

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

Matas Malickas

**DIDŽIŲJŲ DUOMENŲ APDOROJIMAS**

Kursinis darbas

Tikrino: doc. dr. Andrej Bugajev

Vilnius, 2023**Turinys**

[1. Įvadas 5](#_Toc136134312)

[2. Analitinė dalis 5](#_Toc136134313)

[2.1. Vaizdų aptikimas 5](#_Toc136134314)

[3. Literatūra 6](#_Toc136134315)

**Paveikslų sąrašas**

**No table of figures entries found.**

**Lentelių sąrašas**

**No table of figures entries found.**

# Įvadas

Kursiniame darbe nagrinėjamos ir testuojamos didelių duomenų apdorojimo programinės sistemos laboratoriniams darbams. Didieji duomenys yra sudėtingi ir užima daug atminties, jų apdorojimas įprastomis duomenų bazių sistemomis yra nepatogus ir dažnai užtrunka ilgą laiką. Siekiant veiksmingai apdoroti didelius duomenų kiekius, būtina naudoti didelių duomenų sistemas, kurios gali juos apdoroti žymiai efektyviau.

Laboratoriniuose darbuose tiriami duomenys, susiję su siuntų pristatymo ir logistikos įmonės veikla. Remiantis laboratorinio darbo užduotimi, naudojant „Hadoop“ ir „Spark“ sistemas ir pasitelkiant „MapReduce“ metodą, duomenys yra išgryninami ir atitinkamai suagreguojami.

„Hadoop" yra plačiai paplitusi atviro kodo „MapReduce“ realizacija, leidžianti paskirstytai apdoroti didelius duomenų rinkinius naudojant paprastus programavimo modelius. Ji palengvina programuotojui klasikines paskirstytų skaičiavimų problemas. Ji leidžia specialistams apsisaugoti nuo potenčialių gedimų, sumažina reikalingą parašyto bei prižiūrimo kodo kiekį.

„Apache Spark“ (toliau „Spark“) yra sukurta kiek vėliau už „Hadoop“ ir leidžia atlikti skaičiavimus dar greičiau. Skirtingai nuo „Hadoop“, kuris tarpinius duomenis saugo failų sistemoje (HDFS), „Spark“ saugo duomenis operatyvioje atmintyje. Tai lemia sistemos greitesnį veikimą, kadangi nuskaitymas iš failų sistemos tokia dažna operacija. Tai leidžia pasiekti geresnį duomenų apdorojimo greitį. Ši sistema suteikia naudingų pagalbinių paketų duomenų analizei. Pagrindinis „Spark“ trūkumas yra didelis operatyvios atminties poreikis. „Hadoop“ ir „Spark“ suteikia galimybę atlikti sudėtingus ir ilgai užtrunkančius skaičiavimus be būtinybės specialistams patiems kurti sudėtingas sistemas, kurios gebėtų apdoroti didelius duomenų kiekius efektyviai.

# Pirmojo uždavinio sprendimas

## Pirmas laboratorinis darbas

### „Map“ funkcionalumo programavimas

Šiame laboratoriniame darbe buvo atlikta duomenų apie siuntų pristatymą bei logistikos įmonės veiklą analizė. Naudojantis „MapReduce“ programavimo modeliu bei „Hadoop“ platforma, buvo atlikta duomenų apdorojimo užduotis. Jos tikslas buvo pasiversti duomenis į tinkamą apdorojimui formatą bei atrinktus duomenis apjungti į lentelę pagal lauką „maršrutas“. Atlikus šią užduotį buvo apjungti duomenys „svoris“ (pritaikyta sumos operacija), „siuntu skaicius“ (pritaikyta sumos operacija) bei „geografine zona“ (nustatyti unikalių reikšmių dažniai).

Map funkcionalumo įgyvendinimui, buvo sukurtas map.py failas, kuris geba išrinkti reikalingus duomenis tolimesniam rūšiavimui ir reduce operacijai. Šis failas turi pagrindini ciklą, kuris iteruoja per visas pilnas duomenų failo eilutes, panaikina nereikalingus tarpus, paima eilutės duomenis be pradinių ir galinių duomenų atskirimo ženklų ir suskirsto esamą eilutę į atskirus duomenų objektų įrašus.

for line in sys.stdin:

    line = line.strip()

    line = line[2:len(line)-2]

    stripped\_string = line.split('}}{{')

Tada esamame cikle yra kuriamas dar vienas ciklas ir iteruojama per išskirstytus pilnus duomenų objektus. Visi duomenų įrašai yra atskiriami per skiriamuosius ženklus, paruošiami kintamieji, kurie turės būt ištraukti iš duomenų įrašo.

for data\_entry in stripped\_string:

        parsed\_strings = data\_entry.split('}{')

        map\_key = None

        map\_value\_1 = None

        map\_value\_2 = None

        map\_value\_3 = None

Sekantis ciklas yra skirtas išrinkti pačias raktų ir verčių poras, jei jos egzistuoja objekte. Kai baigiama iteruoti per visas vieno įrašo reikšmes, dabartinis ciklas pasibaigia ir duomenys yra rašomi į standartinę išvestį (failą), jei visos vertės egzistuoja.

        for parsed\_string in parsed\_strings:

            (key, value) = parsed\_string.split('=')

            # Skip iteration if value is empty

            if value == '':

                continue

            if key == main\_field:

                map\_key = int(value)

            if key == second\_field:

                map\_value\_1 = float(value)

            if key == third\_field:

                map\_value\_2 = int(value)

            if key == fourth\_field:

                map\_value\_3 = value

        if(map\_key != None and map\_value\_1 != None and map\_value\_2 != None and map\_value\_3 != None):

            print(f'{map\_key}\t{map\_value\_1}\t{map\_value\_2}\t{map\_value\_3}')

### „Reduce“ operacijos implementacija

Sekanti programa yra skirta duomenų rūšiavimui. Ji skaito standartinę išvestį (nustatytas failas) bei deda įrašus į atmintyje esantį masyvą. Tada masyvas yra išrūšiuojamas pagal raktą bei visi įrašai yra surašomi į standartinę išvestį (failą).

Kol raktas yra nepasikeitęs, taip pat yra sekamas geografinių zonų dažnis tam tikram maršrutui (raktui). Tai veikia dedant zonas kaip raktus į dictionary ir jų vertes laikant pasikartojimų skaičiumi. Kai raktas pasikeičia, tada yra išvedamos reikšmės, kurios yra: maršrutas, svorių suma, siuntų skaičiaus suma bei geografinės zonos su jų pasikartojimo dažniais.

for line in sys.stdin:

    line = line.strip()

    # Parse the input we got from mapper

    key, val\_1, val\_2, zone = line.split('\t', 3)

    # convert count (currently a string) to int

    try:

        val\_1 = float(val\_1)

        val\_2 = int(val\_2)

    except ValueError:

        # count was not a number, so silently

        # ignore/discard this line

        continue

    # For this key, put the zone count into the dictionary

    if zone != None and zone in zone\_dict:

        zone\_dict[zone] += 1

    else:

        zone\_dict.setdefault(zone, 1)

    # this IF-switch only works because Hadoop sorts map output

    # by key (here: word) before it is passed to the reducer

    if current\_key == key:

        current\_weight\_sum += val\_1

        current\_parcel\_sum += val\_2

    else:

        if current\_key != None:

            # write result to STDOUT

            print(

                f'{current\_key}\t{current\_weight\_sum}\t{current\_parcel\_sum}', end='')

            # Write zones for this key

            [print(f'\t{zone}\t{zone\_count}', end='')

             for [zone, zone\_count] in zone\_dict.items()]

            print()

        current\_weight\_sum = val\_1

        current\_parcel\_sum = val\_2

        current\_key = key

        zone\_dict = {}

# do not forget to output the last word if needed!

if current\_key == key:

    if zone in zone\_dict:

        zone\_dict[zone] += 1

    else:

        zone\_dict.setdefault(zone, 1)

    print(f'{current\_key}\t{current\_weight\_sum}\t{current\_parcel\_sum}', end='')

    # Write zones for this key

    [print(f'\t{zone}\t{zone\_count}', end='')

        for [zone, zone\_count] in zone\_dict.items()]

A screenshot of a computer screen

Description automatically generated with low confidenceAtlikus „Reduce“ operacija yra gaunamas failas, kuriame yra apjungti duomenys pagal nustatytus kriterijus.

1 pav. Rezultatas po „reduce“ operacijos

### „Combiner tipo užduoties įtraukimo galimybės

„MapReduce“ programavimo modelyje „Combiner“ yra funkcija, kuri veikia tarpines rakto ir reikšmės poras, kurias sukuria „map“ operacija, prieš siunčiant jas „reduce“ operacijai. Jos paskirtis - sumažinti duomenų, kuriuos reikia perduoti tinklu, kiekį ir pagerinti bendrą našumą. „Combiner“ atlieka tarpinių reikšmių agregavimą kiekvienam unikaliam raktui, sujungdamas ir apibendrindamas duomenis. Sujungdamas duomenis, „combiner“ gali padėti sumažinti duomenų apimtį, kurią turės apdoroti *reducer‘is*. Išvestis, gauta po „combine“ operacijos, siunčiama į *reducer‘į*, kur atliekamas tolesnis apdorojimas. Tokie duomenų apjungėjai yra ypač naudingi, kai tarpiniai duomenys yra dideli ir juos galima efektyviai sujungti nedarant įtakos galutiniam rezultatui.

Laboratorinio darbo užduoties atveju, būtų galima panaudoti šį programavimo modelį, kadangi jis leistų sumažinti reikšmių kiekį, kurį gaus „reduce“ operacija. *Combiner‘is* galėtų apjungti reikšmes pagal raktą ir iškarto taikyti sumos operaciją visoms reikšmės, sugrupuotoms pagal raktą. Ši operacija potencialiai galėtų paspartinti skaičiavimo procesą, kadangi skaičiavimai būtų atliekami su mažesniu duomenų skaičiumi bei bendras perduodamų reikšmių skaičius *reducer‘iui* sumažėtų.

## Antrasis laboratorinis darbas

Antrąjame laboratoriniame darbe toliau buvo analizuojamas duomenų rinkinys apie siuntų pristatymo bei logistikos įmonės veiklą. Šiame darbe buvo atliktas duomenų apdorojimas bei jų struktūrizavimas bei, naudojantis „DAG“ programavimo modeliu, atliktas jų agregavimas pagal raktus ir suskaičiuoti kiekybiniai parametrai: mažiausias, didžiausias ir vidutinis aritmetiniai siuntų svorio vidurkiai skirtingose svorio grupėse.

### „DAG“ programavimo modelio veikimas

Šiame darbe naudojama programinė įranga yra „Spark“. Tai sistema, kurios esmė yra yra atlikti „map“ ir „reduec“ operacijas, todėl ji yra labai panaši į „Hadoop“ platformą. Nors abi sistemos veikia paskirstytųjų skaičiavimų aplinkoje, „Spark“ išsiskiria tuo, kad duomenis apdoroja atmintyje, o tai gerokai padidina jos našumą. Atlikdama operacijas atmintyje, „Spark“ sumažina nuskaitymo ir įrašymo į diskaą operacijų poreikį, todėl greičiau vykdomos operacijos ir padidėja efektyvumas. Tačiau šis pranašumas lemia tai, kad ši sistema naudoja žymiai daugiau operatyviosios atminties, lyginant su „Hadoop“.

Vienas iš esminių „Spark“ ir „Hadoop“ skirtumų yra duomenų apdorojimo būdas. „Hadoop“ remiasi gerai žinomu „MapReduce“ modeliu, kai duomenys apdorojami dviem atskirais etapais: duomenų grupavimo pagal raktus (*map‘inimo*) ir „reduce“ operacijos. Tuo tarpu „Spark“ naudoja pranašesnį metodą - nuosavą paskirstytą vykdymo variklį, paremtą kryptiniu acikliniu grafu (DAG).

„DAG“ yra grafas, sudarytas iš viršūnių ir briaunų. Šiame grafe viršūnės atitinka RDD duomenų vienetus, o briaunos nurodo operacijos, kurios yra atliekamos su šiais duomenimis.

Viena iš priežasčių, kodėl buvo sukurtas „DAG“ , yra tai, kad „Hadoop“ atliekamos operacijos nepriklauso viena nuo kitos ir nėra žinoma, kurios operacijos bus atliekamos tolimesniuose žingsniuose. Dėl šios priežasties, tarpiniai rezultatai yra rašomi į diską, kai būtų galima to išvengti. Rašymo į diską metu yra prarandamas greitis bei išnaudojama „HDFS“ atmintis. „DAG“ spendžia šią problemą, kadangi jis leidžia operacijas vykdyti viena po kitos ir pasirūpina duomenų saugojimu, priklausomai nuo to, kokios bus atliekamos operacijos.

Be to, „Spark“ atlieka operacijas ne iš karto, tik tik tada, kai to prireikia. Tai vadinama „lazy evaluation“. Tai atlikti leidžia tas faktas, kad grafas išlaiko operacijų priklausomybes.

### Laboratorinio darbo eiga naudojantis „Spark“

Siekiant atlikti laboratorinį darbą, buvo sudiegta „Apache Spark“ lokalioje programavimo aplinkoje. Programos kode, „Spark“ konteksto inicializacija yra reikalinga, kad būtų galima atlikti veiksmus su RDD duomenų struktūromis. Naudojantis „pyspark“ programinės įrangos paketu failas yra nuskaitomas ir užkraunamas į atmintį.

from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName('MyApp')

sc = SparkContext(conf=conf)

A = sc.textFile('duom\_full.txt')

Toliau buvo sukurtas programinis kodas, kuris pateiktus duomenis pavečia į apdorojamą formatą. Tai buvo atlikta sukuriant papildomą funcija, kuri, pagal duomenų failo struktūrą, atrenka reikalingus duomenis. Ši funkcija išrenka iš duomenų objektų raktą, pagal kurį bus agreguojami duomenys. Rakto vertės yra duomenų faile pateiktos svorių grupės. Funkcija, toliau skaitydama duomenų failą, išrenka svorio vertes ir, pasibaigus nuskaitymo ciklui, grąžina rakto ir vertės porą.

Apsibrėžus pagrindinę map funckija, buvo panaudotos siauros transformacijos, tam kad duomenys būtų apdoroti ir paruošti tolimesnėms operacijoms su jais (operacijai „reduce“). Panaudotos siauros duomenų tranformacijos buvo „map“, „flatMap“ bei „filter“. Jos yra atliekamos vienoje particijoje, kas reiškia, kad duomenys nėra perkeliami tarp skirtingų particijų.

# map

def map\_func(x):

    key = ''

    value = float(0)

    for entry in x:

        e\_key, e\_value = entry.split('=')

        if e\_key == 'svorio grupe':

            key = e\_value.replace('}}', '').replace('{{', '')

        elif e\_key == 'svoris':

            try:

                value = float(e\_value.replace('}}', '').replace('{{', ''))

            except Exception as e:

                pass

    return (key, value)

y = A.map(lambda x: x.strip())\

    .flatMap(lambda x: x.split('}}{{'))\

    .map(lambda x: x[2:] if x.startswith('{{') else x)\

    .map(lambda x: x[:-2] if x.startswith('}}') else x)\

    .map(lambda x: x.split('}{'))\

    .map(map\_func)\

    .filter(lambda x: x[0] != '')

print(y.take(20))

Baigus rašyti „map“ funkcionalumo programinį kodą, buvo sukurtas programos fragmentas, kuris yra atsakingas už „reduce“ operaciją. Tam, kad suagreguoti duomenis ir paruošti juos vidurkių apskaičiavimui, buvo aprašyta funkcija „reduce\_func“, kuri grąžina sudėtas „value“ tipo reikšmių porą. Su paruoštais duomenimis po „map“ operacijos, funkcijai paduodami argumentai yra „x“ ir „y“. Tai yra akumuliatorius bei sekanti iteruojama reikšmė. Tai leidžia atlikti tam tikras operacijas su gautais duomenimis. Šiuo atveju, akumuliatoriaus sukaupta svorio reikšmė yra sudedama su esamos iteracijos svorio reikšme, ir gražinama kaip pirmas poros elementas. Antrasis poros elementas yra to svorio pasikartojimo dažnis, kuriam būna priskirta standartinė reikšmė – 1. Šis skaičius leidžia sekti apdorotų elementų skaičių, nes norint apskaičiuoti vidurkius, yra būtina gauti visą svorių sumą bei visų elementų skaičių.

Siaura transformacija „mapValues“ yra iškviečiama ant „Spark“ RDD duomenų rinkinio ir paruošia prieš tai įvardintą papildomą „value“ kintamąjį, kuris bus naudojamas sekti elementų skaičių. Ši transformacija yra siaura ir nereikalauja duomenų perkėlimo tarp particijų. Sekanti kviečiama transformacija yra „reduceByKey“. Ji yra plataus tipo transformacija ir apdoroja duomenis, kurie gali būti skirtingose particijose. Tai lemia, kad duomenys gali būti perkeliami tarp skirtingų particijų, siekiant įgyvendinti šią transformaciją. „reduceByKey“ apjungia kiekvieno rakto vertes pagal nurodytą funkciją. T.y., pagal svorio grupes apjungia svorius bei visą elementų kiekį. Po šios transformacijos, yra atlikta „reduce“ operacija ir duomenys yra paruošti vidurkių apskaičiavimui.

def reduce\_func(x, y):

    return (x[0] + y[0], x[1] + y[1])

reduced = y.mapValues(lambda x:  (x, 1))\

    .reduceByKey(reduce\_func)

print(reduced.take(20))

Siekiant paskaičiuoti statistiką, buvo įvykdytos siaurosios „mapValues“ transformacijos, kurios, kiekvienam raktui, apskaičiuoja vidurkius, minimalias ir maksimalias reikšmes pagal gautas vertes kiekvienai svorių grupei.

# Statistics

avg = reduced.mapValues(lambda x: x[0] / x[1])

print(avg.collect())

min = reduced.mapValues(lambda x: x[0] if x[0] < x[1] else x[1])

print(min.collect())

max = reduced.mapValues(lambda x: x[0] if x[0] > x[1] else x[1])

print(max.collect())

Įvykdžius šias transformacijas buvo gautas rezultatas, kurio pirmasis elementas yra svorio grupė ir mažiausia šios grupės vertė, svorio grupė ir didžiausia vertė bei svorio grupė ir aritmetinis vidurkis.

[('>300', 516.7904761904762), ('<300', 100.26420454545455), ('<50', 10.383368421052634)]

[('>300', 21), ('<300', 176), ('<50', 285)]

[('>300', 10852.599999999999), ('<300', 17646.5), ('<50', 2959.2600000000007)]

## Trečiasis laboratorinis darbas

Trečiajame laboratoriniame darbe toliau buvo dirbama su duomenimis, aprašančiais siuntų pristatymo bei logistikos įmonės veiklą. Šioje užduotyje buvo pateikti papildomi duomenys apie maršrutus skirtingomis savaitės dienomis. Atliekant laboratorinį darbą, buvo ištirta tiesinė priklausomybė parametro „BendrasLaikas“ nuo parametro „svoris“. Duomenys buvo agreguojami pagal maršrutą ir datą bei buvo pritaikoma sumos operacija. Kiekvieni duomenys buvo analizuoti kiekvienam mašinos tipui atskirai.

### Laboratorinio darbo eiga naudojantis „Spark“

Šiam darbui atlikti buvo panaudoti keli programiniai paketai. Pagrindinis jų – „pyspark“. Šis paketas leidžia atlikti duomenų transformacijas greitai ir efektyviai rašant kodą „Python“ programavimo kalba.

Detalizuojant naudotus įrankius, iš „Spark“ paketo, buvo panaudotas „Spark SQL“ modulis, kuris yra skirtas apdoroti struktūrizuotus duomenis. Jis suteikia programavimo abstrakciją „DataFrame“. Ši duomenų struktūra leidžia laikyti lentelės tipo duomenis, sudarytus iš eilučių ir stulpelių. „DataFrame“ objektais galima lengvai manipuliuoti, ištrinti ar pridėti stulpelius, eilutes ir kitas panašias transformacijas.

# Literatūra