ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА СТРУКОВНИХ СТУДИЈА

**Живојиновић Марко РИН-26/19**

**Размена података у оквиру Oracle BPM архитектуре**

**- примењени истраживачки рад -**



Београд, јун 2021.

САДРЖАЈ:

[1. Увод 2](#_Toc41758052)

[2. *APACHE HADOOP* 3](#_Toc41758053)

[2.1. *Hadoop Distributed File System (HDFS)* 3](#_Toc41758054)

[2.2. *YARN* 7](#_Toc41758055)

[2.3. *MapReduce* 8](#_Toc41758056)

[3. *Hadoop* ekosistem 13](#_Toc41758057)

[3.1. *Pig* 14](#_Toc41758058)

[3.2. *Hive* 14](#_Toc41758059)

[3.3. *Spark* 15](#_Toc41758060)

[3.4. *HBase* 16](#_Toc41758061)

[3.5. *Drill* 17](#_Toc41758062)

[3.6. *Zookeeper* 18](#_Toc41758063)

[3.7. *Oozie* 18](#_Toc41758064)

[3.8. *Flume* 18](#_Toc41758065)

[3.9. *Sqoop* 19](#_Toc41758066)

[3.10. *Solr & Lucene* 20](#_Toc41758067)

[3.11. *Ambari* 20](#_Toc41758068)

[4. Преглед постојећих решења 22](#_Toc41758069)

[5. Анализа Твитер објава у *Hadoop* систему 24](#_Toc41758070)

[5.1. Опис компоненти апликације 24](#_Toc41758071)

[5.2. Развој апликације 25](#_Toc41758072)

[6. Закључак 31](#_Toc41758073)

[7. Литература 32](#_Toc41758074)

# Увод

Када се говори о имплементацији сложених решења намењеним за функционисање великих пословних система, постоје разне стратегије којима се може доћи до жељеног решења оваквих система. Пре свега, питање је за коју платформу је намењено софтверско решење за рад оваквих система. Данас се грубо према платформи за коју су намњене, апликације могу поделити на стандардне десктоп апликације, које се затим морају инсталирати на сваки кориснички уређај да би се омогућио рад крајњим корисницима, андроид апликације које су намењене првенствено мобилним уређајима и веб апликације. У данашње време све чешће се прибегава изради веб апликација које имају задатак да омогуће рад у датом пословном систему. Један од главних разлога зашто су веб апликације све популарније за коришћење јесте независност оперативног система корисничког уређаја од самих апликација. Све што је неопходно да би се приступило апликацији и користиле њене функционалности јесте веб претраживач који поседује готово сваки кориснички уређај данас.

На почетку развоја веб апликација, потребно је размислити о избору програмског језика, односно технологија (*framework*) које ће бити коришћене за израду апликације. Поред овога, битно је размислити о начину организације и чувања података, најчешће у оквиру одређене релационе базе података. Уколико се одабере овакав правац развоја, програмери имају потпуну слободу у коришћењу одабраних технологија и њиховог комбиновања у циљу израде веб апликација различитих намена. Међутим, уочено је да се често многи пословни системи могу свести на опис активности које корисници изводе у оквиру одређених процеса. Због ове чињенице, многе софтверске компаније су развиле своје BPM(*Business Process Management*) софтверске имплементације које имају намену да олакшају развој решења за разне пословне системе. У оквиру овог рада ће бити описан начин рада у оквиру OracleBPM платформе и поређење са другим BPMсистемима.

# BPM системи

Типичан начин рада у оквиру готово било које фирме, без обзира на делатност којом се бави се може описати кроз пословне процесе који се одвијају у оквиру те фирме. BPM служи како би се ови процеси што прецизније описали, и уочила њихова евентуална повезаност, као и комплетан опис свих активности које се могу изводити у оквиру њих. Дакле, BPM као појам није конкретно везан за развој софтверских решења која служе за израду пословних процеса који се одвијају у оквиру неке фирме. BPM представља технику која има своја правила и принципе помоћу којих се дефинишу пословни процеси. Када се говори о побољшању пословања одређене фирме, кроз BPM се не може утицати на побољшање конкретних активности који запослени изводе у оквиру неког процеса. Заправо, кроз BPM се дефинише целокупни пословни систем и његови међусобно повезани елементи. Елементи у оквиру BPM су дакле одређени догађаји који се могу уочити, активности и одлуке. Управо ови елементи су кључни фактори у сваком пословном процесу који одређују његов ток.

## Пословни процеси

У сваком привредном субјекту, фирми, организацији се одвијају процеси који дефинишу пословање. Типични примери процеса који се могу видети у оквиру фирми могу бити:

* Наруџбина – плаћање: Процес започиње трговац, када клијент поручи одређени производ, или закаже услугу, а завршава се када се дати производ испоручи и обави плаћање. Овај тип процеса садржи активности које су везане за верификацију наруџбине, слање нарученог производа, испоруку и уручење рачуна и потврду од стране клијента.
* Жалба клијента – решење проблема: Овај процес започиње у моменту када клијент покрене жалбу због незадовољства пруженом услугом, односно купљеним производом. Процес се не завршава док се клијент, фирма, а у најбољем случају обоје не сложе да је проблем решен.
* Подношење захтева – одобрење: Још једна доста честа врста процеса, који почиње када особа подноси пријаву везану за одређену потребу, који се може завршити одбијањем или одобрењем. Ова врста процеса се може срести у оквиру разних државних служби где грађани подносе захтеве, као што је издавање грађевинске дозволе, на пример.

Како видимо у горе наведеним примерима, пословни процеси су управо оно што се одвија у оквиру фирми и организација када је потребно извршити интеракцију са крајњим корисницима. Начин на који су ови процеси организовани и изведени одређује квалитет услуге који се пружа клијентима, као и ефикасност целокупног пословања једне фирме.

## Компоненте пословног процеса

Како се види у претходном поглављу, пословни процеси се најчешће састоје из више догађаја (*event*) и активности (*activity*). Догађаји се односе на ствари које се у неком моменту догоде, и немају предвиђено временско трајање. Догађаји могу покренути серију активности које следе након њиховог дешавања. На пример, када се деси догађај да клијент поднесе захтев за повраћај новца, запослени има задатак да спроведе активност прегледа описа захтева који је клијент упутио.

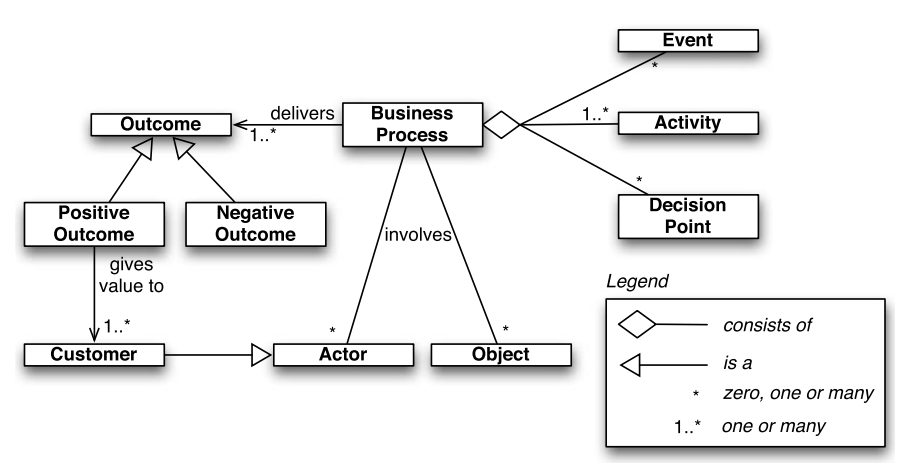
Када је потребно да човек мануелно изврши одређени задатак у оквиру активности, најчешће у смислу одређивања садржаја података који му долазе у датој активности, онда се та активност назива задатак (*task*). На нивоу задатака, потребно је одредити који запослени ће извршити конкретан задатак. Одређени задатак се не мора увек доделити конкретном запосленом, већ се може доделити одређеној корисничкој групи која постоји у оквиру фирме.

Поред догађаја и активности, процеси у главном садрже и тачке одлучивања – *decision points*. Тачке одлучивања представљају „раскрсницу“ у оквиру процеса, односно којим путем ће се наставити извршење датог процеса, најчешће у зависности од вредности неког податка. Често вредност податка који одређује даљи ток процеса бива одређена од стране запосленог који извршава одређену активност.

Процес садржи и одређене чиниоце поред претходно наведених фактора. Ови чиниоци могу бити људи, организације, или софтвер који аутоматизовано извршава задатке постављене пред њега. Процеси дакле садрже најчешће предмете пословања којима се одређена фирма бави, односно њихове описе у виду података, а такође се кроз њих могу размењивати документи који су претходно направљени.

Завршетак процеса у може имати један или више исхода. Уколико се ради о процесу као што је захтев за повраћајем новца од стране клијента, може се десити да се процес успешно заврши (гледано са стране клијента) тако што ће му бити повраћен новац. Могућ исход је и да се деси одбијање од стране фирме, тако да новац неће бити враћен клијенту. Међутим, може се десити и да завршетак једног процеса представља услов за започињање неког новог који је условљен претходним. Ово значи, на датом примеру да се клијент може у неком моменту поново пожалити, а компанија ће имати забележен претходни случај жалбе клијента, па на основу њега у неком моменту одредити ток процеса.

На слици 1. је приказана типична структура једног пословног процеса. Као што је већ речено, један процес се може састојати од неколико догађаја, тачака одлучивања, а макар једне активности. Процес унутар себе укључује чиниоце (људи, групе људи, софтвер...), објекте (документа, подаци...), а на излазу даје неки од могућих исхода који даје одговор крајњем клијенту. Овај скуп компонената представља основну структуру која се може применити на многе видове пословања, односно делатности и помоћу које се могу описати сви различити пословни процеси који се одвијају у оквиру ових делатности.



Слика 1. Компоненте пословног процеса [1]

# Софтверске платформе за BPM

Како се види у претходном поглављу, само моделовање пословних процеса је заправо је идејно решење како треба пословање једне компаније да се одвија у циљу успешног пословања. Данас постоје бројна софтверска решења за моделовање пословних процеса. У основи, сва ова решења садрже неки алат за дефинисање процеса који ће се одвијати. Њихове генералне карактеристике су на први поглед сличне, али је једно од кључних питања да ли ће одређен систем бити компатибилан са постојећим окружењем. Опште је познато да различити софтверски производи и решења захтевају различиту софтверску и хардверску подршку. Под софтвером се најчешће подразумева оперативни систем, или сервер уколико се ради о веб апликацијама. Свако софтверско решење се базира на одговарајућој технологији односно програмском језику. Фирмама у којима се постојећа решења базирају на *PHP* технологији и *Apache* серверу погодује имплементација BPMрешења које подржава ову врсту технологија. Због овога је битно препознати који тип технологија се тренутно имплементира у оквиру компаније и која BPM платформа је најпогоднија за примену у постојећу инфраструктуру. Такође, BPM платформе се ослањају на одређену релациону базу података у којој се чувају сви подаци који се тичу процеса и других активности. Нека BPM решења подржавају MySQL, нека Oracle, а нека SQL Server базе података. Најчешће се унутар фирми примењује BPM решење које подржава постојећи тип базе података.

Модерна решења BPM платформи се најчешће базирају на сервисно оријентисаној архиви. SOA (*Service Oriented Architecture*) подразумевају концепт развоја сервиса који се могу користити од стране различитих апликација. Ово значи да се пословни процес у оквиру BPM платформе може иницирати, односно контролисати чак из апликација које су независно инсталиране у постојећем окружењу.

Такође, поред алата за моделовање пословног процеса који је потребно одабрати у складу са жељеном BPM платформом, потребно је одабрати језик за моделовање процеса (*modeling language*). Овај језик, слично било ком програмском језику садржи сопствену семантику и синтаксу. Једна од кључних карактеристика BPM алата за моделовање јесте да подржи неки од популарних језика за моделовање процеса, као што су BPMN 2.0, BPQL, BPEL, EPC, UML, XML… Неке BPM платформе подржавају више језика за моделовања, док неке подржавају само један.

Сва модерна BPM решења најчешће садрже и командну таблу – *dashboard*. Овде администратори система могу да виде како се пословни процеси одвијају, као и да извуку одогварајуће анализе. Командне табле имају задатак да омогуће надгледање процеса у реалном времену, као и генерисање извештаја и графикона како би се приказао ток текућих пословних процеса. Командна табла је једна од кључних ствари како би се видела ефикасност обављања пословних процеса, као и како би се размотриле могућности даљег унапређења њихове имплементације и организације.

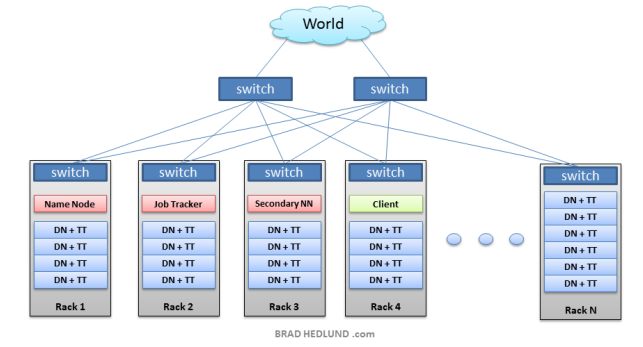
# Oracle BPM

Као што је већ речено, данас су софтверске компаније развиле бројна BPM решења за дизајнирање односно моделовање пословних процеса. Један од могућих избора је свакако Oracle BPM платформа. Ова платформа омогућава развој велике палете различитих апликација које заједно функционишу и размењују податке кроз њихов централни део – пословни процес. Oracle као светски позната компанија већ поседује доста признатих решења у оквиру рачунарских система и информационих технологија. Једна од најпознатијих релационих база података јесте Oracle база података, на којој се управо базира и њихова BPM архитектура пословних система. Целокупан систем се базира на *Java* програмском језику и пратећим *Java EE* веб технологијама.

Да би се ушло у дубља разматрања функционисања овог система неопходно је разумети основе стандардних *Java EE* веб апликација. Код стандардних *Java EE* веб апликација, могуће је одабрати неки од актуелних сервера који подржавају инсталацију апликација на њих. Као један од најпознатији свакако јесте *Apache Tomcat* сервер, који се вероватно најчешће и помиње у стручној литератури. У основи, сваки сервер (гледано са софтверске стране) поседује одређена конфигурациона подешавања, и језгро које омогућава саму инсталацију веб апликација на њега. *Java EE* технологије су главним делом сконцентрисане на рад са *Java Servlet*

Како смо већ видели, *Hadoop* представља један систем за обраду велике количине података кога карактерише дистрибуирана инфраструктура. Развијен је од стране фондације „Apache“. Омогућава корисницима да развијају дистрибуиране апликације, а да корисници не морају у потпуности да знају шта се „у позадини“ дешава. Главне целине које сачињавају *Hadoop* систем се могу поделити на *HDFS (Hadoop Distributed File System)* и *MapReduce* модел. *HDFS* нуди високе могућности складиштења података, док *MapReduce* омогућава високе перформансе при обради ових дистрибуираних података. Захваљујући томе што је сам *Hadoop* систем отвореног кода (*open-source*), модел *MapReduce* и *HDFS* су постали саставни део многих организација и компанија. Због популарности ових *Hadoop* технологија, стално се развијају и нови алати и технологије које су развијене на темељу *Hadoop* система.

## *Hadoop Distributed File System (HDFS)*

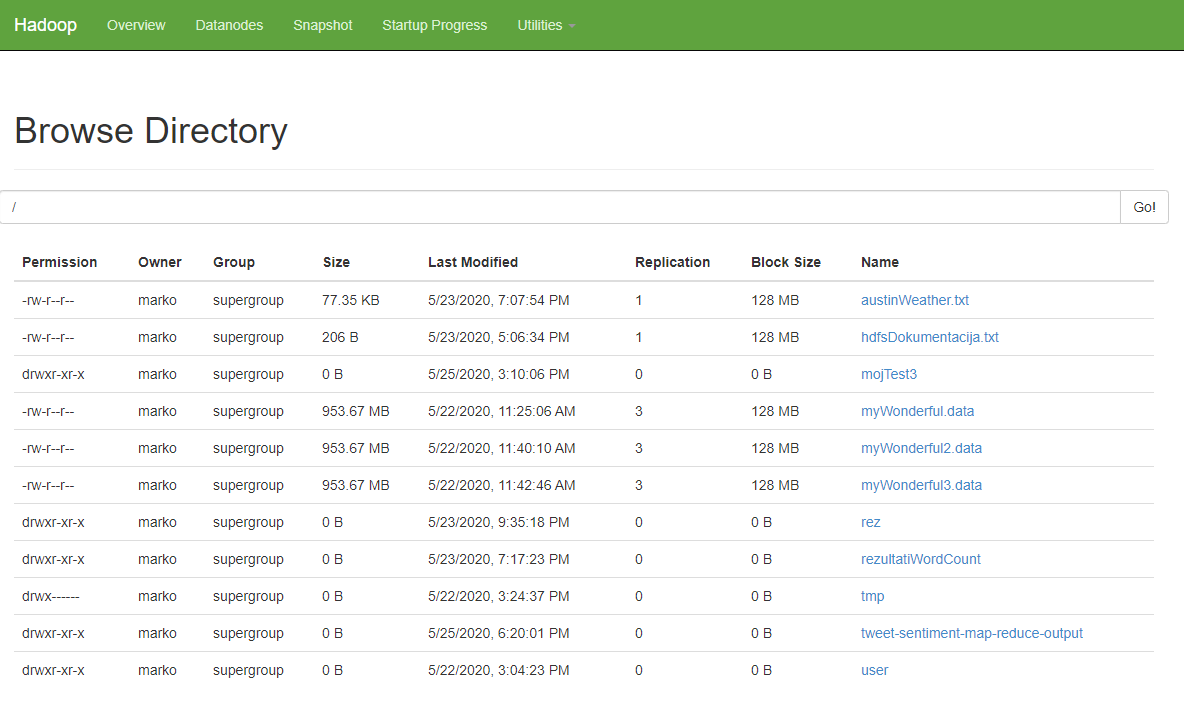


Слика 2. *Hadoop Cluster* [2]

На слици кластера су унутар рафова приказани су поред кључних чворова везаних за само складиштење велике количине података приказани и чворови који служе да би се извршио конкретан упис одређених података на кластер или да захтевају извршење различитих послова у виду обраде података итд. Чворови који обављају ове функционалности се називају *Clientnode* чворови. Они имају инсталиран *Hadoop*, са свим подешавањима везаним за конфигурацију самог кластера, али не представљају ни *master* чвор, а ни *slave* чвор. *Clientnode* чворови се свакако не могу директно обратити *Namenode* чворовима да би захтевали извршење одређеног задатка, већ то раде преко посредничког сервиса *JobTracker* који има улогу да обради ове захтеве, који су најчешће типа *MapReduce*, о чему ће бити речи касније.

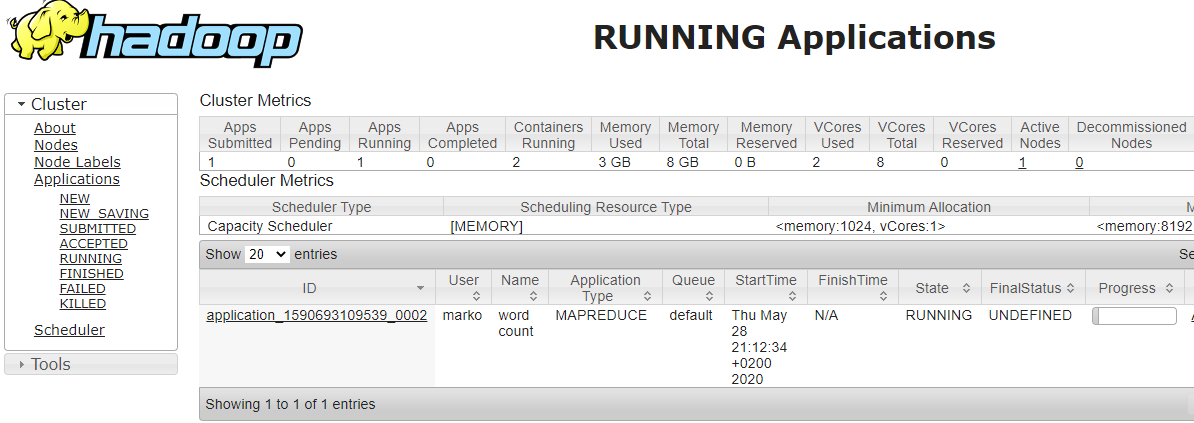
Када говоримо о смештању блокова података и њиховом реплицирању на кластеру, једноставно решење јесте да се реплике блокова смештају на јединственим рафовима. Ово решење спречава губитак података у случају и да цео раф „падне“, односно престане да исправно функционише. Такође, ово решење омогућава бржи проток при читању података, с обзиром да се подаци могу читати у паралели са различитих рафова. Дакле, блокови података, односно њихове реплике су равномерно распоређени на кластеру. Међутим, ово решење смањује брзину уписа података, због тога што процес уписивања мора да пребаци блокове података на више различитих рафова, а већ је објашњено да је комуникација између различитих рафова спорија.

Често се примењује другачије решење када је фактор реплицирања података 3. *HDFS* складишти податке тако што једну реплику блока података на једном чвору унутар рафа, другу реплику података у неком другом рафу, а последњу реплику на неки други чвор унутар истог рафа. Због овог решења, осетно су боље перформансе уписивања података. Шансе за отказивање рафова су далеко мање од отказивања појединачних чворова. Дакле, у овом решењу, блокови података нису равномерно распоређени на кластеру, већ су организовани тако да се једна трећина реплика налази на једном чвору, друга трећина на том рафу, а остатак је дистрибуиран на остале рафове унутар кластера. Ово решење побољшава перформансе уписивања без угрожавања поузданости система или перформанси читања.



Слика 3. Структура *HDFS* система

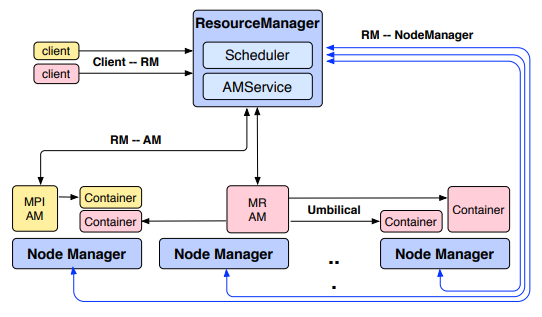
*HDFS* подразумевано нуди неколико веб интерфејса за надгледање целокупне структуре свог система датотека, као и извршавање тренутних послова на систему. Ови интерфејси се подразумевано налазе на адресама [*http://localhost:50070/explorer.html*](http://localhost:50070/explorer.html) и [*http://localhost:8088/cluster*](http://localhost:8088/cluster). На првој адреси се налази графички кориснички интерфејс који приказује структуру датотека и директоријума на целокупном систему, са наведеним бројем репликација, величином итд. На другој адреси се налази графички интерфејс за праћење пре свега извршавања послова који се обављају у *Hadoop* систему. Послови који се извршавају најчешће су *MapReduce* апликације, које могу да се прате у зависности у ком стању извршења се налазе. (*New, Submitted, Running…*) *Hadoop* решење се у пракси чешће реализује у оквиру *Linux* окружења, односно оперативног система, мада је могуће уз одређено подешавање конфигурационих датотека и подесити рад система на *Windows* платформи.



Слика 4. Праћење *Hadoop* апликација

## *YARN*

*YARN (Yet Another Resource Negotiator)* представља посебан *framework* *Hadoop* система, који олакшава извршавање и праћење дистрибуираних апликација било ког типа на *HDFS*. Омогућава неопходне процесе (*daemons*) за праћење послова који треба да се изврше на систему и обрађује их. Надгледа извршење свих послова, управља ресурсима, односно утрошком меморије и процесорске снаге за извршавање одређених апликација на *Hadoop* систему. У првим верзијама *Hadoop* система, *YARN* уопште није постојала, већ су њене функционалности биле интегрисане у оквиру саме *MapReduce* компоненте. *MapReduce* је обављао више функционалности, које су горе наведене, дакле надгледање утрошка меморије и процесорске снаге, обрада самих података са система итд. *YARN* је од верзије 2.0 *Hadoop* система постао независан део задужен пре свега за управљање ресурсима. [6] *YARN* може да покреће, надгледа и обезбеђује ресурсе за *Hadoop* апликације различитих типова, а не само *MapReduce* апликације, мада су оне најчешће. Првобитне верзије *Hadoop* система су подржавале само *MapReduce* апликације. Додавањем независне *YARN* компоненте која је задужена за обављање горе наведених функционалности, *Hadoop* систем је постао још скалабилнији, лакши за праћење и заказивање послова који треба да се изврше.

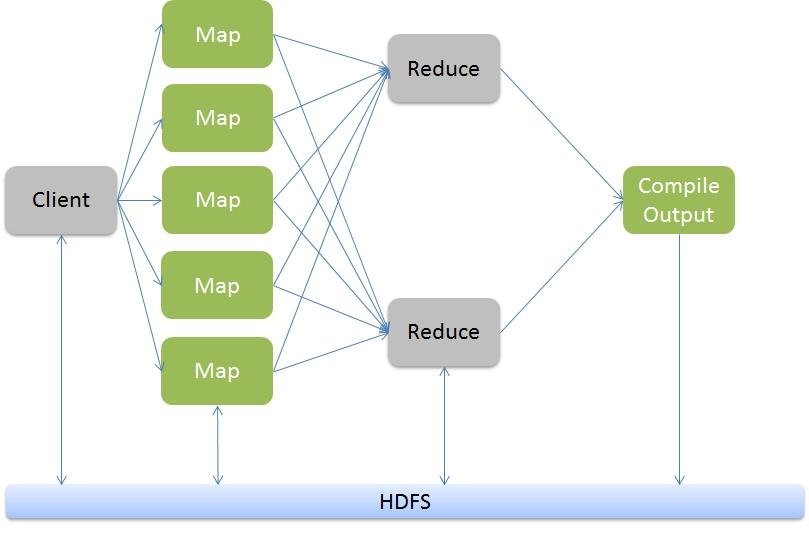


Слика 5. *YARN* архитектура [6]

Сам *YARN* се састоји из неколико компоненти: *Resource manager*, *Node manager*, *Application master* и *Resource containters*. Захваљујући овим компонентама, извршавање и праћење послова је децентрализовано и боље организовано, јер свака компонента обавља свој део задатка. *Resource manager* обавља задатке везане за прихватање послова од корисника, заказивање и додељивање ресурса неопходних за обављање задатих послова. *Node manager* компонента представља *slave* компоненту у овом случају, која је инсталирана на сваком чвору, и има задатке надгледања извршења послова и обавештавања главне *Resource manager* компоненте о стању послова. *Application master* компонента се генерише за сваку апликацију, како би преговарала о неопходним ресурсима везаним за функционисање апликације, као и рад са *Node manager* компонентом. Последња компонента *Resource contianers* служи како би се логички означио део ресурса које користи свака апликација.

## *MapReduce*

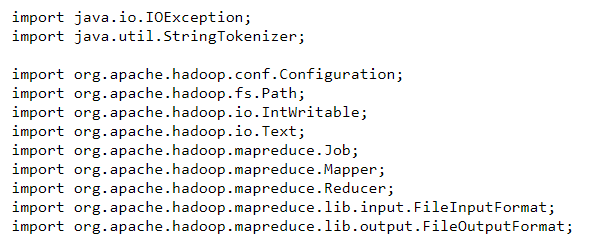
*MapReduce* је представљен 2004. године од стране компаније „Google“. Данас се користи широм разних послова везаних за обраду велике количине података. Користи се у обради задатака везаних за *Big data* аналитику, сортирање података, претраживање итд. *MapReduce* је кључна технологија којом располаже и *Hadoop* систем. Омогућава паралелно извршавање различитих задатака над великом количином података и даје могућност имплементације сопствених решења од стране програмера. [3]



Слика 5. Реализација *MapReduce* у *HDFS* систему [5]

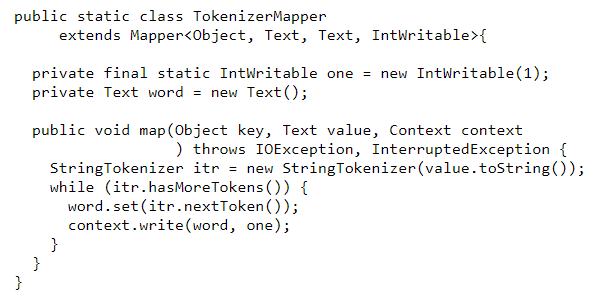
*MapReduce* представља програмски модел за обраду података. Користи се у разним програмским језицима, попут језика *Lisp*. Унутар *Hadoop* система, реализован је као софтверски *framework*, намењен за лако писање апликација које обрађују велике количине података, паралелно на великим кластерима (хиљаде чворова) са различитим хардверским конфигурацијама. Обезбеђује поуздано извршавање задатака, који најчешће спадају у разна претраживања, сортирања и обраду података. Када је започето извршавање операције *MapReduce*, тај процес се назива *job*. Подаци које обрађује *MapReduce* се чувају најчешће у оквиру самог *HDFS* система, а и резултат који се враћа се такође чува у оквиру *HDFS*. Сам модел *MapReduce* се састоји из две фазе. У обе фазе, улазни и излазни подаци се обрађују у виду уређених парова кључ-вредност. Из његовог имена се може закључити које су то фазе. Прва фаза представља фазу мапирања података – *Map*. Мапирање представља поступак читања улазних података из датотеке и формирања уређених парова који се даље прослеђују *Reduce* фази на обраду. У литератури се овај поступак најчешће објашњава на примеру *WordCount* програма, који је реализован помоћу овог *framework* – а. Задатак овог програма је да као резултат да одговор колико има којих речи у тој датотеци. Програм има два обавезна аргумента која прима – путању до датотеке која се налази на *HDFS* систему и путању до жељеног директоријума који ће бити формиран на систему, у коме ће бити смештени излазни резултати. У наставку текста биће дат опис рада програма *WordCount*, и описан начин имплементацијe *MapReduce* функционалности.

Када се на рачунар инсталира *Hadoop*, у оквиру његових директоријума се налазе библиотеке за укључивање у оквиру стандардних *Java* апликација. Такође, ту су и готове *.jar* датотеке као примери које нам *Apache* организација даје да тестирамо функционалности *Hadoop* система. За случај *WordCount* програма, потребно је укључити библиотеке приказане на слици 3.



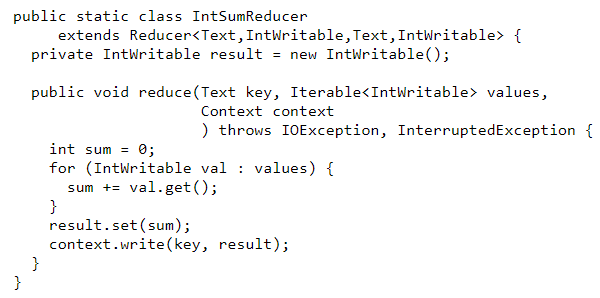
Слика 6. Укључивање библиотека [4]

Ове библиотеке садрже неопходне класе за конфигурисање наших класа које треба да испројектујемо да врше функционалности *Map* и *Reduce*. Да би корисничка класа имплементирала функционалност *Map*, неопходно је да буде реализована тако да наслеђује класу *Mapper*, којој се приликом навођења наслеђивања прослеђују 4 параметра. Ови параметри се односе на типове вредности које наша класа добија на улазу и које прослеђује на излаз. Како је већ речено, овде се увек ради о подацима типа кључ-вредност, тако да се прва два параметра односе на улазне аргументе, а друга два на излазне вредности, респективно. Идеја програма за бројање понављања речи унутар датотеке јесте да се у фази мапирања читају *String* подаци унутар датотеке, и за сваку прочитану реч се додељује вредност „1“.



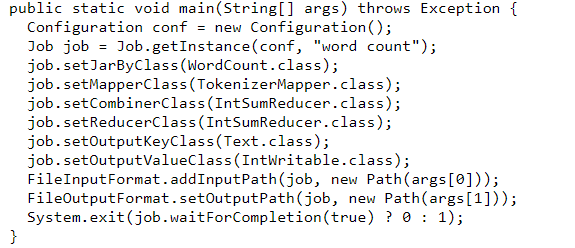
Слика 7. Имплементација *Mapper* класе [4]

На почетку је неопходно дефинисати две променљиве типа *IntWritable* и *Text*. Ове променљиве служе управо да би се мапирале вредности за сваку реч, и проследиле даље на *Reduce* фазу. Класа која има улогу мапирања мора да имплементира методу *map* која практично дефинише реализацију тачног поступка мапирања. Ова метода најчешће поред улазних вредности кључ-вредност (*key-value*) садржи и аргумент типа *Context*. Аргумент *context* представља посредника између фаза *Map* и *Reduce*.



Слика 8. Имплементација *Reducer* класе [4]

Што се тиче имплементације класе *Reducer*, она се реализује на сличан начин као и горе наведена класа *Mapper*. Такође је потребно приликом наслеђивања навести које типове улазних односно излазних аргумената ова класа треба да прими, што је у директној вези са излазом класе *Mapper*. Неопходно је да улазни аргументи кључ-вредност класе *Reducer* буду истог типа као и излазни аргуметни кључ-вредност класе *Mapper*. Даље, неопходно је дефинисати методу *reduce*, која прима као аргументе управо излазне вредности *Mapper* класе, односно методе *map*. У случају програма за бројање понављања речи у датотеци, методa *reduce* се реализује тако да има дефинисану променљиву за суму, која се инкрементира за сваку реч, и враћа суму одређене речи као излаз. По аутомазитму, дефинисано је да се резултати *Reducer* класе, односно практично целокупне *MapReduce* операције складиште у одређеном директоријуму на *HDFS* систему, који корисник треба да наведе у такозваној *Driver* класи. У примеру *WordCount* програма, ова подешавања су конфигурисана директно унутар *main* методе програма, приликом његовог покретања.

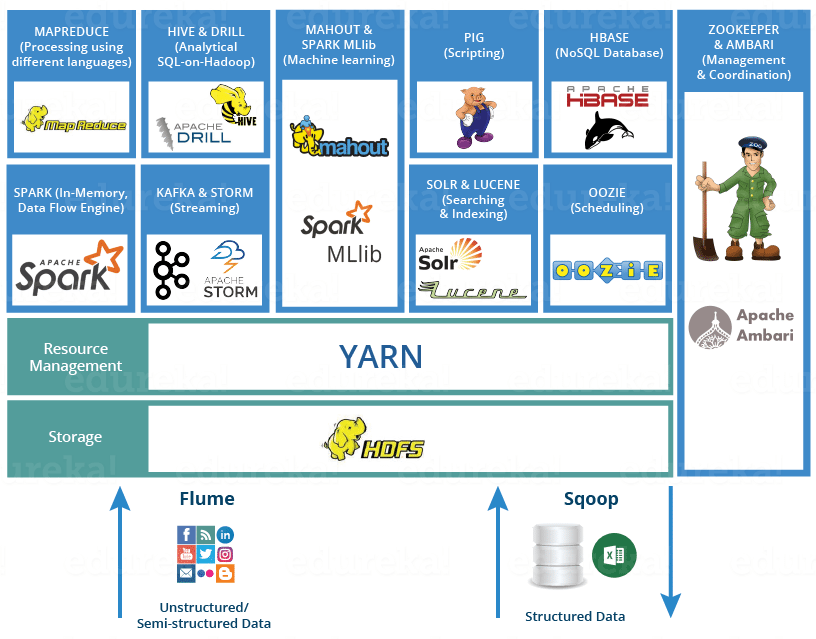


Слика 9. Имплементација *Driver* подешавања [4]

Променљива типа *Configuration*, практично представља референцу на тренутну везу са *HDFS* системом. Да би овај „посао“ извршења операције *MapReduce* успешно могао да се прати, и да би могао уопште да се покрене, дефинише се променљива *Job*, који мора да садржи информације управо о тренутној конфигурацији, односно *HDFS* систему, као и назив самог посла који треба да се обави. Даље је неопходно подесити које класе су дефинисане да обављају *MapReduce* задатке, као и који су типови излазних вредности. Помоћу статичких метода *addInputPath* и *setOutputPath* наводе се путање до *HDFS* датотеке чији садржај треба да се прочита, тј. анализира, као и излазни директоријум где ће бити сачувани резултати операције.

# *Hadoop* ekosistem

Мада је сам *Hadoop* веома робустан и скалабилан и у стању да реши већину проблема, он у свом основном окружењу постаје врло брзо неодржив у случају великих апликација. Програмски код потребан за дефинисање фазе мапирања и редуковања у овим случајевима постаје веома комплексан, неразумљив и непрегледан, поготово када је реч о апликацијама које се извршавају и одржавају дужи низ година. [7] Приступ подацима на *HDFS* систему помоћу основних функционалности *Hadoop* дистрибуираног система датотека је такође врло ограничен и непрактичан у случају комплекснијих случајева употребе. Из ових разлога, након настанка самог *Hadoop* развили су се бројни помоћни алати и подсистеми засновани на њему, који у великој мери поједностављују и убрзавају развој комплекснијих апликација, а и додатно олакшавају процес одржавања. Сви ови алати се у мањој или већој мери ослањају на сам *Hadoop* и сви они заједно представљају појам који се назива *Hadoop* екосистем.

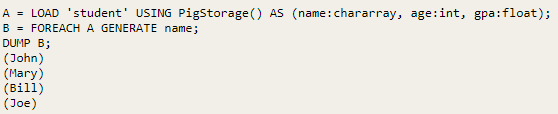


Слика 10. Архитектура *Hadoop* екосистема [8]

Генерално, алати који чине овај екосистем се највише тичу две делатности, а то су технике приступа подацима (*Data access*) и технике управљања подацима (*Data management*). Сви ови алати у главном имају богату подршку за програмски језик *Java*, а поред њега, пошто се доста користи програмски језик *Python* у области рада са великом количином података, постоје и библиотеке развијене и овај програмски језик. Сви ови алати се међусобно могу допуњавати и надограђивати један на други, због тога што им је свима заједничка основа управо само језгро *Hadoop* система, које представљају већ наведене компоненте – *YARN*, *MapReduce* и *HDFS*.

## *Pig*

*Pig* је алат намењен анализи великих количина података. Састоји се из две главне компоненте. Прва компонента јесте платформа која је неопходна да би се на њој извршавали *Pig* програми. Када кажемо *Pig* програми, већ видимо да се ту ради и о засебном програмском језику, који је *script* типа, и назива се *PigLatin*. Овај језик се користи управо да би се извршавали програми и вршила анализа великих количина података. Овај језик веома подсећа на класичан *SQL* језик за писање упита над стандардним базама података. Што се тиче програмера, он добија веома једноставну структуру писања наредби у језику *PigLatin*, док у позадини компајлер заправо цео код преводи у *MapReduce* посао који треба да се изврши.

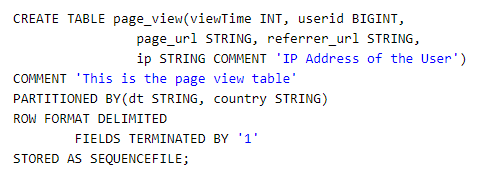


Слика 21. Пример *PigLatin* скрипте

Слично другим програмским језицима, наредбе се морају завршавати знаком тачка-зарез. Генерално, основна структура сваког упита написаног на *PigLatin* језику је следећа. Прво иде *LOAD* наредба која служи да се прочитају подаци складиштени на систему. Затим иду жељене наредбе везане за обраду прочитаних података односно њихову трансформацију. Крај упита се увек означава наредбом *DUMP* која служи да би се приказале добијени резултати.

## *Hive*

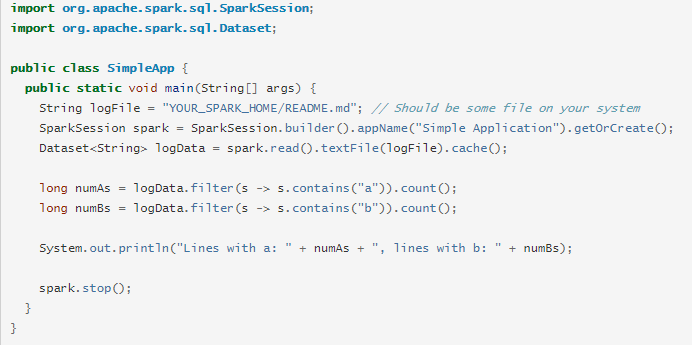
Алат *Hive* је изумео Фејсбук, да би омогућио једноставан приступ складиштеним подацима. *Hive* одликује сопствени *SQL* језик за упите који се назива *HQL* (*Hive Query Language*). Подржава све стандардне типове података као и *SQL* и могу се користити већ дефинисане, односно складиштене процедуре за обраду одређених података. Због тога што се и овај алат ослања на основне компоненте самог *Hadoop* система, код написан помоћу *HQL* језика се у позадини такође преводи у *MapReduce* операције које су неопходне да се изврше. *Hive* се може конфигурисати тако да у оквиру локалног *HDFS* система датотека складишти све податке везане за организацију својих табела.



Слика 12. Пример *HQL* наредби

## *Spark*

*Spark* је један од најпопуларнијих алата унутар *Hadoop* екосистема. Представља *framework* за *real-time* (већ док подаци путују) анализу велике количине података у дистрибуираном окружењу. *Spark* је написан у програмском језику *Scala* на универзитету „Berkeley”. Овај алат извршава израчунавања и процесирање података у *RAM* меморији кластера, како би повећао брзину извршавања у односу на стандардне *MapReduce* послове. Управо због побољшања перформанси у односу на стандардну обраду података у основној верзији *Hadoop* система постао је један од најкоришћенијих алата у оквиру овог екосистема. *Spark* нуди такође богату подршку за многе језике у оквиру разних библиотека, а поред програмских језика *Java* и *Python*, могуће је имплементирати *Spark* решења и у оквиру програмских језика *R* и *Scala*. Свакако, иако *Spark* омогућава бржу обраду података који стижу, *Spark* и сам *Hadoop* систем се не такмиче међусобно, већ се допуњују. *Hadoop* омогућава добру организацију и складиштење великих количина података на кластерима, а *Spark* омогућава брзу обраду тих података у меморији. Када се споје могућности *Spark* алата за високу брзину обраде велике количине података и напредна аналитика са могућностима складиштења велике количине података на *Hadoop* систему, добијају се најбољи резултати.

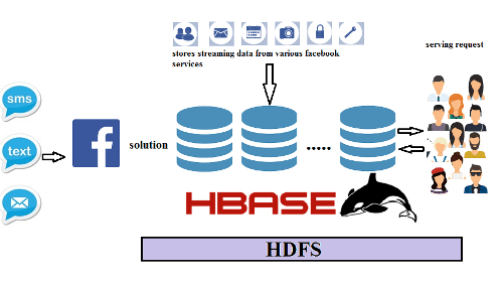


Слика 13. Имплементација *Spark –* а у програмском језику *Java*

На примеру на слици 13. приказано је укључивање библиотека неопходних за рад са *Spark* методама унутар програмског језика *Java*. Пример илуструје пример програма који има задатак да преброји број понављања слова „*a*“ и „*b*“ унутар одређене датотеке. У овом примеру је коришћена датотека која се налази на обичном систему датотека, али ову вредност је могуће заменити тако да се чита садржај и датотеке која је складиштена у оквиру *HDFS*. Неопходно је, слично као и код обичнe *MapReduce* операције извршити подешавање сесије, односно конфигурације посла који треба да се обави, тако што је неопходно навести име. Даље је могуће преко веб интерфејса који су већ поменути пратити извршење ове *Spark* апликације.

## *HBase*

*HBase* представља базу података отвореног кода, која није релационог типа, односно представља *NoSQL* базу. Подржава разне типове података, и управо због овога је у могућности складиштења било чега унутар *Hadoop* екосистема. Ова база података је пројектована по узору на *BigTable* компаније „Google“. *HBase* је осмишљен да се имплементира унутар *Hadoop* система тако да пружа функционалности сличне као и *BigTable*. Омогућава поуздан начин складиштења велике количине података, што је у главном заједничка особина коју би требало да поседују све базе података, нарочито када су у питању огромне количине података. *HBase* је написан у програмском језику *Java*.

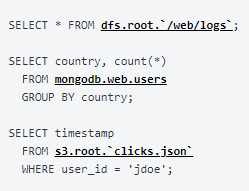


Слика 13. Складиштење података у *HBase* [9]

Управо због тога што су на чистом *HDFS* систему перформансе насумичног читањаи уписа података лоше, осмишљена је ова база која омогућава брзо складиштење и претраживање података у великим табелама. *HBase* се користи код случајева када је потребно брзо извршавање упита у бази, да би се повратила мања количина података која нам је потребна. Унутар саме базе, подаци се чувају у формату кључ – вредност који је типичан код многих *Big data* решења. Такође и за *HBase* постоје библиотеке за програмски језик *Java*, одакле се може програмирати упис, читање, модификација података итд.

## *Drill*

*Drill* подржава рад са више *NoSQL* база података и система датотека, укључујући *HBase*, *MongoDB*, *MapR-DB*, *HDFS*… Један упит написан помоћу овог алата може да придружи податке из више база података. На пример, могуће је у упиту придружити одређене податке везане за кориснике, који се налазе у *MongoDB* бази заједно са датотеком *eventLogs* из *Hadoop* – а. *Drill* поседује свој оптимизер који аутоматски реструктуира упит како би га прилагодио свакој бази за обраду. Такође, *Drill* подржава обраду података у локалу, односно могуће је инсталирати *Drill* и саму базу на исте чворове. Овај алат подржава *JSON* модел података, односно кључ – вредност начин записа, који омогућава упите над комплексним – угњежденим подацима, и другим структурама овог типа које су све чешће код модерних апликација, као и нерелационих база података. Овај алат се такође лако скалира, могуће га је инсталирати на једном чвору и тестирати његове функционалности, а затим га раширити на осталим чворовима у *Hadoop* систему. [10]



Слика 14. Упит помоћу алата *Drill* [10]

## *Zookeeper*

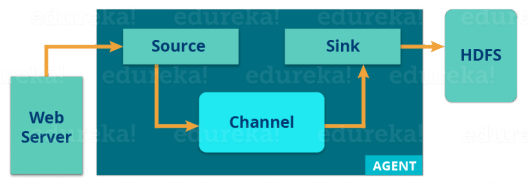
Приликом развоја дистрибуираних апликација, управљање и координација појединих делова система су сигурно међу најтежим задацима у овом процесу, и то првенствено због фактора на које крајњи корисник нема утицаја. Најчешћи проблеми су непоузданост и непредвиђени откази појединих делова инфраструкутре, који по правилу доводе до кашњења или потпуног прекида рада информационог система. *Zookeeper* има задатак да минимизује утицај ових проблема на дистрибуиране системе засноване на *Hadoop* технологији. У оваквим системима, често постоји потреба за комуникацијом између појединих делова компоненти система, иако ти делови нису у директној вези или чак и не знају за постојање других компоненти. *Zookeeper* омогућава решавање ових проблема кроз одговарајуће механизме координације и откривања сервиса, у циљу лакшег праћења целог дистрибуираног система. Најкраће речено, *Zookeeper* представља систем за поуздану конфигурацију и синхронизацију дистрибуираних система. Једна од његових најважнијих особина јесте висока поузданост. *Zookeeper* се аутоматски реплицира на више локација унутар *Hadoop* кластера, што омогућава несметано функционисање система, и у случају отказа више чворова. [11]

## *Oozie*

*Oozie* практично представља компоненту екосистема која омогућава заказивање послова који требају да се обаве. *Oozie* се успешно интегрише са остатком екосистема. Подржава неколико типова послова (као што су *MapReduce, Pig, Hive…*), као и специфичне послове попут скрипти или *Java* програма које је потребно покренути. *Oozie* користе *Hadoop* администратори да би извршили комплексне анализе на *HDFS* систему. Постоје окружења са пријатним интерфејсом за распоређивање послова које је неопходно обавити, као и класично писање наредби у терминалу. [12]

## *Flume*

Прихватање података је важан део *Hadoop* екосситема. *Flume* представља сервис који обезбеђује прихватање неструктуираних и полуструктуираних података у *HDFS*. Представља поуздано и дистрибуирано решење које помаже при сакупљању, агрегацији и премештању великих количина података. Помаже при сакупљању онлајн токова података са различитих извора, као што је мрежни саобраћај, друштвене мреже, мејл поруке, *log* датотеке у *HDFS* систему итд.

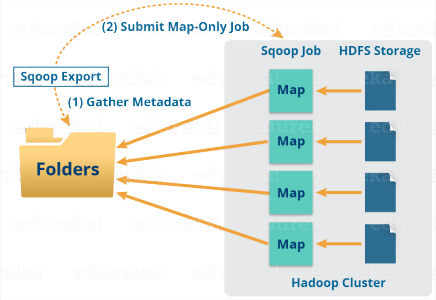


Слика 15. Имплементација *Flume* алата [8]

Постоји компонента унутар самог алата која се назива *Flume* агент, који обезбеђује прихватање великих токова података из различитих извора у *HDFS*. У примеру на слици 15, веб сервер представља извориште података. Друштвена мрежа Твитер представља један од најпопуларнијих извора великих количина података. *Flume* агент има три компоненте: *source, sink* и *channel*. *Source* прихвата податке са улазних токова и чува податке привремено у каналу (*channel*). Канал представља привремену локацију за складиштење података између изворишта података и података складиштених на *HDFS*. *Sink* сакупља податке са канала и уписује податке трајно у *HDFS*. [8]

## *Sqoop*

*Sqoop* представља још једну компоненту овог екосистема опет везану за прихватање података. Главна разлика у односу на алат *Flume* је у томе што *Flume* прихвата смо неструктуиране и полуструктуиране податке у *HDFS*, док *Sqoop* има могућности да ради и са структуираним подацима из стандардних релационих база података и да их уписује у *HDFS*. Када се извршава *Sqoop* операција, главни задатак се дели на подзадатке, односно *Map* операције. Ове операције појединачно увозе делове података у *Hadoop* екосистем. На овај начин, целокупан посао прихватања великих токова података се дели у мање задатке и постижу се боље перформансе. *Sqoop* подржава и извоз података са *Hadoop* система на сличан начин. Што се тиче извоза података, опет се посао дели на мање делове и на крају се заједно комбинују како би се добили комплетни подаци који су затражени. Најчешће се помоћу алата *Sqoop* ови подаци затим складиште у релационе базе података (*MySQL, Oracle, SQL Server…*). [8]



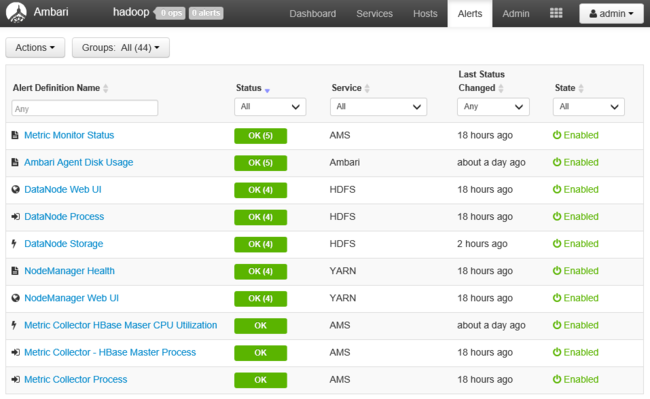
Слика 16. Имплементација *Sqoop* читања података [8]

## *Solr & Lucene*

*Apache Solr* и *Apache Lucene* представљају два сервиса који се користе за процес индексирања и претрагу унутар *Hadoop* екосистема. Ова два сервиса се практично ослањају један на други. *Lucene* представља *Java* библиотеку која пружа могућности индексирања и претраге, као и проверавања исправности речи, токенизацију стрингова итд. Развијен је и модул за програмере који преферирају програмски језик *Python* – *PyLucene*. *Solr* представља сервер високих перформанси за претраживање, и развијен је уз помоћ самог *Lucene* *framework* - а. Одликује га висока скалабилност и отпорност на отказе, дистрибуирано индексирање, претраживање и аналитика. Омогућава лако укључивање помоћу библиотека доступних програмерима у програмском језику *Java* као и другим језицима. [8]

## *Ambari*

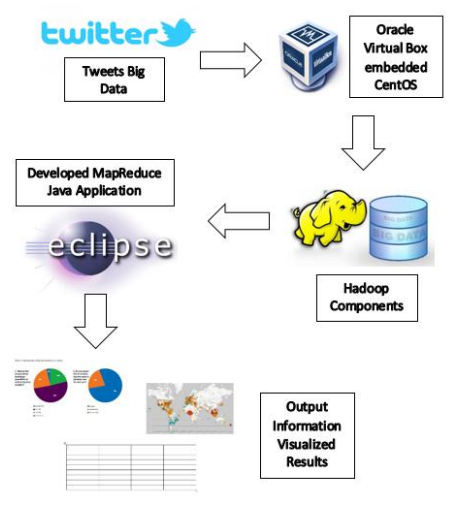
*Ambari* представља још један део *Hadoop* екосистема, односно пројекат фондације *Apache*, који има за циљ да *Hadoop* екосистем учини лакшим за управљање и организацију. Овај *framework* садржи софтверске пакете за управљање и надгледање кластера на *Hadoop* систему. Овај алат омогућава прегледан веб интерфејс коме се може приступити из интернет претраживача, како би се његове функционалности лакше пратиле и организовале. Овај алат такође нуди свој систем за давање узбуна уколико нешто крене по злу у целокупном екосистему (отказ неког чвора, мало слободног простора за складиштење података...). [8]



Слика 17. Веб интерфејс за *Ambari framework* [13]

# Преглед постојећих решења

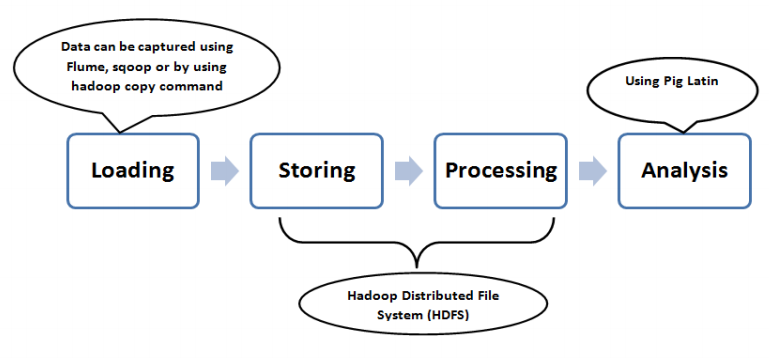
У овом раду ће у наредном поглављу бити приказан развој једне варијанте решења за анализу података са друштвене мреже Твитер у *Hadoop* систему. Друштвене мреже генерално представљају једне од главних носиоца великих количина података који су често неструктуирани. Већ постоје многи радови, који се тичу управо анализе ових података са друштвених мрежа, њиховог складиштења и обраде. Аленези и Месбах [14] се у свом раду такође баве развојем једне апликације намењене за анализу података са друштвене мреже Твитер. У њиховом раду је описан развој апликације засноване на *Java – Hadoop* технологији. Задатак те апликације је да се за реч рак (*cancer*) на друштвеној мрежи Твитер прикажу резултати у виду одговора у ком граду је највише поменут тај појам. Анализира се датотека која садржи велику количину података. Да би осигурали да је стратегија анализе успешна, аутори су осмислили алгоритам како би се сви синоними речи канцер третирали при анализи. Апликација је реализована помоћу развојног окружења *Eclipse*. Аутори овог рада су се ослонили на скуп велике количине података (*dataset*) преузет од *ICWSM* организације, који садржи више од 14 милиона твитова корисника. Садржај се анализира стандардним *MapReduce* процедурама да би се добили жељени резултати. Искоришћен је *Excel* како би се на крају добили графички прикази добијених резултата. Такође, за потребе израде овог рада, аутори су користили виртуелну машину са подигнутим оперативним системом *CentOS 6.4*. са подигнутим *HDFS* системом.



Слика 18. Идејно решење апликације за анализу података са друштвене мреже Твитер [14]

Аутори Лин и Риабој [15] у свом раду објављују утиске рада управо на друштвеној мрежи Твитер. У раду је приказана употребна вредност *Hadoop* система за организацију података унутар саме друштвене мреже, и дискутовано је о могућностима проширења, адаптацији података итд. Обрађене су теме везане за „копање података“ (*data mining*) на самој друштвеној мрежи.Коришћенe су *Pig* скрипте како би се помоћу њих реализовало машинско учење на њиховој платформи. Ове скрипте су направљене тако да обраде податке потребне за учење као листе уређених парова који се онда складиште на трајну локацију (*HDFS*). [15]

Џен и Вишал [16] користе *Pig* алат унутар *Hadoop* екосистема како би извршили анализу података о кривичним делима, који су присутни у великој количини. Анализирали су велике количине података прикупљених из различитих извора. Алат *Sqoop* је искоришћен како би се ови обимни подаци структуиране форме повезали са стандардном релационом базом. Искоришћени су *JDBC* конектори како би се апликација повезала са базом података. За анализу ових података се користе скрипте написане у језику *PigLatin*. Целокупно складиштење података је реализовано на *HDFS* систему, а у раду је и укратко описан принцип функционисања *MapReduce* технике, као и самог *Hadoop* *framework* – а. Циљ апликације јесте да се извуку важне информације из великог скупа података о кривичним делима. Дати су резултати укупног броја криминалних радњи за сваку државу, број најчешћих видова криминалних радњи, и број злочина почињених над женама.



Слика 19. *Framework* за анализу података о криминалним делима [16]

У свом раду Шила [17] анализира осећања корисника (*sentiment analysis*) на друштвеној мрежи Твитер уз помоћ *Hadoop* система. Користећи јавне *API* интерфејсе које ова друштвена мрежа нуди, могуће је приступити делу објављених твитова од стране корисника и анализирати њихов садржај. У раду су укратко описане технике визуализације резултата, мапирања твитова који се прихватају за анализирање, и начин мерења резултата унутар *HDFS* система. У самом раду је урађено и поређење са резултатима других радова који су се бавили такође анализом сентимената на овој друштвеној мрежи и њихови резултати до којих су дошли.

# Анализа Твитер објава у *Hadoop* систему

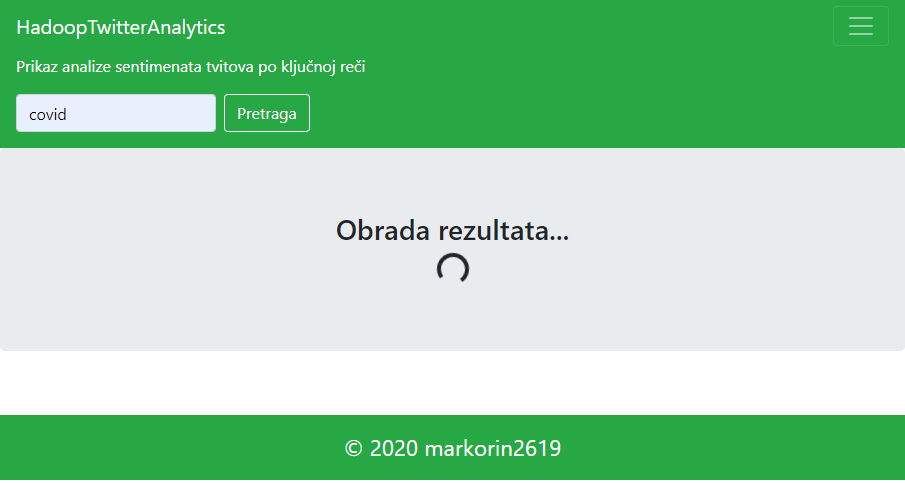
Решење које ће бити приказано у овом раду представља један вид анализе објава корисника на друштвеној мрежи Твитер. Ова друштвена мрежа је одабрана због тога што у време писања овог рада има највише *API (Application Programmable Interface)* услуга које пружа програмерима да приступе подацима са ове мреже. Пре почетка рада да би се могло приступати овим *API* сервисима, неопходно је направити *twitter developer* налог да би се добили акредитиви, односно неопходни подаци који се аутоматски генеришу, да би се знало који програмер је позвао *API*. Ипак, постоје извесна ограничења у погледу количине података коју Твитер може да достави програмерима. Тренутна ограничења су у томе што је могуће добити највише 100 резултата у виду објава корисника (твитова) добити по једном позиву *API*. Такође, није могуће направити више од 180 захтева истовремено. [18] Ово су тренутно бесплатне опције, док постоје и *premium* налози који се плаћају, а који нуде знатно више података који могу да се добију приликом једног позива Твитеровог сервиса.

Због тога што Твитер представља отворену мрежу, у коју је укључено и пуно битних личности из свакодневног живота, могу се извући многи подаци о томе какво мишљење има нека особа на одређену тему, како се осећа итд. Назив за анализу ових осећања корисника које они исказују кроз своје објаве – твитове, назива се анализа сентимената (*sentiment analysis*) и ова анализа је веома чест случај код анализе друштвених мрежа и проучавања великих количина података који долазе са разних друштвених мрежа. Анализа сентимената се најчешће спроводи помоћу *NLP (Natural Language Processing)* библиотека. Задатак ових библиотека јесте да се из одређеног дела текста извуче закључак о томе какав утисак оставља тај део текста. Утисак може бити позитиван, негативан и неутралан. Ово представља добру основу за имплементацију у анализи велике количине текстуалних података које генеришу корисници. Могуће је извући закључке о томе како који корисник реагује на задату тему, шта му се допада а шта не итд.

## Опис компоненти апликације

За апликацију реализовану у овом раду првобитан покушај је био да се имплементира *Stanford NLP* библиотека, која је доступна за интеграцију и у програмском језику *Java*. Међутим, тестирањем *MapReduce* посла који апликација треба да обави помоћу ове библиотеке, дошло се до закључка да је потребно превише ресурса да би могао да се покрене *MapReduce* посао у тренутној конфигурацији на којој је тестирана ова апликација. Ова библиотека сама садржи пуно предефинисаних фраза на основу којих управо и врши анализу сентимената одређеног текста. Због ове чињенице, извршна *jar* датотека постаје преобимна, и приликом извршавања посла дође до преоптерећења ресурса. Због овога, одабран је веб сервис *Semantria* који омогућава давање резултата анализе сентимената на основу прослеђеног текста који се шаље овом сервису.Тренутна конфигурација на којој је реализовано ово решење је постављена у режиму *Single Node Cluster*, које омогућава практично да један обичан рачунар буде главни и једини унутар целог *Hadoop* система, и да се на њему самоме постави *HDFS* за складиштење велике количине података. Ово је згодно због тога што је могуће тестирати разне *Hadoop* апликације „у локалу“, без професионалног серверског окружења, а уколико се решење покаже као успешно, могуће је лако скалирати целокупан систем и проширити га са онолико чворова колико је неопходно.

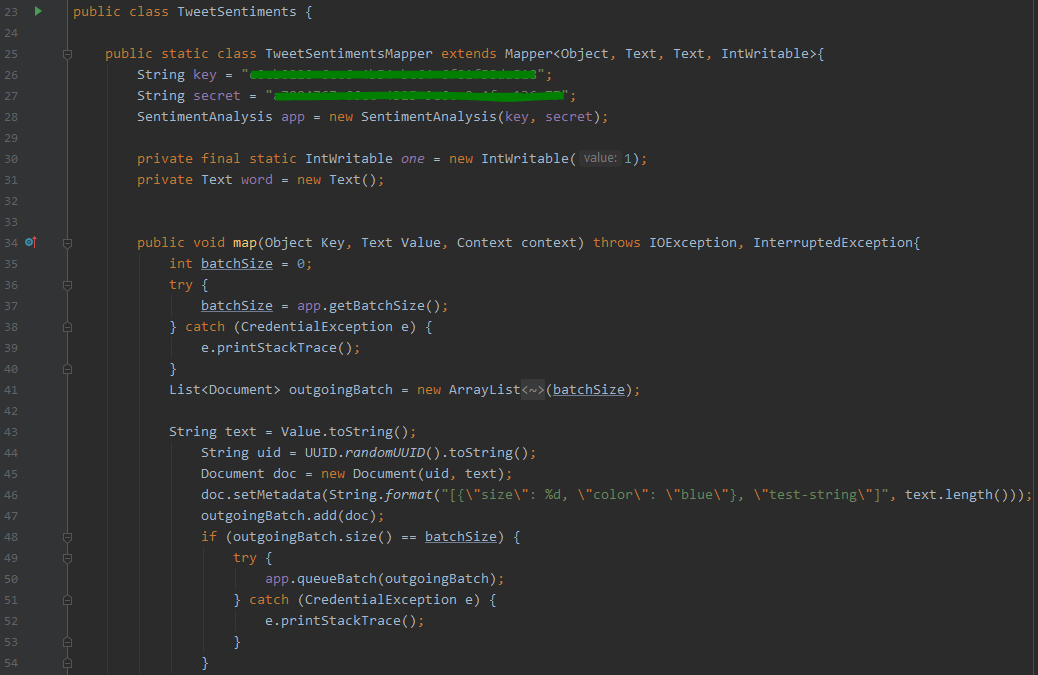
Апликација је написана у *Java* програмском језику, с обзиром да сам *Hadoop* екосистем обилује библиотекама за овај програмски језик. Прихватање података функционише кроз стандардне Твитер *API* услуге које ова друштвена мрежа нуди, и директно се складиште у оквиру локалног *HDFS* система. Сама апликација практично представља једноставну веб апликацију, реализовану помоћу *Spring* *framework* – а који има задатак да на једној страници захтева од корисника да унесе термин за који га занима анализа сентимената на друштвеној мрежи Твитер, а затим добија одговор када се заврши *MapReduce* операција каква је анализа за тај појам. За дизајн саме странице је коришћен *framework Bootstrap 4.4*.



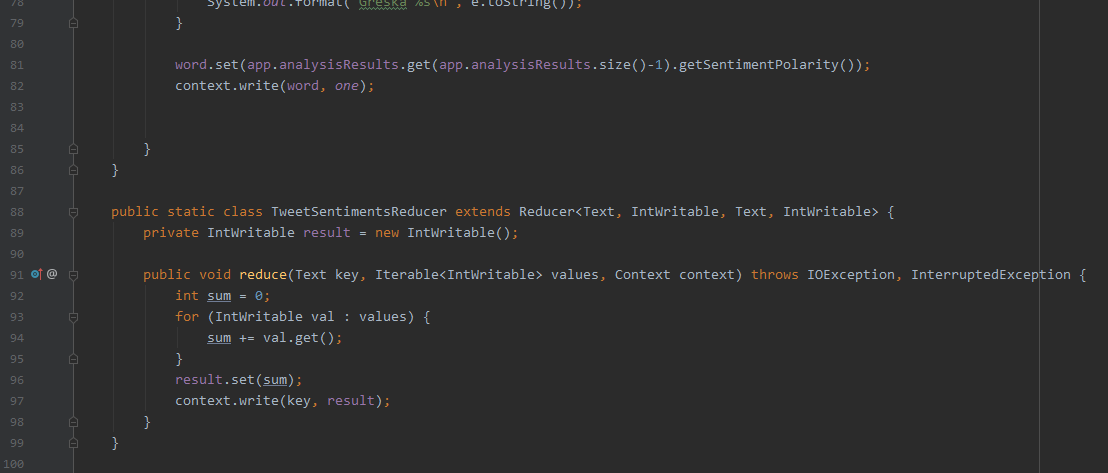
Слика 20. Изглед апликације за анализу сентимената

## Развој апликације

Пошто се ради о апликацији која ће се извршавати у оквиру *Hadoop* система, неопходно је дефинисати *MapReduce* део апликације који треба да обради анализу сентимената за сваку објаву корисника. Библиотека *Semantria* слично као и Твитер платформа користи систем аутентификације на њен систем да би се могле користити услуге анализе сентимената. Неопходно је поседовати кључ и тајну лозинку за коришћење ових услуга, које се добијају када се прави налог за коришћење ове библиотеке на њиховом званичном сајту. Сама логика реализације ове *MapReduce* операције се састоји од тога да се у фази мапирања једна објава корисника шаље на анализу библиотеци за *NLP*, да би се добио резултат у виду позитивних, негативних или неутралних сентимената корисника. Затим се овај резултат бележи тако што се као излаз даје уређени пар кључ – вредност, где кључ представља сентимент корисника (*Positive, Neutral, Negative*), док вредност увек представља број 1, како би се у фази редуковања сабрале све вредности и добио коначан однос ових сентимената, а затим даље проследио на графички приказ веб страни.

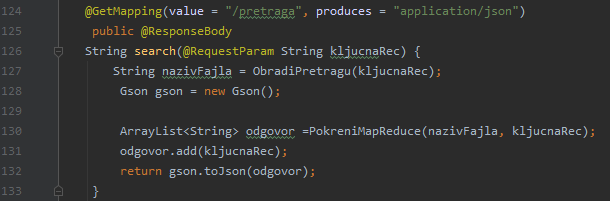


Слика 21. Приказ кода за фазе мапирања



Слика 22. Приказ кода фазе редуковања

Кључни део реализације *Map* фазе јесте у дохватању самих резултата како би се проследили на фазу редуковања. Што се тиче класе за *Reduce* фазу, реализована је тако да прихвати све излазне вредности из фазе мапирања, како би сумирала за сваки кључ број понављања, тако да ће се добити као крајњи излаз однос сентимената за одређени појам за који је претрага започета.



Слика 23. Имплементације методе *search*

У суштини, обрада задатака започиње када корисник унесе са веб стране појам за који га интересује анализа сентимената, и тај захтев се обрађује унутар методе *search*. Ова метода има задатак да позове остале методе које треба да изврше своје функционалности, у виду обраде претраге, што представља позив методе која прави датотеку са резултатима Твитер претраге на *HDFS* систему и тамо складишти резултате.

Слика 24. Код за позив Твитер *API*

Сама Твитер претрага и позив *API –* ја се врши преко *twitter4j* библиотеке. Потребно је подесити назив упита, што представља кључну реч за коју се траже подаци о објавама са друштвене мреже Твитер. Такође, пошто коришћена библиотека за *NLP* ради само са енглеским речима, претрага је прилагођена за енглеско говорно подручје. Када се добију подаци, они се у складној форми даље прослеђују методама које треба да врше њихово складиштење у оквиру *HDFS* датотека. Искоришћена је класа *Timestamp* из стандардних *Java* библиотека, како би се за сваку датотеку уписану у *HDFS* генерисао „временски отисак“, односно како би се резултати сваке извршене претраге могли сачувати у један директоријум, а свака датотека имати различито име које ће увек бити јединствено захваљујући овом временском отиску.

Слика 25. Реализација кода који се тиче уписивања резултата у *HDFS* систем

Слика 26. Приказ *HDFS* система након неколико претрага

Захваљујући библиотекама које су омогућене у *Java* програмском језику за приступ *HDFS* систему, могуће је на једноставан начин креирати нову датотеку у оквиру овог система, тако што се наводи њено име, и локација главног *Namenode* чвора, који је у овом случају *hdfs://localhost:9000/*. Крајњи упис се врши помоћу методе *write*, која из променљиве типа *String*, у коју су складиштени привремено резултати, уписује у новокреирану *HDFS* датотеку. Постоји метода која покреће већ формирану апликацију која има задатак да изврши *MapReduce* посао у виду анализе сентимената, којој се прослеђује управо формирана *HDFS* датотека, из које треба да анализира садржај. Креира се локална датотека „log.txt“, како би се преусмерио излаз системских порука које даје *MapReduce.*

Слика 27. Код који се за покретања *MapReduce* посла

Приказ резултата обраде се врши преко *ajax* позива на страну сервера, одакле се добијају управо подаци о анализи сентимената, а затим се ови подаци користе како би се графички приказали резултати претраге. За приказ резултата је коришћена библиотека *Chart.js*, која омогућава исцртавање многих стандардних графичких приказа у оквиру једне веб стране.

Решење приказано у овом раду свакако представља експерименталну основу за стварно усавршавање *Hadoop* платформе за анализу података. Могуће је уколико имамо различите скупове података такође имплементирати одређене делове апликације које ће бити намењени раду само са тим подацима, као и интегрисању са другим подацима. Због тога што се у овом решењу током *MapReduce* посла стално позива удаљени *API* сервис за анализу сентимената из објава корисника, перформансе саме претраге нису најоптималније. Оптималније решење би било када би се инсталирало у правом окружењу, са пуно рачунара односно чворова и применила *Stanford NLP* библиотека. Перформансе би биле осетно побољшане, а *MapReduce* функционалности би се заиста могле извршавати као такве, с обзиром да би подаци били дистрибуирани преко читавог кластера. Такође, коришћена библиотека *Semantria* није у потпуности оптимизована, тако да се дешава да за одређене реченице добијемо резултате који говоре да су те реченице у негативном контексту, а оне су заправо позитивне.

Слика 26. Приказ резултата анализе података

# Закључак

*Hadoop* и његови алати захваљујући својим особинама омогућавају одличан начин за складиштење и обраду великих количина података. Већ су развијена бројна решења на основу *Hadoop* система, а у овом раду је приказана једна могућа варијанта за употребу овог система. Сама анализа сентимената представља веома широку грану за истраживање. Битно је имати довољно добре библиотеке и правилно их употребити у оквиру *MapReduce* функционалности која је обезбеђена у *Hadoop* систему. Могућа надоградња решења приказаног у овом раду би се могла огледати у додавању *HBase* базе података за складиштење резултата анализе, као и у проширивању броја чворова, односно другачијег конфигурисања самог *HDFS* система. Програмски језик *Java* има огромну заједницу програмера који свакодневно развијају нове библиотеке које пружају начине за интеграцију са разним технологијама у области информационих технологија, па тако и *Hadoop* алата. Због тога што је *Hadoop* отвореног кода, постоји читава заједница његових програмера који стално раде на његовом усавршавању и надоградњи постојећих верзија његовог екосистема. Како се количина података свакодневно увећава, потреба за *Hadoop* системом и другим алатима за брзо и ефикасно обрађивање података ће сигурно све више расти у будућности.

# Литература

[1] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers „Fundamentals of Business Process Management“, 2018.

[2] <https://www.scirp.org/journal/paperinformation.aspx?paperid=88720>, приступано 20.05.2021

[3] S. Wu, „Big data processing with Hadoop“, 2015.

[4] <https://hadoop.apache.org/docs>, приступано 20.05.2020.

[5] <https://www.dezyre.com/hadoop-tutorial/hadoop-mapreduce-tutorial->, приступано 20.05.2020.

[6] V.K. Vavilapalli, A. C. Murthy... „Apache Hadoop YARN: yer another resource negotiator“, 2013.

[7] D. Šušak, Z. Đurić, „Analiza Apache Hadoop – a i mogućnosti upotrebe“, 2019.

[8] <https://www.edureka.co/blog/hadoop-ecosystem>, приступано 25.05.2020.

[9] <https://beyondcorner.com/learn-apache-hbase/facebook-messenger-case-study-with-apache-hbase/>, приступано 25.05.2020.

[10] <https://drill.apache.org/>, приступано 25.05.2020.

[11] <https://hortonworks.com/apache/zookeeper>, приступано 25.05.2020.

[12] <https://www.tutorialspoint.com/apache_oozie/apache_oozie_quick_guide>, приступано 25.05.2020.

[13] <https://docs.microsoft.com/bs-latn-ba/azure/hdinsight/hdinsight-hadoop-manage-ambari>, приступано 25.05.2020.

[14] S. Alenezi, S. Mesbah, „Big Data Spatial Analytics in Social Networks using Hadoop“, 2015

[15] J. Lin, D. Ryaboy, „Scaling Big Data Mining Infrastructure: The Twitter Experience“, 2013

[16] A. Jain, V. Bhatnagar, „Crime Data Analysis Using Pig with Hadoop“, 2015

[17] L. J. Sheela, „A Review of Sentiment Analysis in Twitter Data Using Hadoop“, 2016

[18] <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets>, приступано 25.05.2020.