno samo 5 ključnih *feature-a* korištenih u procesu treniranja modela. Jedinstveni identifikator vijesti, u ovom slučaju redni broj vijesti, nije korišten u procesu klasifikacije, i koristi se isključivo u svrhu lakše interpretacije rezultata izvršavanja.

	Naslov	Objavljivač	Tema	Portal	Datum objave
1	China exports decline	San Francisco Chronicle	dn8H6sYjpAZnsg	www.sfgate.com	1394490417
2	Snatch titanfall special	TheSixthAxis	dw0Jnfj8vEPaLZ	www.thesixthaxis.com	1394493817
3	Electronic cigarettes target youth	The Tribune-Democrat	dhXo4PyiryVJcP	www.tribune.com	1397780341
4	Reading to kids early is good	WLS-TV	dfsVzKmXH-GH	abc7chicago.com	1403660608
5	Bob Dylan Lyrics sold	Big News Network	dEN4U7C7gzXJb	www.bignews.com	1403655411

Slika 12. - Ulazne vijesti za proces klasifikacije

Pokretanjem aplikacije sa datim ulaznim podacima, istrenirani prediktivni model kao izlaz daje raspodjelu vjerovatnoća po kategorijama za svaku od ulaznih vijesti. Na Slici 13. je dat pregled izlaznih rezultata, gdje je svaki red identifikovan jedinstvenim identifikatorom vijesti. Zbog jednostavnosti prikaza, brojne vrijednosti su zaokružene na tri decimalna mjesta.

	t	m	e	b
1	0,039	0,001	0,015	0,945
2	0,898	0,084	0,001	0,017
3	0,065	0,901	0,002	0,032
4	0,005	0,890	0,005	0,100
5	0,034	0,004	0,886	0,076

Slika 13. - Izlazna raspodjela vjerovatnoća po kategorijama

Iako su navedeni primjeri kategorizacije *Web* vijesti čisto ilustrativne prirode, korišteni u svrhu demonstracije principa mašinskog učenja kao i cjelokupnih rezultata istraživanja, treba imati u vidu da bi implementirano rješenje potpuno identično funkcionisalo i u produktivnom *Hadoop* okruženju. Količina podataka za obradu bi bila daleko veća i reprezentativnija, pa bi samim tim i hardverski zahtjevi za uspostavljanje klastera bili znatno drugačiji. Određena degradacija performansi bi svakako bila očekivana, a kolika bi ona tačno bila zavisi od obima i strukture podataka kao i od veličine i kvaliteta hardverske infrastrukture. Kada je riječ o tradicionalnim sistemima za obradu i skladištenje, oni bi u slučaju manjih količina podataka zasigurno imali bolje performanse u odnosu na *Hadoop* sistem. S druge strane, ovakvi sistemi u većini slučajeva ne bi bili u mogućnosti skladištiti, a ni obraditi, velike količine podataka sa kojima *Hadoop* sistem radi.

6. ZAKLJUČAK

U ovom radu izvršena je detaljna analiza osnovnih pojmova, arhitektura, tehnika, tehnologija i metodologija, koje se u današnje vrijeme koriste pri razvoju informacionih sistema za obradu velikih količina podataka. Kroz detaljnu analizu *Apache Hadoop*-a i njegovih osnovnih komponenti, utvrđene su dobre i loše osobine svake od njih. Na taj način izolovani su tipovi aplikacija za čiju implementaciju je pogodno primijeniti *Hadoop* aplikativni *framework*. Zbog obima podataka koji se obrađuje, *Hadoop* rješenja nisu pogodna u situacijama gdje se očekuje odziv sistema u realnom vremenu. Takođe, zbog *MapReduce API*-ja poprilično niskog nivoa apstrakcije, *Hadoop* se nije dobro pokazao u praksi prilikom realizacije kompleksnih aplikacija, sa dužim periodima razvoja i održavanja.

U cilju otklanjanja ili ublažavanja navedenih nedostataka osnovnog *Hadoop* okruženja razvijeni su brojni pomoćni alati, koji se jednim imenom nazivaju *Hadoop* ekosistem. S posebnom pažnjom je obrađen i *Apache Spark,* čijom upotrebnom je omogućeno poboljšanje performansi u odnosu na *MapReduce*, kroz skladištenje međurezultata obrade u radnu memoriju. Fleksibilnost pri implementaciji *Spark job*-ova, koja se ogleda u mogućnosti definisanja proizvoljnih faza izvršavanja, je još jedan od razloga sve veće upotrebe ovog *framework*-a u realizaciji rješenja orijentisanih ka intenzivnim obradama.

Kroz praktični dio rada analiziran je jedan od mogućih slučajeva upotrebe *Apache Hadoop* aplikativnog *framework*-a i

pojedinih dijelova njegovog ekosistema, prevenstveno *Apache Spark*-a. Realizovana je relativno kompleksna aplikacija za kategorizaciju *Web* vijesti, korištenjem *Spark*-ove biblioteke za mašinsko učenje. Treba napomenuti da dobijeni rezultati analize uveliko zavise od dostupnog trening skupa podataka, kao i dostupne hardverske infrastrukture, koji su u ovom slučaju bili minimalni. U slučaju realnog informacionog sistema, gdje se skladišti i obrađuje daleko više podataka i gdje postoji optimalna hardverska infrastruktura, prednosti *Hadoop* ekosistema bi se mogle u potpunosti iskoristiti. Samim tim i dobijeni rezultati istraživanja bi bili drugačiji. Tačnost prediktivnog modela bi zasigurno varirala, s obzirom na dostupnost veće količine podataka za proces treniranja. Promjena u performansama bi također bila uočljiva, gdje bi vrijeme izvršavanja direktno zavisilo kako od količine podataka koji se obrađuju, tako i od broja čvorova unutar klastera, ali i same hardverske infrastrukture na kojoj je klaster izgrađen.

LITERATURA

[1] The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, https://www.emc.com/leadership/digital-universe/2014iview/digital-universe-of-opportunities-vernon-turner.htm, 2018
[2] David Reinsel, John Gantz, John Rydning, The Digitalization of the World, IDC White Paper, 2018
[3] Chuck Lam, Hadoop in Action, Manning Publications Co. Greenwich, CT, USA, 2010.
[4] Alex Holmes, Hadoop in Practice, Manning Publications, 2012
[5] Apache Spark, http://spark.apache.org, 2018
[6] Tom White, Hadoop: The Definitive Guide 4th Edition, O'Reilly, 2015
[7] Apache Zookeeper, https://hortonworks.com/apache/zookeeper, 2018
[8] Cloudera Reference Architecture, https://www.cloudera.com/documentation/other/reference-architecture.html, 2019
[9] Cloudera Security, https://www.cloudera.com/documentation/enterprise/latest/topics/security.html, 2019
[10] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2019
[11] Apache Spark FeatureHasher, https://spark.apache.org/docs/2.3.0/ml-features.html#featurehasher, 2019
[12] HBase, https://hortonworks.com/apache/hbase/, 2018

Dragan Šušak, dipl. ing. diva-e Platforms GmbH, Munich, DE
Kontakt: susakdragan@gmail.com
Oblast interesovanja: mašinsko učenje, data science, objektno-orijentisano programiranje i modelovanje, Internet programiranje, razvoj Web aplikacija, Web servisi, sigurnost, sistem integracija

prof. dr Zoran Đurić, Elektrotehnički fakultet, Univerzitet u Banjoj Luci, RS, BiH
Kontakt: zoran.djuric@etf.unibl.org
Oblast interesovanja: sigurnost, kriptografija, PKI, platni sistemi i protokoli, formalna verifikacija, mašinsko učenje, data science, objektno-orijentisano programiranje i modelovanje, Internet programiranje, razvoj mobilnih aplikacija, XML-bazirana međuoperativnost, Web servisi, računarske mreže, penetration testing, sistem integracija

infoM

UPUTSTVO ZA PRIPREMU RADA

1. Tekst pripremiti kao Word dokument, A4, u kodnom rasporedu 1250 latinica ili 1251 ćirilica, na srpskom jeziku, bez slika. Preporučen obim – oko 10 strana, single prored, font 11.
2. Naslov, abstakt (100-250 reči) i ključne reči (3-10) dati na srpskom i engleskom jeziku.
3. Jedino formatiranje teksta je normal, bold, italic i bolditalic, VELIKA i mala slova (tekst se naknadno prelama).
4. Mesta gde treba ubaciti slike, naglasiti u tekstu (Slika1...)
5. Slike pripremiti odvojeno, VAN teksta, imenovati ih kao u tekstu, radi identifikacije, u sledećim formatima: rasterske slike: jpg, tif, psd, u rezoluciji 300 dpi 1:1 (fotografije, ekranski prikazi i sl.), vektorske slike – cdr, ai, fh,eps (šeme i grafikoni).
6. Autor(i) treba da obavezno priloži svoju fotografiju (jpg oko 50 Kb), navede instituciju u kojoj radi, kontakt i 2-4 oblasti kojima se bavi.
7. Maksimalni broj autora po jednom radu je 5.

Redakcija časopisa Info M