Универзитет у Београду
Математички факултет

Немања Вељковић 1007/2012

Приказ развојног оквира *Hadoop Map - Reduce* на примеру анализе геномских секвенци

Мастер рад

Београд,
2017.

Универзитет у Београду

Мастер рад

Аутор:	Немања Вељковић 1007/2012
Наслов:	Приказ развојног оквира *Hadoop Map - Reduce* на примеру анализе геномских секвенци
Ментор:	др Саша Малков Математички факултет, Универзитет у Београду
Чланови комисије:	др Владимир Филиповић Математички факултет, Универзитет у Београду
	др Јована Ковачевић Математички факултет, Универзитет у Београду

Датум одбране: _____

Садржај

Глава 1

Увод

Време у којем живимо можемо назвати дигиталном ером. Свакодневно се на различите начине производе огромне количине података, које у великој мери остају необрађене. Данијел Прајс је проценио да је само до краја 2016. укупна количина података на Вебу, не укључујући Мрачни Веб[1], на Интернет постављен 1.1 зетабајт (2^{70}, 10^{21}) [1]. Предвиђа се да ће до 2020. бити произведено укупно 44 зетабајта података.

Извори тих података су различити. У највеће спадају Цern, берзе, друштвене мреже, биоинформатички институти... Са развојем интернета ствари (енг. *Internet of Things*) машине почињу да генеришу много више података од оних који су настали људском активношћу. У такве податке спадају разни логови, идентификације радио-фреквенција (RFID), сензорске мреже, GPS руте, малопродајне трансакције... Обим јавно доступних података повећава се сваке године. Због тога организације и компаније више не морају да обрађују само сопствене податке. У будућности ће њихов успех у великој мери зависити од способности да извуку вредности из података које су произвеле друге организације или компаније.

Тако велику количину података није могуће једноставно обрадити на једној машини, што нас наводи на паралелну употребу више радних станица. Први проблем који се може јавити јесте отказивање хардверских делова. Један од начина да се обезбеди заштита од губитка података услед квара јесте репликација података на више дискова. Редундантне копије података чувају се на више места у

[1] Мрачни Веб (енг. *Deep Web*) - део Веба чији садржај није индексиран стандардним претраживачима из било ког разлога.

систему, тако да је у случају квара на располагању резервна копија. Други проблем представљају брзина читања и квалитетна обрада тих разасутих података [2].

Такви подаци, који се мере у терабајтима, петабајтима и већим мерним јединицама, популарно се називају *BigData*. За њихову обраду потребно је комбиновање различитих, добро познатих технологија, као и нових, мање познатих, ради добијања релевантних података у реалном времену. Најпопуларнија технологија за обраду *BigData* је *Apache Hadoop*. *Hadoop* је прилично велика колекција софтвера, која укључује још и дистрибуирани систем датотека. Представља развојни оквир који омогућава дистрибуирано процесирање великих скупова података на кластерима рачунара уз коришћење једноставног "програмског модела" [3].

Према једној од дефиниција, *BigData* се заснива на особинама које су испрва одређене као "три В", а касније су додате још неке [4]:

- Волумен (енг. *Volume*): Велике количине података, обично од терабајта до петабајта и шире
- Брзина обраде (енг. *Velocity*): Подаци који се стварају брзим темпом требало би да се анализирају у скоро реалном времену
- Разноврсност (енг. *Variety*): Подаци могу бити структуирани у разним форматима или у базама података, али могу бити и потпуно неструктурирани
- Поузданост (енг. *Veracity*): Подаци могу бити недовољно тачни, тј. непоуздани, па је потребно на време препознати такве податке и прочистити или их отклонити
- Вредност (енг. *Value*): Подаци који се обрађују доносе организацији неку повратну информацију значајну за даљи рад, иначе су бескорисни
- Повезаност (енг. *Valence*): Могућност повезивања података у графове, попут атома. Како мрежа расте временом, тако се повећава и вредност тих података

Hadoop је поуздана, скалабилна платформа за складиштење и анализу. С обзиром на то да ради на различитим врстама хардвера и отвореног је кода, веома је приступачна и када је реч о цени. *Map-Reduce* представља програмски модел који

се користи за извршавање дистрибуираних операција над подацима смештеним на различитим системима датотека. Пружа могућност да се из огромне количине података, који су непрегледни за обраду и претраживање због своје величине, на основу парова кључ/вредност издвоје значајне информације. У основи оквира *Hadoop Map-Reduce* налази се систем за пакетну обраду (енг. *batch processing*), који није погодан за интерактивну употребу. Резултати упита не могу се очекивати брзо. Упити обично трају неколико минута, тако да се резултат не може добити у реалном времену. За обраду у реалном времену постоје други алати, који ће бити описани у даљем тексту.

Истовремено са напретком техника за истраживање података на *Map-Reduce* системима, у биологији се усавршавају технике секвенцирања. Крајем осамдесетих година прошлог века започет је дуг и мукотрпан рад на пројекту секвенцирања људског генома (енг. *Human Genome Project*) са циљем откривања и мапирања свих гена (функционалних сегмената ДНК) у људском организму. У томе се напокон и успело 2003. године. У оквиру пројекта је откривено да људски геном има преко три милијарде базних парова. Базни парови се састоје од гена, за које се процењује да их има између 19.000 и 20.000. Рад на покушају разумевања начина испољавања и интеракције ових гена траје и данас, иако је већ доста тога познато. На основу анализе људског ДНК можемо сазнати много тога: установити предиспозиције за наследна обољења и потенцијалне здравствене ризике, као и генеалошке податке о коренима фамилије.

Циљ овог мастер рада је сагледавање реалних добити од примене развојног оквира *Hadoop Map-Reduce*, уместо уобичајеног централизованог приступа развоју софтвера. Улазни подаци развијане тест-апликације су геномске секвенце, а задатак је пописати и пребројати све подсеквенце дате дужине. Процес је дистрибуиран на више чворова и дели се на две фазе: прва је распоређивање (у коју је укључено и филтрирање и сортирање) свих могућих подсеквенци, а друга обрада и пребројавање тако срећених подсеквенци.

У поглављу 2 описан је екосистем *Hadoop-a*, његове основне компоненте, често коришћени алати, предности и недостаци. Затим су у поглављу 3 описани неки од основних појмова генетике, попут ДНК, секвенцирања ДНК и формата за записивање геномских секвенци *FASTQ*. Поглавље 4 садржи опис технологија и алата

који су коришћени у раду, а који не припадају развојном оквиру *Hadoop*. У поглављу 5 биће описан *Hadoop Map-Reduce* пројекат написан у Јави, у којем се приказује како подесити задатак за *Map-Reduce*.

Глава 2

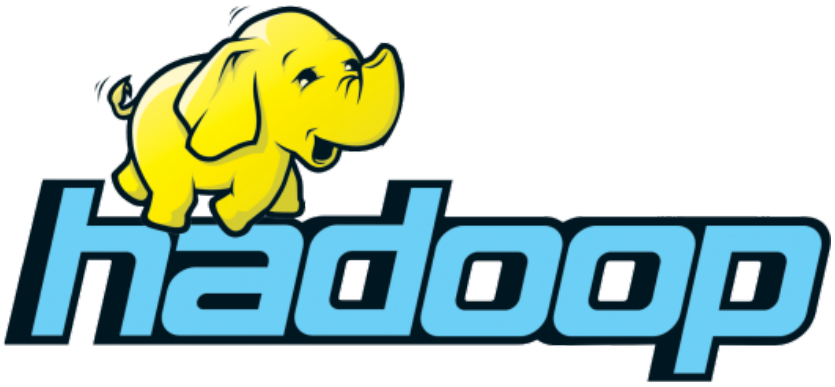
Развојни оквир *Hadoop*

Идеја *Hadoop-a* је да уместо што податке прилагођавамо израчунавању – израчунавање прилагодимо подацима. С тим у вези није потребно првобитно филтрирање података, већ се израчунавање врши на читавом скупу података. Сам екосистем је подељен на неколико независних делова ради бржег опоравка након хардверских грешака. Садржи четири основне компоненте на које се надовезују додатни алати [2]:

- *Hadoop Common* - садржи библиотеке и услуге које су потребне другим модулима
- *HDFS (Hadoop Distributed File System)* - дистрибуирани систем датотека који складишти податке на стандардним машинама, пружајући врло високу укупну "пропусност" (енг. *bandwith*) преко целог кластера
- *Map-Reduce* - скалабилан програмски модел који дели обраду података на много различитих процеса
- YARN (*Yet Another Resource Negotiator*) - платформа за управљање ресурсима, одговорна за управљање рачунарским ресурсима у кластерима, која врши распоређивање корисника и апликација.

На ове основне компоненте надовезују се остале апликације, чија је најважнија особина поузданост података. Оне се могу користити и независно, у односу на потребе и знања корисника. Такве апликације су *Apache PIG, Apache Hive, HBase*...

Њима се приступа путем *Map-Reduce* задатака, који се могу писати у програмским језицима Јава или Пајтон. Затим се за сваку од ових апликација користи њен специфичан програмски језик, попут *SQL-a* у *Hive* или *Pig Latin*, чиме се крајњим корисницима, најчешће аналитичарима података, омогућава прегледање у њима погодном формату.



Слика 2.1: *Hadoop* лого

Пројекат *Hadoop* започео је Даг Катинг (*Doug Cutting*), који је творац и познате библиотеке за претраживање текста *Apache Lucene*. Интересантно је да је *Hadoop* било име играчке слона његовог сина, те одатле потиче идеја за назив и лого апликације (слика 2.1). *Apache Hadoop* вуче корене из машине за претрагу Веба, која је део пројекта *Apache Nutch*.

Гуглови инжењери су 2003. објавили чланак о систему датотека који су они користили и који је постао узор за *HDFS* [5]. Наредне године објављују још један чланак, који представља први запис о *Map-Reduce* концепту [6]. Већину главних алгоритама инжењери ускоро успевају да прилагоде за *Nutch*, користећи *Map-Reduce* и *HDFS*. Након успешног имплементирања у продукционој верзији, формира се независни пројекат под именом *Hadoop*.

2.1 HDFS - *Hadoop* дистрибуирани систем датотека

HDFS је дистрибуирани, скалабилни и преносни систем датотека писан у Јави у циљу подршке за развојни оквир *Hadoop*. Свака инстанца чвора има један именски чвор (енг. *NameNode*) и кластер чворова података (енг. *DataNode*) који формирају *HDFS* кластер [2, 3]. *HDFS* може да складишти велике количине датотека,

обично у распону од гигабајта до петабајта, на више машина. Основни концепт дизајна је да користећи пуно чворова вршимо процесирање у ранијим фазама тако што датотеке делимо у ситније блокове који се распоређују по чворовима, којих може бити и на стотине хиљада. Ти чворови обично нису скупи, тако да настаје скалабилан и не тако скуп систем. Улазна датотека може бити већа од највећег диска у кластеру, али то не представља сметњу, јер се дели на блокове.

Овај систем датотека је веома отпоран на грешке и није хардверски захтеван. Како је намењен за сервере, тачније за стотине или чак хиљаде сервера који имају различите компоненте и за које постоји одређена вероватноћа отказивања, главни циљ архитектуре *HDFS* је да брзо открије грешке и да их аутоматски отклони. Има једноставан модел за приступ датотекама – упиши једном, читај више пута. *HDFS* није намењен за интеракцију са корисником, већ је више окренут несметаној обради података од стране других алата. *HDFS* је дизајниран тако да има архитектуру главни/подређени (енг. *master/slave*). Овде би то значило да један кластер *Hadoop* има један и само један именски чвор и више чворова података. Обично је један сервер главни и на њему се инсталира именски чвор, а на осталима се инсталирају чворови података. Главни сервер на којем је инсталиран именски чвор контролише приступ датотекама и управља именским простором (енг. *The File System Namespace*), који подржава традиционалну хијерархију. Корисник управља датотекама на главном серверу, види директоријуме и датотеке. Даље се датотеке деле у блокове и складиште на остале сервере на којима је инсталиран чвор података који служи за складиштење блокова и дозвољава креирање блокова, брисање и понављање.

Сумирано, именски чвор чува метаподатке блокова и крајњим корисницима приказује саму датотеку, а не блокове који корисницима нису читљиви, док чвор података чува податке. Ово би се могло посматрати као да се у именском чвору чувају адресе блокова, име и број копија. *HDFS* има ту могућност да се блокови уписују више пута. Ово смањује ризик од губљења података. Подразумевани број копија је три, што значи да ће се сваки блок копирати три пута. Копије не морају бити на једном серверу, већ могу да се сместе на све сервере у кластеру на којима је инсталиран чвор података. Тако се смањује ризик од губљења података. Комуникација између сервера у кластеру одвија се помоћу протокола *TCP/IP*, па самим тим и именски чвор комуницира са чворовима података. Поред тога, чворови података периодично шаљу сигнал именском чвору, што означава да је чвор жив. Када чворови података не шаљу сигнал то је знак да је дошло до отказа и потребно

је заменити чвор. Тај сигнал се назива откуцај срца (енг. *Heartbeat*) и показује да све функционише у најбољем реду. Постоји могућност копирања главног сервера, односно постоји и секундарни именски чвор, који је активан истовремено са примарним, па се у одређеним интервалима формира слика распореда података на примарном и копира на секундарни именски чвор.

Подразумевана величина блока података у чвору података износи 64MB, а често се повећава и на 128MB. Перформансе зависе од тога колико је велика датотека која се обрађује, ако је довољно велика, онда се користи оптималан број блокова, јер је *Hadoop* предвиђен за такве случајеве. У случају да је датотека мања, и величина блокова је мања и то може довести до проблема (мапирање се врши по блоку, а И/О операције су споре). Да би се то превазишло, мање датотеке пре постављања у *HDFS* треба спојити или надовезати на више већих.

Постоји више приступа *HDFS* систему датотека: *Web Rest Api, Java Api* преко класе *FileSystem, C/libhdfs* и путем *HDFS* конзоле, која користи команде сличне стандардним *UNIX* командама.

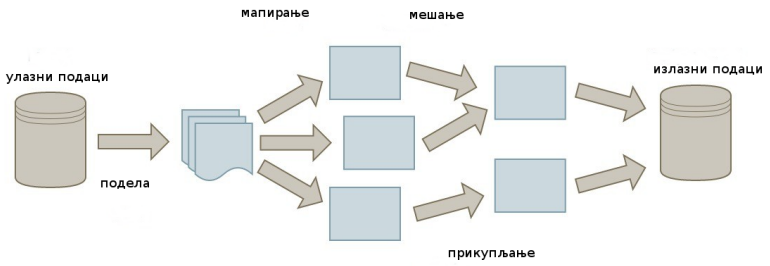
2.2 Програмски модел распоређивање-прикупљање

Распоређивање-прикупљање (енг. *Map-Reduce*) представља програмски модел за процесирање великих количина података који почива на дистрибуирању делова израчунавања на различите чворове и затим на њиховом паралелном извршавању. Први задатак је да се изврши дељење података у парчиће (енг. *chunks*) који се затим распоређују, сортирају и на крају обједињују.

Модел се може посматрати и као две одвојене целине, део за распоређивање и део за прикупљање [7]. Распоређивање се састоји од: читања записа, распоређивања, комбиновања и партиционисања блокова. Прикупљање се такође може раздвојити на послове мешања, сортирања, сумирања и форматирања излаза. На слици 2.2 је приказано како изгледа ток података.

Поступак читања записа своди се на превођење улазних података у одговарајуће записе погодне за распоређивање. Подаци се прослеђују у форми кључ/вредност.

У фази распоређивања на сваком пару кључ/вредност извршава се функција *Map()*, која производи парове посреднике. Важно је прецизно одредити парове, јер од



Слика 2.2: Процес распоређивања и прикупљања

тога зависи ефикасност наредних корака. Кључ зависи од тога како ће се даље груписати парови, док је вредност информација значајна за анализу у фази прикупљања.

Комбинатор представља локални прикупљач који се може користити у фази распоређивања за конкретни чвор. Његова сврха је да смањи број парова посредника који се након распоређивања прослеђују осталим чворовима. Често могу доста унапредити перформансе, с тим што није сигурно да ће се извршити локално распоређивање.

Код партиционисања настају парчићи од парова посредника, обично парче по прикупљачу. За свако парче се одреди хеш код, најчешће по алгоритму *md5sum*. Затим се за свако од њих обавља операција модул по броју прикупљача, што узрокује да се кључеви дистрибуирају равномерно на прикупљаче по методи случајног узорка, али се ипак осигурава да ће кључеви са истом вредношћу са различитих распоређивача бити распоређени на истом прикупљачу.

Фаза прикупљања почиње корацима мешања и сортирања. Прво се врши преношење излазних датотека на локалну машину, затим се ти појединачни делови сортирају у већу листу. Сврха сортирања је груписање идентичних кључева како би се могло лакше итерирати кроз вредности у каснијим фазама. Све радње се врше аутоматски, тако да није могућа значајнија измена плана мешања и сортирања. Евентуално се може проследити компаратор на основу којег ће се вршити сортирање.

У фази обједињавања прикупљач извршава функцију прикупљања кроз дате групе. Подаци се тада сакупљају, филтрирају и комбинују. Резултат рада је нула или

више парова кључ/вредност који се прослеђују у завршну обраду. На крају се формирани парови записују назад на *HDFS*, при чему сам формат записа може бити различит.

Препорука је да се користи један распоређивач по диску, док један прикупљач долази на једно процесорско језгро. Оптималан број зависи и од величине блока у самом задатку. Боље је имати више излазних датотека, али треба водити рачуна и да се послови не извршавају предуго. Један прикупљач по језгру осигураће паралелно извршавање. Такође не треба компликовати у вези вредности кључа, па се препоручује да то буде логична вредност која ће олакшати даље мешање, груписање и сортирање које врши сам *Hadoop*.

Кључ може бити и композитни, ако је такав задатак. Некада може бити корисно направити секвенцу наизменичних послова распоређивања и прикупљања, како би се убрзало и олакшало извршавање. Резултат прикупљања се даље прослеђује распоређивачу, који сада ради другачије распоређивање, и тако више пута. Вредност може садржати и неке метаподатке који нису садржани у сировим подацима, нпр. редни број.

Као и у релационим базама података, врло битна операција јесте спајање табела (енг. *JOIN*) [8]. У *Hadoop-у* се спајање врши на основу кључа, тако да је битно добро одредити одговарајући кључ у делу распоређивања, а ако има неких сувишних података, њих треба пребацити у вредност. Програмски модел је погодан за операције множења матрица, које се доста користе у машинском учењу, истраживању података и статистичким анализама. Најбоље ради када су задаци распоређивања и прикупљања јасно раздвојени и независни. Распоређивачи виде само поједине делове података. То обично може бити врло корисно ограничење, али има и случајева када је то недостатак.

2.3 YARN

YARN је систем за управљање извршавањем дистрибуираних апликација. Главна функција YARN-а је управљање ресурсима унутар кластера. Састоји се од две компоненте: планера (енг. *Scheduler*) и управљача апликацијама (енг. *Applications Manager*). Ове компоненте су делови управљача ресурсима (енг. *Resource Manager*).

Планер је компонента која брине о алокацији ресурса апликација које се извршавају. Битно је напоменути да је потребно водити рачуна само о ресурсима, односно не брине се о томе какав је статус апликације која се извршава; не прати се рад апликације. Како се брине само о алокацији ресурса, не мора се водити рачуна о томе да ли је дошло до грешке или је код лош, што значи да ће ресурси бити додељени некој апликацији док год њен рад не прекине корисник или нека друга компонента. Управљач апликацијама рукује апликацијама писаним за *Hadoop*. Његова сврха је да прихвати задатак, испита ресурсе и донесе закључак шта треба прво да се изврши. Управљач апликацијама је такође задужен и да рестартује задатак, односно апликацију, ако дође до неке грешке [9].

Главна компонента YARN-а је управљач ресурсима, која управља свим доступним ресурсима кластера и на тај начин помаже у управљању дистрибуираних апликација које раде на систему YARN. Управљач ресурсима ради заједно са компонентама које су додате YARN-у. Ту спадају надзорник апликације (енг. *A per-application ApplicationMaster*), управљач чвора (енг. *A per-node slave NodeManager*) и контејнер чвора (енг. *A per-application Container running on a NodeManager*). Надзорник апликације је задужен да у сарадњи са управљачем ресурса преговара о ресурсима и ради заједно са управљачем чвора како би пратили и извршили неки задатак. Управљач чвора је тај који преговара о извршењу задатака и шаље извештаје управљачу ресурса. Када планер "узме" ресурсе који су потребни за неку апликацију, он их "ставља" у контејнер.

Доприноси система YARN:

- Скалабилност – како моћ процесирања и количина ресурса константно расту у центрима за обраду података, могућности YARN-а су добродошле. Постојање компоненте планера умногоме олакшава управљање великим *Hadoop* кластерима.
- Компатибилност са моделом *Map-Reduce* – већ написане апликације могу да раде на YARN-у, који је у новијим верзијама постављен као међуслој између *Map-Reduce-а* и *HDFS-а*.
- Побољшана искоришћеност кластера – у претходној верзији *Hadoop-а* постојали су именовани слотови за сваки *Map-Reduce* процес, касније је то избачено и само резервисање ресурса је довело до великог олакшања. Много

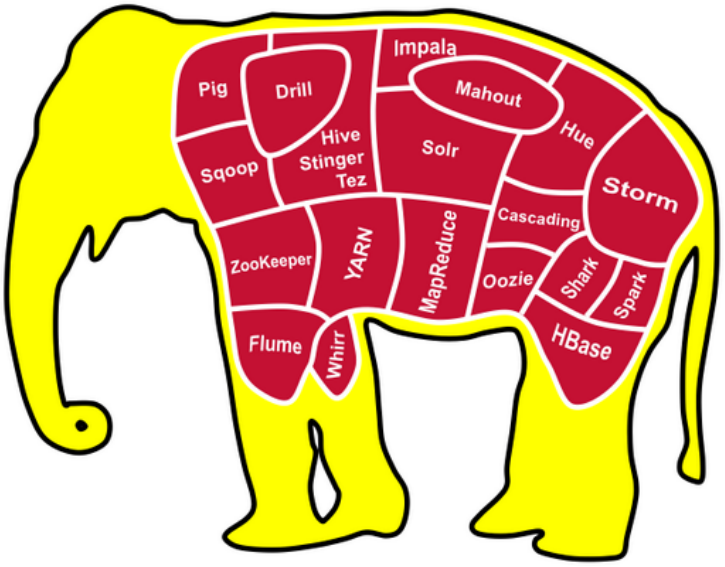
је боље имати контејнер који је гарантован у односу на потребе, него именовани слот који баш и није флексибилан.

- Могућност обраде података у реалном времену – са YARN-ом је могуће писати апликације по систему процесирања графова и итеративном моделу који се комбинују са *Map-Reduce-ом*. Стиче се утисак да се резултати неке обраде добијају скоро у реалном времену.
- Подржана је агилност, што је посебно добро при писању апликација за *Hadoop*.

2.4 Екосистем Hadoop-а

Екосистем *Hadoop-а* је скуп алата, односно пројеката који могу да раде на платформи *Hadoop*. У те алате спадају и *HDFS* и *Map-Reduce*, који су и делови платформе, али се воде и као *Apache* пројекти. Ако *Hadoop* гледамо као једну целину, на њега је могуће додати велики број разних алата, великом већином развијених од *Apache* фондације. Ти алати су способни да комуницирају са *Hadoop* компонентама и њихов рад може да буде независтан, а неки од њих су прављени само за *Hadoop* [10]. Ту спадају:

- дистрибуирани систем датотека - *HDFS*
- алати за дистрибуирано програмирање - *Map-Reduce, Apache Pig, Apache Tez*
- *NoSql* базе података - *Apache HBase, Apache Accumulo*
- SQL базе података - *Apache Hive, Apache HCatalog*
- алати за уношење података - *Apache Flume, Apache Sqoop, Apache Storm*
- алати за програмирање сервиса - *Apache Zookeeper*
- алати за рапоређивање (енг. *scheduling*) - *Apache Oozie* и *Apache Falcon*
- алати за машинско учење - *Apache Mahout*
- алати који олакшавају инсталацију компоненти - *Apache Ambari, HUE*
- безбедносни алати - *Apache Knox*



Слика 2.3: Екосистем *Hadoop-а*

Неки од чешће коришћених алата из екосистема *Hadoop-а* су:

- *Apache HCatalog* је алат који омогућава лакше управљање табелама и складиштења података на *Hadoop-у*, што корисницима разних алата пружа могућност за анализу података и омогућује им да лакше читају и пишу податке. *HCatalog* је слој на *Hadoop-у* који омогућава презентовање података са *HDFS-а* у виду табела и ослобађа кориснике брига о томе где су и у којем формату сачувани подаци. *HCatalog* може да прикаже податке из текстуалних фајлова, који су у формату *RCFile* или *CSV*, и све их приказује у табеларном облику.

- *Apache Hive* је инфраструктура за складиштење података изграђена на платформи *Hadoop*. Служи за обраду података и представља подразумевани стандард за SQL упите над великим количинама података, односно *BigData*. *Hive* пружа програмски језик врло сличан SQL-у, под називом *HiveQL*. Користи се да олакша организовање и складиштење великих количина различитих података из различитих извора. Пружа могућност корисницима да претраже, структуирају и анализирају податке за пословну интелигенцију (енг. *business intelligence*). Начин рада је веома сличан релационом моделу, табеле су сличне

табелама из релационог модела, а подаци су организовани у поретку од већих ка мањим јединицама. Базе су сачињене од табела, а табеле од партиција. Подацима се приступа на основу упита који су скоро идентични SQL-у, али је разлика у томе што *Hive* не подржава брисање и ажурирање података.

- *Apache HBase* је нерелациона (енг. *NoSql*), дистрибуирана база података написана у програмском језику Јава. Представља надоградњу Гугловог пројекта *Bigtable*. Ова база података је предвиђена да ради на *Hadoop-у*, тачније на *HDFS-у*. *HBase* је база података оријентисана на колоне и веома је отпорна на отказивања. Једна од најбитнијих особина јој је брз приступ подацима који су раштркани, што је од велике помоћи ако се обрађује неколико терабајта података, односно неколико милијарди редова, а потребно је на пример 10 највећих вредности из целе табеле. *HBase* нуди и трансакционе операције корисницима, попут ажурирања, додавања или брисања података, али ипак не може да се користи као потпуна замена за традиционалне трансакционе базе података. Основне предности које доноси су флексибилни модел података, квалитетна подршка у програмском језику Јава и обрада података у реалном времену. Табеле ове базе могу да служе као улаз или као излаз за задатке из *Map-Reduce*, што може да буде изузетно практично и применљиво.

- *Apache Sqoop* је алат који служи за пренос података из и у кластер *Hadoop*. Његова посебна намена је да ради са релационим базама података. Осим трансфера података у *HDFS*, *Sqoop* може директно да ради са *Hive-ом* или *HBase-ом*. Предност *Sqoop-а* огледа се у брзом преносу података унутар *HDFS-а*.

- *Apache Spark* је пројекат отвореног кода који се развија под покровитељством фондације *Apache*, а чини алат за брзу и ефикасну обраду великих количина података и један је од најлакших алата за учење и развој *BigData*. Представља јединствени алат за развој апликација за обраду података, било да се ради о пакетној, проточној или интерактивној обради података. Развој је врло лак јер се користе разноврсни и једноставни интерфејси, који значајно оптимизују перформансе. Лако се интегрише у различите системе за чување података, иако су премештање података између система и интеграција са различитим компонентама скупе операције.

Spark омогућава коришћење RAM меморије кластера рачунара за извршавање комплексних и захтевних послова при раду са подацима. Веома је лако проширити укупне капацитете кластера у *Spark*-у једноставним додавањем RAM меморије или додавањем нове машине у кластер. Податке представља кроз своје богате структуре података, као што су скупови података отпорни на дистрибутивност (енг. *Resilient Distributed Datasets - RDD*), оквири података (енг. *DataFrames*) и скупови података (енг. *DataSets*). Поред меморије, подаци се могу складиштити и на диску, а та одлука је препуштена програмеру. При обради података ове структуре се могу чувати у радној меморији, што значајно повећава перформансе обраде. Структуре података у *Spark*-у су непроменљиве, па тако свака операција над неком структуром ствара нову структуру.

Грешка која се често прави када је реч о позицији *Spark-а* у *BigData* сфери јесте да се *Spark* сматра заменом за *Hadoop*. Уколико је потребно ажурирати податке у некој табели, *Spark* не представља добар избор. У оквиру *Spark SQL-а* не постоји наредба за мењање података. Формати фајлова са којима *Spark* најчешће ради нису оптимизовани за операције мењања података, јер је потребно најпре обрисати старе записе и затим додати нове, што може да произведе велику количину малих фајлова. Добра пракса приликом рада са *Spark-ом* јесте избегавање рада са малим датотекама, јер таква ситуација лоше утиче на перформансе. *Spark* нема своје складиште података и доста се ослања на *HDFS*, а може да користи као изворе и нерелационе базе података, при чему се често користи *Apache Cassandra*.

- *MRUnit* је алат за тестирање јединица кода, посебно прилагођен тестирању *Map-Reduce* задатака, који умногоме олакшава рад [11]. *Hadoop* обично ради у дистрибуираном окружењу, на више кластера и више Јава виртуелних машина, тако да тестови чине развој једноставнијим, јер омогућавају да се тестирани случај изолује колико год је могуће. Један приступ је традиционални: одреди се улаз, покрене се посао или један одређени део посла и упоређује се да ли је излаз једнак очекиваном. Провере (енг. *assertions*) се пишу ручно. Други приступ је да се унапред одреде и улаз и излаз и да се развојном оквиру препусти да све сам одради.

2.5 Предности и недостаци *Hadoop-а*

Програмски модел *Map-Reduce* се најбоље показао у апликацијама које се баве обрадом логова, претраживањем и индексирањем на Интернету, затим њиховом анализом, кластеровањем и класификовањем. Такође често се користи у машинском учењу, за прављење система за препоручивање, у бионформатици и анализи генома [12].

Врло важно правило у *Map-Reduce* је правило недељивости, што значи да су сви рапоређивачи међусобно независни, а када се заврши распоређивање, започиње фаза прикупљања. Између њих не постоји дељење нити података, нити критичних региона, јер би то успорило дистрибуирано израчунавање. Правило недељивости значајно олакшава писање функција *Map()* и *Reduce()* и омогућава ефикаснију паралелизацију послова. Уколико је потребно, могуће је дефинисати неке непроменљиве структуре података, које би се прослеђивале између послова.

Недостатак модела *Hadoop Map-Reduce* је да се процес мора поделити на фазу распоређивања и фазу прикупљања, што се не може увек применити, нпр. када се требају спојити различити скупови података. Код многих алгоритама је од кључног значаја постојање глобалних структура са дељивим приступом током израчунавања, што је врло тешко имплементирати у *Map-Reduce*, јер су функције *Map()* и *Reduce()* међусобно независне. Такође, послови распоређивања и прикупљања имају одређени трошак при покретању процеса. То је прихватљиво, јер у већини случајева је тај трошак амортизован током обраде многих парова кључ/вредност. Међутим, за мале скупове података тај трошак при покретању се не може толерисати. Алтернатива је да се не користе глобална стања и да се делови алгоритма извшавају паралелно, на различитим деловима података. Коначно решење се добија спајањем појединачних резултата. *Map-Reduce* се заснива на читању података са хард диска, што врло често успорава перформансе. За неке сложеније операције се препоручује рад у Јави. Остали језици могу бити неефикасни, поготови ако се не ради са текстом него нпр. бројевима у децималном запису.

Проточни алгоритми (енг. *streaming*) представљају алтернативни програмски модел који се бави великим количинама података са ограниченим рачунским и складишним ресурсима. Овај модел претпоставља да су подаци представљени алгоритму као један или више токова улаза који се обрађују по реду и само једном. Извори токова података могу бити датотеке у дистрибуираном фајл систему, али

и подаци из спољних извора или неког другог уређаја за прикупљање података. Може се користити за рад са временским серијама (нпр. вести, твитови, логови са сензора). Пошто су проточни алгоритми релативно једноставни они често могу искористити предност модерних графичких картица, које имају велики број релативно једноставних процесорских јединица.

Глава 3

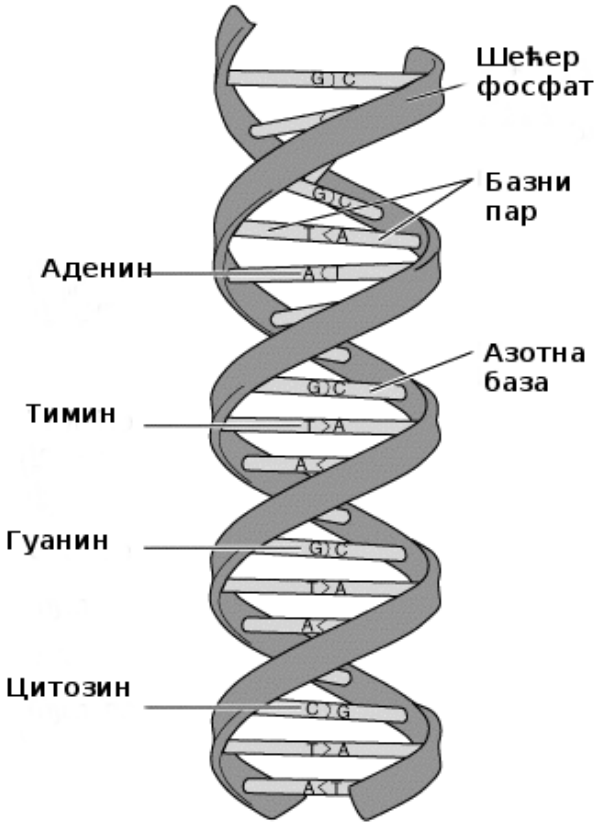
Геномске секвенце

3.1 ДНК - општи појмови

ДНК - Дезоксирибонуклеинска киселина (енг. *DNA*) молекул је који кодира генетичке инструкције које се користе за развој и функционисање свих познатих живих организама и многих вируса. Поред РНК и протеина, ДНК је један од три главна макромолекула неопходна свим познатим облицима живота. ДНК се састоји од дугих, спиралних ланаца чије карике чине парови четири базе: аденина (А) и тимина (Т), односно цитозина (Ц) и гуанина (Г), као и шећера дезоксирибозе и фосфатне групе (слика 3.1). ДНК је код људи у једрима ћелија организован у 23 пара који се називају хромозоми. Ово детерминистичко упаривање омогућава "механизам копирања": двострука спирала ДНК се одмотава и појављују се комплементарни базни парови, стварајући две тачне копије, оригиналне ДНК ланцу.

3.2 Секвенцирање ДНК

Секвенцирање ДНК представља читање узорка крви пацијента (улазни подаци) и стварање варијанти за узорак ДНК (излазни подаци). Формално, ДНК секвенцирање је процес одређивања прецизног поретка нуклеотида унутар молекула ДНК. Укључује било који метод или технологију која се користи за одређивање поретка четири базе из нашег генома - аденин (А), гуанин (Г), цитозин (Ц) и тимин (Т) [8].



Слика 3.1: Изглед ДНК спирале

У време кад је *Map-Reduce* алгоритам објављен трошак секвенцирања генома износио је 20 милиона долара и било је потребно неколико месеци рада да би се добила геномска секвенца. Данас секвенцирање генома кошта само неколико стотина долара и траје само неколико дана. За откривање првог људског генома биле су потребне деценије, док је процена да је само у 2014. години секвенцирано 228.000 генома широм света, што представља количину од 20 петабајта (PB). Графикон напретка у секвенцирању генома организација које користе *Hadoop* за секвенцирање у зависности од развоја *Hadoop-a* приказан је на слици 3.2.

Један од главних циљева секвенцирања ДНК јесте проналажење варијанти, пошто је већина наших ДНК идентична, а само се у малом проценту разликују. Такође је важна и идентификација генетских варијанти као што су једнонуклеотидни полиморфизми или скраћено снипови (енг. *single nucleotide polymorphisms*), што подразумева појаву замене места једног нуклеотида неким другим нуклеотидом.

На пример, уколико је посматрана секвенца на некој локацији гена у некој биолошкој врсти састављена од низа нуклеотида AAGCCTA и уколико је дошло до генетске модификације у AAGCTTA, следи да се секвенце разликују у једном нуклеотиду. Тада се говори о појави једнонуклеотидног полиморфизма. Последица тога је да се услед ове мутације јавља различит облик посматраног генома. Идентификација и издвајање снипова из сирових генетичких секвенци укључује многе алгоритме и примену разноврсног скупа алата. Поравнање секвенци је упоређивање два или више ДНК или протеинских секвенци. Главна сврха поравнавања секвенци је да истакне сличности међу њима.

Слика 3.2: Однос развоја технологија

Популарни формат за записивање скенираних делова ДНК секвенци је *FASTQ*. Формат *FASTQ* је текстуални формат за чување како биолошке секвенце, тако и њене оцене квалитета. За дату датотеку *FASTQ* једна секвенца ДНК је представљена са четири линије (слика 3.3). Свака од линија има своју улогу:

- Прва линија почиње знаком "@" и представља идентификатор секвенце и опционални опис
- Друга линија представља секвенцу слова (A, T, Ц, Г, или H за непознато) и носилац је генетске информације
- Трећа линија почиње знаком ’+’ и најављује да се у наредном реду налази опис квалитета секвенце са датим идентификатором

- Четврта линија кодира вредности квалитета за секвенцу у другом реду и мора садржати исти број симбола као слова у низу

```
@NCYC361-11a03.q1k bases 1 to 1576
GCGTGCCCGAAAAAATGCTTTTGGAGCCGCGCGTGAAAT...
+NCYC361-11a03.q1k bases 1 to 1576
!)))))****(((***%%((((*(((+,**(((+***+,-...
```

Слика 3.3: Пример записа генома у формату *FASTQ*

Задатак овог мастер рада је пописати и пребројати све подсеквенце дате дужине. Улазни подаци развијане тест-апликације су геномске секвенце записане у формату *FASTQ*. У овом примеру је битна једино друга линија која представља секвенцу слова (А, Ц, Г, Т, или Н за непознато) и носилац је генетске информације, док се остале занемарују.

Постоји више изазова приликом секвенцирања ДНК:

- Постоји неколико техника за генерисање датотека *FASTQ*, јер се дужина ДНК секвенци разликује у зависности од технологије секвенцирања
- Величина улазних података може бити огромна. Величина узорка појединачне секвенце ДНК може бити до 200GB
- Једном снажном серверу потребно је до 80 сати да обради узорак података и добије снипове
- Постоје многи алгоритми и кораци за секвенцирање ДНК, па одабир одговарајућих комбинација алата који су обично отвореног кода представља озбиљан изазов
- Скалабилност, јер треба оптимизовати број распоређивача и прикупљача

Глава 4

Преглед технологија

4.1 Јава

Јава (енг. *Java*) представља програмски језик опште намене који је конкурентан, класно заснован, објектно-оријентисан и од верзије Јава 8 функционалан програмски језик, специјално дизајниран да има што је могуће мање имплементационе зависности. Компајлирани Јава код може да се покреће на свим платформама које подржавају Јаву без потребе за рекомпајлирањем, односно језик функционише по принципу "пиши једном, покрени било где". Јава апликације се преводе на бајткод, који се покреће на било којој Јава виртуелној машини (енг. *JVM*), без обзира на архитектуру рачунара. Јава је један од најпопуларнијих програмских језика у примени, користи се у разним типовима апликација попут Веб, мобилних, али и огромних серверских апликација [13].

Јаву је развио Џејмс Гослинг 1995. године у компанији *Sun Microsystems*, која је касније постала део корпорације *Oracle*. Синтакса великим делом потиче из језика C и C++, али за разлику од њих има објекте ниског нивоа. Програмски модел *Map-Reduce* је изворно написан у Јави, као и већина алата у *Hadoop* екосистему. Данас се доста користе и Пајтон (енг. *Python*) и Скала (енг. *Scala*), међутим, Јава и даље има бољу подршку и већу улогу у екосистему *Hadoop-a*.

4.2 *Bash* скрипте

Bash (Bourne-again shell) је интерпретер и програмски језик у облику командне линије који је написао Брајан Фокс (*Brian Fox*) 1987. године за пројекат ГНУ (енг. *GNU's not Unix*). Представља подразумевани интерпретер на већини Линукса. Садржи сопствени скриптни језик, помоћу којег је могуће остварити висок ниво аутоматизације послова јер подржава петље, гранања, променљиве и управљање њиховим вредностима уз помоћ регуларних израза. Подржава аритметичке операције, управљање стандардним улазом, излазом и излазом за грешке и преусмеравање у датотеке. Врло корисне функционалности су аутоматско завршавање команди и имена датотека и директоријума, историја претходних команди и уређивање командне линије [14]. Свака скрипта почиње командом која одређује тип командног интерпретера:

```
#!/bin/sh
```

Пошто се платформа *Hadoop* ослања на Линукс потребно је основно знање команди за свакодневни рад у окружењу. Такође, команде за рад на систему HDFS су направљене по узору на *Bash*. Неке од основних команди приказане су у примеру 4.1

```
#!/bin/sh
#napravi novi direktorijum input
hdfs dfs -mkdir /input
#izlistaj sadrzaj direktorijuma input
hdfs dfs -ls /input
#kopiraj na HDFS iz lokalnog direktorijuma
hdfs dfs -copyFromLocal /user/tmp1.txt input/
#ispis meta podataka: broj blokova, ispravnost...
hdfs dfs -copyToLocal /user input/tmp1.txt
#kopiraj sa HDFS-a na lokalni direktorijum
hdfs fsck /input
#dodela svih prava direktorijumu input
hdfs dfs -chmod 777 /input
```

Пример кода 4.1: Опис често коришћених HDFS команди

4.3 *Apache Maven*

Apache Maven је софтверски алат за управљање и изградњу пројекта. Помоћу овог алата се може припремити кôд за дистрибуцију. Такође, уз помоћ *Maven-a* у пројекат можемо на лак начин укључити Јава библиотеке (енг. *jars*) које су нам потребне. Сва правила дефинишемо у датотеци **pom.xml**, која мора садржати групу, предмет за употребу, верзију и назив пројекта. У истој датотеци дефинишемо и начин паковања, да ли ће бити jar (енг. *Java Application Arhive*) или war (енг. *Web Application Arhive*). У *dependencies* делу pom.xml-а дефинишу се библиотеке које су потребне пројекту. Без *Maven-a* се све библиотеке морају поставити ручно [15].

Постоји неколико фаза животног циклуса изградње пројекта: валидација пројекта, компилација изворног кôда, при чему се кôд смешта у директоријум *target* у пројекту, паковање где се компајлиран кôд из *target* директоријума пакује у формат за дистрибуцију, на пример, jar. Затим се направљени пакет инсталира на локалном репозиторијуму да би пројекат могао да се користи локално у другим пројектима као зависност. Последња фаза је распоређивање. У овој фази се пакет копира на удаљени репозиторијум и на тај начин постаје доступан осталим развојним инжењерима и пројектима. Још једна често коришћена опција је чишћење (енг. *clean*). Target директоријум се празни пре извршавања главне *Maven* команде, тј. пре поновне компилације кôда.

```
   <dependencies>
       <dependency>
           <groupId>org.apache.hadoop</groupId>
           <artifactId>hadoop-mapreduce-client-core</artifactId>
           <version>2.8.0</version>
       </dependency>

       <dependency>
           <groupId>org.apache.hadoop</groupId>
           <artifactId>hadoop-common</artifactId>
           <version>2.8.0</version>
       </dependency>
</dependencies>
```

Пример кода 4.2: Пример укључивања потребних библиотека у пројекат

4.4 Лог4j

Пошто се *Map-Reduce* извршава паралелно, веома је тешко квалитетно дебаговати задатке написане за *Map-Reduce*. Због тога је битно да се направи добро логирање извшавања апликације, како би се лакше пронашао проблем и приликом развоја, а посебно при одржавању апликације у току извршавања. У Јава свету најпознатији алат за логирање је Лог4j (енг. *Log4j*). Дистрибуира се под лиценцом *Apache Software* [16]. Лог4j се може конфигурисати преко екстерних конфигурационих датотека у току извршавања, обично као *.xml* или *.properties* формат. Надгледа процес логирања у смислу дефинисања нивоа приоритета и нуди механизме за усмеравање информација о евиденцији до великог броја дестинација, као што су база података, датотека, конзола, системски логови, итд. Користи више нивоа, у које по приоритету спадају: *ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL*.

```
log4j.rootLogger=INFO, file, stdout
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=/home/nemanja/Downloads/logging.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
```

Пример кода 4.3: Пример како се подешава упис логова у датотеку

Глава 5

Анализа геномских секвенци

5.1 Задатак

Задатак овог мастер рада је пописати и пребројати све подсеквенце дате дужине. Улазни подаци развијане тест-апликације су геномске секвенце. Секвенце су записане у раније описаном формату *FASTQ* и налазе се у датотеци *fastq.txt*. Запис једне секвенце се састоји од 4 линије, од којих свака има своју улогу. У овом примеру је битна једино друга линија која представља секвенцу слова (А, Ц, Г, Т, или Н за непознато) и носилац је генетске информације, док се остале занемарују. Процес се дели на две фазе: прва је распоређивање свих могућих подсеквенци, а друга фаза је обрада и пребројавање тако сређених подсеквенци.

5.2 Коришћена *Hadoop* архитектура

Платформа *Hadoop* се може бесплатно преузети са веб стране *Apache* фондације *http://hadoop.apache.org/*, где се налазе и упутства за инсталацију на оперативним системима Линукс и Виндоуз. Постоје три начина за инсталацију:

- Локални режим рада (енг. *Standalone Mode*) - инсталација на једном рачунару, при чему се кластер састоји од само једног чвора. Корисно за учење и откривање грешака у коду.

- Псеудо-дистрибуирани режим рада (енг. *Pseudo-Distributed Mode*) - симулација кластера од неколико чворова система *Hadoop* на једном рачунару. Углавном се користи за учење и истраживање.
- Потпуно-дистрибуирани режим рада (енг. *Fully-Distributed Mode*) - кластер који се састоји од већег броја чворова. Рачунари су подељени у групе на основу физичке удаљености у мреже рачунара (енг. *Rack*). Овај начин је прикладан за употребу у продукцији.

Постоје компаније које развијају сопствене дистрибуције *Hadoop-a* и чије су главне предности олакшана инсталација и систем корисничке подршке. У познатије спадају:

- *Cloudera Distribution Hadoop (CDH)*
- *Hortonworks Data Platform (HDP)*
- *MapR*
- *Microsoft Azure HDInsight: Hadoop in the Azure cloud*
- *IBM BigInsights*
- *Amazon Web Services: Amazon Elastic MapReduce (Amazon EMR)*

Наведене платформе пружају широку палету услуга, од којих су неке бесплатне. Садрже и неке додатне апликације које олакшавају одржавање и преглед рада система *Hadoop*. У овом раду коришћена је основна компонента *Apache Hadoop* која је инсталирана на једном рачунару у локалном режиму рада (јер је за псеудо-дистрибуирани потребно више радне меморије рачунара), али се генерално за продукциону употребу препоручује нека од горе наведених дистрибуција.

5.3 Структура пројекта

За потребе демонстрације система *Hadoop Map-Reduce* направљен је *Maven* Јава пројекат. Пројекат је развијен у окружењу *IntelliJ IDEA Community edition 2016.3*.

Коришћени су Јава 8, *Apache Hadoop 2.8.0*, *Maven* за довлачење потребних библиотека, као и Лог4j за логирање. Акценат се ставља на главне делове, а то су функције распоређивања и прикупљања, као и иницијално читање генетских података у формату *FASTQ*. Целокупан изворни код апликације доступан је на адреси: https://github.com/munjaza/fastq.

Структура пројекта је следећа:

- **src/main/java/fastq**

  Подразумевани читач у *Hadoop-у* чита улазне податке линију по линију, али за формат *FASTQ* се запис једне геномске секвенце састоји од четири реда. Класе за рад са форматом датотека *FASTQ* преузете су од професора Махмуда Парсиана и јавно су доступне на адреси: https://github.com/mahmoudparsian/data-algorithms-book.

  Класе које се користе за учитавање геномске секвенце:

  - *BaseComparator* - дефинише начин како се пореде подсеквенце. У овом случају се користи лексикографско поређење.
  - *BasePartitioner* - дефинише како ће се вршити партиционисање блокова пре фазе прикупљања.
  - *FastqInputFormat* - помоћна класа која иницира нови објекат за читање записа *FastqRecordReader*.
  - *FastqRecordReader* - дефинише препакивање на начин да се четири линије које чине један запис спајају у једну и међусобно раздвајају знаковима ",;,". Након тога је врло једноставно у методи за распоређивање *Mapper()* извршити поделу ниске и извући други члан низа као што је приказано у скрипти 5.2.

- **src/main/java/impl**

  Пакет у којем су дефинисане основне класе за приказ модела *Map-Reduce*. Ту спадају:

  - *FastqCountSubsequences* - основна класа у којој су дефинисани распоређивач, прикупљач, датотека са улазним подацима и датотека где ће се генерисати излазни подаци приказана у скрипти 5.1.

- *FastqMapper* - класа која служи за распоређивање, приказана у скрипти 5.2.
- *FastqReducer* - класа која служи за прикупљање, приказана у скрипти 5.3.

Треба нагласити да се покрећу два задатка за *Map-Reduce* који чине један процес рада (енг. *workflow*) како би се добио излаз сортиран по броју појављивања подсеквенци у опадајућем редоследу. Наиме, први задатак врши формирање подсеквенци дате дужине и њихово пребројавање. Као резултат добија се несортирани излаз. Други задатак врши само сортирање и ту није нагласак на методама за распоређивање и обједињавање, већ на методама за поређење и сортирање.

У класи *FastqCountSubsequences* метода која прима аргументе командне линије *int run(String [] args)* покреће задатак за *Map-Reduce* и извештава да ли је задатак успешно извршен. Као улазни подаци шаљу се називи улазног директоријума и директоријума где ће се поставити резултат извршавања, као и параметар који одређује дужину тражене подсеквенце. Потребно је дефинисати улазни и излазни формат података, као и које ће се класе користити за њихово парсирање. Основно је да се дефинишу класе за распоређивање (*Mapper*) и прикупљање (*Reducer*), које се разликују у зависности од задатка. У првом задатку је дефинисано да улогу комбинатора, тј. локалног прикупљача, преузме класа *FastqReducer*. Такође, може се видети да су у задацима искоришћене различите класе за партиционисање и поређење при сортирању и груписању. Битно је напоменути да је у другом задатку дефинисано да се користи само један прикупљач, како би резултат била једна датотека са сортираним садржајем.

```
public int run(String [] args) throws Exception {

  String INPUT_PATH = args[0];
  String TMP_PATH = args[1];
  String OUTPUT_PATH = args[2];

  logger.info("run(): input  args[0]=" + INPUT_PATH);
  logger.info("run(): tmp args[1]=" + TMP_PATH);
  logger.info("run(): output args[2]=" + OUTPUT_PATH);

  Configuration conf = new Configuration();
  if(args.length == 4) {
    conf.set("length", args[3]);
```

```
    logger.info("run(): length args[3]=" + args[3]);
  }

  Job job = Job.getInstance(conf, "Fastq count");
  job.setJarByClass(FastqCountSubsequences.class);
  job.setInputFormatClass(FastqInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);
  job.setSortComparatorClass(BaseComparator.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  job.setMapperClass(FastqMapper.class);
  job.setCombinerClass(FastqReducer.class);
  job.setReducerClass(FastqReducer.class);

  FileInputFormat.setInputPaths(job, new Path(INPUT_PATH));

  if(TMP_PATH.equals("output/")) {
    FileSystem.get(getConf()).delete(new Path(TMP_PATH), true);
    TextOutputFormat.setOutputPath(job, new Path(TMP_PATH));
  }

  boolean status = job.waitForCompletion(true);

  if(status) {
    Job job1 = Job.getInstance(conf, "Fastq sort");
    job1.setJarByClass(FastqCountSubsequences.class);
    job1.setInputFormatClass(KeyValueTextInputFormat.class);
    job1.setOutputKeyClass(SequencePair.class);
    job1.setOutputValueClass(NullWritable.class);
    job1.setMapperClass(SortingMapper.class);
    job1.setPartitionerClass(SequencePartitioner.class);
    job1.setSortComparatorClass(SequenceComparator.class);
    job1.setGroupingComparatorClass(GroupComparator.class);
    job1.setReducerClass(SortingReducer.class);
    job1.setNumReduceTasks(1);
    job1.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.setInputPaths(job1, new Path(TMP_PATH));
    if (OUTPUT_PATH.equals("outputSort/")) {
      FileSystem.get(getConf()).delete(new Path(OUTPUT_PATH), true);
      TextOutputFormat.setOutputPath(job1, new Path(OUTPUT_PATH));
    }
```

```
      job1.setOutputFormatClass(TextOutputFormat.class);
      status = job1.waitForCompletion(true);
    }

    logger.info("run(): status="+status);
    return status ? 0 : 1;
  }
```

Пример кода 5.1: Задатак за Map - Reduce

У класи *FastqMapper*, која служи за распоређивање, потребно је предефинисати методу *map()* на начин који одговара случају који се обрађује. Пошто је улазни формат *FASTQ*, тип улазне вредности биће станардна класа *Text* за развојни оквир *Hadoop*. Кључ се у овом случају не користи, па ће бити типа *Object*. Резултат рада је пар кључ/вредност, тј. подсеквенца и број појављивања у реду, што је приказано у методи *void cleanup(Context context)*. Логика како се ради је да је улазна секвенца подељена на подсеквенце дате дужине, које се чувају у листи. Уколико се нека подсеквенца појави више пута у једној секвенци увећава се вредност бројача, иначе се поставља на вредност један.

```
public class FastqMapper extends Mapper<Object, Text, Text, IntWritable
    > {

  private Map<String, Integer> dnaBaseCounter = null;
  private IntWritable counter = new IntWritable();
  private Text base = new Text();

  protected void setup(Context context) throws IOException,
    InterruptedException {
     dnaBaseCounter = new HashMap<String, Integer>();
  }

  protected void cleanup(Context context) throws IOException,
    InterruptedException {
     for (Map.Entry<String, Integer> entry : dnaBaseCounter.entrySet())
    {
       base.set(entry.getKey());
       counter.set(entry.getValue());
       context.write(base, counter);
     }
  }
```

```
  public void map(Object key, Text value, Context context) throws
   IOException, InterruptedException {
    String fastqRecord = value.toString();
    String[] lines = fastqRecord.split(",;,");
    String sequence = lines[1].toUpperCase();
    List<String> subSequences = new ArrayList<String>();

    Configuration conf = context.getConfiguration();
    Integer length;
    if(conf.get("length") != null) {
      length = Integer.valueOf(conf.get("length"));

      for (int i = 0; i <= sequence.length() - length; i++) {
        String s = sequence.substring(i, i + length);
        if (s.matches("[a-zA-Z]+")) {
          subSequences.add(s);
        }
      }
      for (String s : subSequences) {
        Integer count = dnaBaseCounter.get(s);
        if (count == null) {
          dnaBaseCounter.put(s, 1);
        } else {
          dnaBaseCounter.put(s, count + 1);
        }
      }
    }
  }
}
```

Пример кода 5.2: Класа која служи за распоређивање

У класи *FastqReducer*, која служи за обједињавање, потребно је предефинисати методу *reduce()* на начин који одговара случају који се обрађује. Улазни параметри су подсеквенца дате дужине и број њених појављивања у иницијалној секвенци. Пошто су се пре позива ове класе извршиле фазе распоређивања и партиционисања, подсеквенце су груписане на исте чворове и сада се врши само просто сабирање резултата из фазе распоређивања.

```
public class FastqReducer extends Reducer<Text, IntWritable, Text,
    IntWritable> {
  public void reduce(Text key, Iterable<IntWritable> values,  Context
    context) throws IOException, InterruptedException {
    Integer total = 0;
    for(IntWritable val: values){
      total += val.get();
    }
    context.write(key, new IntWritable(total));
  }
}
```

Пример кода 5.3: Класа која служи за прикупљање

- **src/main/java/impl**

  Пакет у којем су дефинисане класе које се користе за сортирање резултата у другом задатку за *Map-Reduce*. Кључне фазе у овом процесу су сортирање и партиционисање, где се одвија сав посао овог задатка.

  Састоји се од класа:

  – *SequencePair* - класа коју чине подсеквенца и њен бројач. Садржи предефинисану методу за начин поређења по вредности бројача: *int compareTo(SequencePair sequencePair)*.

  – *SequenceComparator* - одређује да се поређење врши у опадајућем редоследу.

  – *GroupComparator* - дефинише да се груписање врши по подсеквенцама.

  – *SequencePartitioner* - дефинише начин како се врши партиционисање.

  – *SortingMapper* - класа за распоређивање која као кључ враћа пар (подсеквенца/број појављивања), а као вредност предефинисану класу *NullWritable* из развојног оквира *Hadoop*.

  – *SortingReducer* - класа за прикупљање, која не ради ништа ново, само пролеђује на излаз оно што је стигло као резултат сортирања.

Глава 6

Резултати

Као пример анализе геномских секвенци у формату *FASTQ* коришћена је датотека величине 5.2MB, која се састоји од 65532 линије, тј. 16383 геномских секвенци, јер је свака секвенца одређена са четири линије, према формату *FASTQ*.

```
#!/bin/sh

cd /home/hadoop/fastq/

hdfs dfs -rm -r /user/hadoop/input/* /user/hadoop/output/* /user/hadoop/
    outputSort/*
hdfs dfs -mkdir -p /user/hadoop/input /user/hadoop/output /user/hadoop/
    outputSort
hdfs dfs -copyFromLocal /home/hadoop/fastq/input/fastq.txt /user/hadoop/
    input

hadoop jar fastq.jar impl.FastqCountSubsequences -Dmapreduce.map.memory.mb
    =8192 -Dmapreduce.reduce.memory.mb=8192 -Dmapreduce.job.maps=11 input/
    output/ outputSort/ 15

rm -rf outputSort/part-r-00000 outputSort/_SUCCESS
hdfs dfs -copyToLocal /user/hadoop/outputSort /home/hadoop/fastq/
```

Пример кода 6.1: Скрипта којом се покреће задатак из командне линије

У скрипти 6.1 дефинисан је начин како се из командне линије оперативног система Линукс, на коме је инсталирана платформа *Hadoop*, покреће задатак. Потребно је да на путањи */home/hadoop/fastq/input/* постоји датотека *fastq.txt*. У скрипти

је описан начин како се датотека пребацује на систем датотека *HDFS* командом *-copyFromLocal*, затим како се покреће апликација за подсеквенце дужине 15, на основу формиране извршне датотеке *fastq.jar*, а логирање се преусмерава у датотеку *joboutput*. На крају се крајњи резултат пребацује на локални систем датотека командом *-copyToLocal*.

Скрипта се може покренути неограничен број пута, јер су имплементиране потребне операције брисања и прављења нових датотека. Пример како изгледа улаз у формату *FASTQ* приказан је у скрипти 6.2. Као резултат настаје датотека *SUCCESS*, која служи као провера да ли је задатак успешно извршен. У зависности од броја прикупљача може бити више датотека са траженим израчунавањем. Прва има назив *part-r-00000* и може се видети у скрипти 6.3. У приказаној апликацији у другом задатку за *Map-Reduce* је дефинисано да ће се користити један прикупљач да би резултат била једна датотека са сортираним садржајем. У случају да нема фазе прикупљања тада је формирана датотека под називом *part-m-00000*, настала као резултат фазе распоређивања.

Апликација је за потребе тестирања покренута на две различите рачунарске конфигурације. У првом случају коришћен је кућни рачунар на коме је и развијена апликација. Укупно време трајања извршавања задатка било је око 5 минута. Други случај представља кластер од три виртуелне машине, подигнуте у облаку са доста јачим перформансама. Тада је задатак одрађен за нешто мање од једног минута, што управо представља праву снагу система *Hadoop*.

```
@ILLUMINA-GA_62844AAXX:2:1:1001:4564#0/1
NATCACTCGGCCTTGTTGACTCAGCATCTGAAGCGTATGAAAGACAGTGGCCAACTCGATATGGT
+ILLUMINA-GA_62844AAXX:2:1:1001:4564#0/1
BMONOUXUVW\]]^^OVVYY_^X_^BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
@ILLUMINA-GA_62844AAXX:2:1:1001:12442#0/1
NACGAACACGACCATTCTTACGTCGACCTACAAGTTGAGAAAAAGCTTCATCTTGAGCAGCAGGG
+ILLUMINA-GA_62844AAXX:2:1:1001:12442#0/1
BSTSSXWVVUZ__^_\\[XX[V[[ZWXWWW_‘V]^\]X[[[XMNN^^K^^_^_BBBBBBBBBBB
@ILLUMINA-GA_62844AAXX:2:1:1001:17295#0/1
NTCTAGGAGAGTTCCAATTTTCCTGTGTCTTATTCACTGCAGTTTTAGAGAATGCTTGCCGTGAG
+ILLUMINA-GA_62844AAXX:2:1:1001:17295#0/1
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

Пример кода 6.2: Првих неколико линија улазне датотеке у формату *FASTQ*

```
TTTTTTTTTTTTTTT 2438
CCCCCCCCCCCCCCC 1660
AAAAAAAAAAAAAAA 268
GATCGGAAGAGCGGT 211
CGGAAGAGCGGTTCA 207
ATCGGAAGAGCGGTT 207
TCGGAAGAGCGGTTC 205
GGAAGAGCGGTTCAG 205
GAGCGGTTCAGCAGG 205
GAAGAGCGGTTCAGC 204
AGCGGTTCAGCAGGA 202
```

Пример кода 6.3: Првих неколико линија добијеног резултата

Добијена датотека је величине 25.5MB, где су приказане геномске секвенце сортиране према броју појављивања опадајуће. У скрипти 6.4 приказан је извештај о успешно извршеном задатку *Hadoop Map-Reduce*. Извештај приказује колико је било улазних секвенци, затим након фазе распоређивања колико је парова формирано, како је извршено мешање, сортирање и партиционисање и на крају колико је парова добијено као резултат фазе прикупљања података.

```
7/08/31 09:42:56 INFO input.FileInputFormat: Total input paths to process:1
17/08/31 09:42:56 INFO mapreduce.JobSubmitter: number of splits:1
17/08/31 09:42:56 INFO mapreduce.JobSubmitter: Submitting tokens for job:
   job_1504163382128_0011
17/08/31 09:42:57 INFO impl.YarnClientImpl: Submitted application
   application_1504163382128_0011
17/08/31 09:42:57 INFO mapreduce.Job: The url to track the job: http://
   nemanjaDataNode:8088/proxy/application_1504163382128_0011/
17/08/31 09:42:57 INFO mapreduce.Job: Running job: job_1504163382128_0011
17/08/31 09:43:03 INFO mapreduce.Job: Job job_1504163382128_0011 running in
    uber mode : false
17/08/31 09:43:03 INFO mapreduce.Job:  map 0% reduce 0%
17/08/31 09:43:19 INFO mapreduce.Job:  map 67% reduce 0%
17/08/31 09:43:33 INFO mapreduce.Job:  map 100% reduce 0%
17/08/31 09:43:43 INFO mapreduce.Job:  map 100% reduce 100%
17/08/31 09:43:44 INFO mapreduce.Job: Job job_1504163382128_0011 completed
   successfully
17/08/31 09:43:44 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=9572040
    FILE: Number of bytes written=19539218
```

```
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=5195164
    HDFS: Number of bytes written=25505224
    HDFS: Number of read operations=9
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=4
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=2
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=28387
    Total time spent by all reduces in occupied slots (ms)=13531
    Total time spent by all map tasks (ms)=28387
    Total time spent by all reduce tasks (ms)=13531
    Total vcore-milliseconds taken by all map tasks=28387
    Total vcore-milliseconds taken by all reduce tasks=13531
    Total megabyte-milliseconds taken by all map tasks=29068288
    Total megabyte-milliseconds taken by all reduce tasks=13855744
  Map-Reduce Framework
    Map input records=16383
    Map output records=1416859
    Map output bytes=28337180
    Map output materialized bytes=9572032
    Input split bytes=118
    Combine input records=1416859
    Combine output records=1416859
    Reduce input groups=1416859
    Reduce shuffle bytes=9572032
    Reduce input records=1416859
    Reduce output records=1416859
    Spilled Records=2833718
    Shuffled Maps =2
    Failed Shuffles=0
    Merged Map outputs=2
    GC time elapsed (ms)=2404
    CPU time spent (ms)=44160
    Physical memory (bytes) snapshot=1617965056
    Virtual memory (bytes) snapshot=4795645952
    Total committed heap usage (bytes)=1650458624
  Shuffle Errors
    BAD_ID=0
```

```
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=5195046
  File Output Format Counters
    Bytes Written=25505224
```

Пример кода 6.4: Део лог датотеке *joboutput*

Глава 7

Закључак

У овом раду приказане су неке од компоненти екосистема *Hadoop* и како се могу применити на обраду великих количина података, попут пребројавања геномских секвенци. Највише пажње је посвећено стандардном алгоритму за обраду великих количина података распоређивање-обједињавање (енг. *Map-Reduce*), који омогућава да се на релативно лак начин обраде велике количине података на јефтиним хардверским компонентама. У зависности од потреба корисника система врло лако се може комбиновати са осталим алатима из екосистема *Hadoop*. Као програмски језик коришћена је Јава, из разлога што је сам алат писан у Јави и што се аутор рада у тренутку писања бавио Јава програмирањем. Често коришћен програмски језик за ову платформу јесте и Пајтон, који је све више у употреби у последње време, поготово у области истраживања података.

Приликом рада било је потребно осмислити ефикасну поделу задатака. Основне компоненте система, класе за прикупљање и обједињавање, извршавају целокупан посао. Развојни оквир *Hadoop Map-Reduce* омогућава да се највише пажње при развоју апликације усмери управо на ове делове, без много напора око комуникације у дистрибуираном окружењу. На крају је, ради лепшег приказа, извршено сортирање резултата и тиме приказана добра пракса комбиновања више задатака за *Map-Reduce*. Једна од мана јесте што брзина добијања информација није на завидном нивоу, јер се подаци налазе на дисковима рачунара, што представља уско грло. У сврху превазилажења тога може се користити платформа *Apache Spark* која функционише по сличном принципу, али се подаци чувају у рачунарској меморији. То доводи до повећања цене израчунавања, што преставља један од главних услова приликом одлучивања који систем користити.

Закључак је да *Hadoop Map-Reduce* представља једну зрелу платформу за обраду великих количина података, на релативно јефтиним рачунарским конфигурацијама. Не би требало да се користи као замена за неке стандардне системе за обраду података, попут релационих база података, јер се они ту никако не могу такмичити. Међутим, када су релационе базе података немоћне да на одговарајући начин обраде дате податке, ту треба тражити улогу за *Hadoop*. Препорука је да се користи када је могуће паралелизовати послове и приликом обраде великих датотека. То је управо сфера биоинформатике, затим истраживања података насталих током телекомуникационих операција, али и у претраживању корисних информација са друштвених мрежа.

Библиографија

[1] Daniel Price. Facts and stats about the big data industry. 2015. URL http://cloudtweaks.com/2015/03/surprising-facts-and-stats-about-the-big-data-industry/.

[2] Tom White. *Hadoop The Definitive Guide, 4th Edition*. O'Reilly Media, Farnham Great Britain, 2015.

[3] San Diego University of California. Hadoop platform and application framework. 2017. URL https://www.coursera.org/learn/hadoop/.

[4] San Diego University of California. Introduction to big data. 2017. URL www.coursera.org/learn/big-data-introduction.

[5] Shun-Tak Leung Sanjay Ghemawat, Howard Gobioff. The google file system. 2003. URL https://research.google.com/archive/gfs.html.

[6] Sanjay Ghemawat Jeffrey Dean. Mapreduce: Simplified data processing on large clusters. 2004. URL https://research.google.com/archive/mapreduce.html.

[7] Adam Shook Donald Miner. *MapReduce Design Patterns*. O'Reilly Media, 2013.

[8] Mahmoud Parsian. *Data Algorithms: Recipes for Scaling Up with Hadoop and Spark*. O'Reilly Media, 2015.

[9] Званична Веб презентација apache hadoop. 2017. URL http://hadoop.apache.org.

[10] Data science srbija блог. 2017. URL http://www.datascience.rs.

[11] Mrunit tutorial. 2017. URL https://cwiki.apache.org/confluence/display/MRUNIT/MRUnit+Tutorial.

[12] Chris Dyer Jimmy Lin. *Data-Intensive Text Processing with MapReduce*. University of Maryland, College Park, 2010.

[13] 2017. URL https://en.wikipedia.org/wiki/Java_(programming_language).

[14] Gnu bash упутство. 2017. URL https://www.gnu.org/software/bash/manual/bash.html.

[15] Званична Веб презентација apache maven. 2017. URL https://maven.apache.org.

[16] Log4j туторијал. 2017. URL http://www.tutorialspoint.com/log4j/.