УДК 004.852

Ловчинський Сергій Броніславович
студент
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Ловчинский Сергей Брониславович
студент
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Lovchinsky S.
student
National Technical University of Ukraine
«Igor Sikorsky Kyiv Polytechnic Institute»

ОБРОБКА ТА АНАЛІЗ ДАНИХ З ВИКОРИСТАННЯМ APACHE SPARK

ОБРАБОТКА И АНАЛИЗ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ APACHE SPARK

DATA PROCESSING AND ANALYSIS USING APACHE SPARK

Анотація. *Аналіз і дослідження вирішення задачі розподіленого оброблення даних за допомогою Apache Spark.*

У цій статті представляється дослідження найсучасніших методів оброблення даних та виділено їх переваги і недоліки у контексті вимог до розподіленої системи. Показується підхід Apache Spark до вирішення задач розподіленого оброблення даних. Демонструється на тестових прикладах використання Apache Spark.

Ключові слова: *Розподілені системи, обробка даних, потокова обробка, аналіз даних, великі дані.*

Аннотация. *Анализ и исследования решения задачи распределенного обработки данных с помощью Apache Spark.*

В этой статье представляется исследование современных методов обработки данных и выделены их преимущества и недостатки в контексте требований к распределенной системе. Показывается подход Apache Spark к решению задач распределенной обработки данных. Демонстрируется на тестовых примерах использование Apache Spark.

Ключевые слова: *Распределенные системы, обработка данных, потоковая обработка, анализ данных, большие данные.*

Abstract. *Analysis and research of the solution of the problem of distributed data processing with the help of Apache Spark.*

This article presents an investigation of modern data processing methods and identifies their advantages and disadvantages in the context of requirements for a distributed system. Apache Spark's approach to solving distributed data processing problems is shown. It is demonstrated on test cases using Apache Spark.

Keywords: *Distributed systems, data processing, stream processing, data analysis, large data.*

Постановка проблеми. Аналіз сучасних тенденцій розвитку інформаційних систем свідчить про ускладнення використовуваних в них структур даних при одночасному збільшенні оброблюваної інформації, а також кількості інформації, що зберігається. Для обробки дедалі більшого обсягу інформації потрібно нове апаратне і відповідне програмне забезпечення, оскільки існуюче вже не справляється з поставленими завданнями. Таким чином, з'явилася ідея об'єднання обчислювальних потужностей двох і більше комп'ютерів для вирішення складних, непосильних для кожного з них окремо завдань. Поява локальних мереж призвело до розвитку нової галузі розробки програмного забезпечення, а саме створення розподілених систем. [1]

Зі збільшенням обсягу оброблюваної інформації з'явилася необхідність розподіленої обробки даних з використанням сучасних систем управління базами даних. Даний напрямок в даний час дослідження в області розподіленої обробки полягають в створенні

і обслуговуванні розподілених баз даних і проектуванні додатків, що дозволяють організувати розподілені обчислення. Основний напрямок використання засобів обчислювальної техніки — зберігання і обробка великих обсягів інформації.

Цей напрямок широко використовується в сучасних інформаційних системах. У найширшому сенсі інформаційна система являє собою програмний комплекс, функції якого полягають у підтримці надійного зберігання інформації в пам'яті комп'ютера, виконанні специфічних для даного застосування перетворень інформації, наданні користувачам зручного і легкого інтерфейсу.

У цій статті визначаються вимоги до побудови розподілених систем для оброблення даних. Вивчаються й описуються архітектурні рішення в галузі обробки великих обсягів даних та підходи до розробки програмного забезпечення, які дозволяють реалізувати платформу таким чином, аби вона задовольняла означені потреби.

Метою статті є аналіз і дослідження вирішення задачі розподіленого оброблення даних за допомогою Apache Spark. Для реалізації цього завдання поставлені наступні задачі:

1. Дослідити і вивчити основні принципи роботи, виявити ефективні способи і методи розподіленого оброблення даних в прикладних системах.

2. Проаналізувати існуючі рішення та підходи в даній предметній області.

3. Розглянути підходи до побудови систем оброблення даних з використанням Apache Spark.

4. Розробити актуальні тестові приклади, які демонструють використання Apache Spark.

Актуальність задачі. За останні роки в високопродуктивних обчислювальних системах відбулися значні зміни, пов'язані зі зростанням потреби в обчислювальних ресурсах в багатьох прикладних задачах обробки та аналізу даних Сучасні інформаційні системи породжують постійно зростаюче надходження даних у великих об'ємах. На сьогоднішній день навіть невеликі бізнес-системи генерують величезну кількість даних, не кажучи вже про великі і складні системи з таких сфер як медицина, аналіз і моделювання фізичних процесів, криптографія, і т.д. Як наслідок, масштаби даних і складність сучасних алгоритмів аналізу даних і машинного навчання ростуть незрівнянно швидше, ніж обчислювальні потужності комп'ютерів. В зв'язку з цим неминуче з'являються та набувають широкого поширення інструменти для розподіленого аналізу даних. Одним із засобів для побудови таких систем і є Apache Spark.

Виклад основного матеріалу дослідження. Ключовими вимогами до системи розподіленого аналізу даних являються: висока продуктивність, прозорість, відкритість, масштабованість, відмовостійкість. Розглянемо кожну вимогу детальніше.

Комп'ютерна продуктивність це кількісна характеристика швидкості виконання операцій з оброблення та збереження даних на комп'ютері. Розподілені системи програмного забезпечення, які мають властивість продуктивності характеризуються такими основними показниками:

- Споживання ресурсів центрального процесора;
- Споживання оперативної пам'яті;
- Робота з файловою системою;
- Час виконання операції.

Прозорість системи припускає приховування деталей реалізації, структури, факту розподілення ресурсів та процесів між багатьма апаратними та програмними ресурсами, а також деталей координації їх роботи. Тобто розподілена система, яка задовольняє вимогу прозорості працює у вигляді єдиної централізованої інформаційної системи.

Відкрита система — це система, яка реалізує відкриті специфікації на інтерфейси, сервіси (послуги середовища) і підтримувані формати даних, достатні для того, щоб дати можливість належним чином розробленому прикладному програмному забезпечити переносимість в широкому діапазоні систем з мінімальними змінами, взаємодіяти з іншими застосуваннями на локальних і віддалених системах, і взаємодіяти з користувачами в стилі, який полегшує перехід користувачів від системи до системи. [2]

Система називається масштабованою, якщо вона здатна збільшувати продуктивність пропорційно додатковим ресурсам. Масштабованість можна оцінити через ставлення приросту продуктивності системи до приросту використовуваних ресурсів. Чим ближче цей показник до одиниці, тим краще. Також під масштабністю розуміється можливість нарощування додаткових ресурсів без структурних змін центрального вузла системи. [3]

Розглянемо підхід Apache Spark до вирішення задач розподіленого оброблення даних. Apache Spark адресована фахівцям в області аналізу даних або дослідникам і інженерам-програмістам. Саме вони зможуть отримати найбільшу вигоду від залучення фреймворка Spark для вирішення своїх завдань. Багата колекція бібліотек таких як MLlib, GraphX, що входять до складу Spark, допоможе фахівцям в області аналізу даних вирішувати статистичні завдання, непосильні єдиному комп'ютеру. Інженери-програмісти, в свою чергу, зможуть писати розподілені програми на основі Spark і управляти промисловими додатками. Програмісти і дослідники по-різному зможуть задіяти Spark для вирішення великих розподілених завдань в своїх областях. [4]

Apache Spark — це універсальна і високопродуктивна кластерна обчислювальна платформа. [5] Фреймворк створювався з метою охопити якомога ширший діапазон робочих навантажень, які перш вимагали створення окремих розподілених систем, включаючи додатки пакетної обробки, циклічні алгоритми, інтерактивні запити і потокову обробку.

Обробка та аналіз даних вимагає гнучкості, швидкої реакції і постійних інновацій. Apache Spark дозволяє фахівцям з обробки та аналізу даних і розробникам прикладних програм працювати практично з будь-якими типами даних із будь-яких джерел. Завдяки великій кількості компонентів Spark підтримує велике різноманіття видів аналізу даних. Володіючи універсальністю для роботи в багатьох середовищах, Apache Spark надзвичайно зручний для створення алгоритмів, що дозволяють витягати корисну інформацію із складних типів даних. [6]

Нижче проводиться аналіз отриманих результатів, а саме поведінка фреймворку Apache Spark при вирішені різних задач. Демонструються описані типи задач які буде розв'язано засобами Apache Spark, безпосередньо алгоритми розв'язання, а також виведені результати роботи фреймворку.

Підрахунок кількості слів в тексті. У першому прикладі буде вирішено просту задачу підрахунку кількості заданих слів у файлі, використовуючи кілька перетворень для створення набору даних (String, Int) так званих пар підрахунків, після чого результати роботи програми будуть збережені у файл. На початку потрібно створити екземпляр JavaRDD, який буде містити зчитаний текст з файлу.

```
JavaRDD<String> textFile = sc.text-
File("hdfs://…");
```

Після цього розділити вхідний текст на масив строк зі слів і помістити результат в новий JavaRDD.

```
JavaRDD words = lines.flatMap(s ->
    Arrays.asList(SPACE.split(s)).it-
    erator());
```

Створити набір даних (String, Int) пар підрахунків з отриманих строк.

```
JavaPairRDD<String, Integer> ones =
    words.mapToPair(s -> new Tuple2<>
    (s, 1));
```

Далі викликається метод для підрахунку кількості слів.

```
JavaPairRDD<String, Integer> counts =
    ones.reduceByKey((i1, i2) -> i1 + i2);
```

Результат роботи тестової програми зберігається на диску за допомогою наступного метода.

```
counts.saveAsTextFile("hdfs://…");
```

Вище описана тестова програма також демонструє простоту у створенні прикладних програм використовуючи високорівневий API, який дозволяє сконцентруватися на предметній стороні задачі, а не на її реалізації.

Результати випробувань тестової програми підрахунку слів представлено в порівняльній таблиці 3.1

Таблиця 3.1

Підрахунок кількості заданого слова

Розмір файлу	100 Мб	1 Гб	5 Гб
Час виконання	5 с	31 с	72 с

За даними таблиці наведено графічні характеристики параметрів на рисунку 3.1

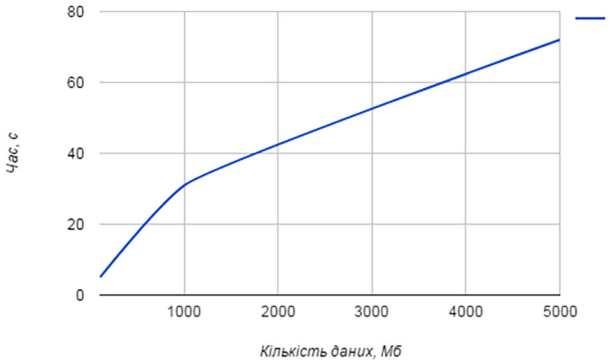

Рисунок 3.1. Відношення часу виконання від кількості даних

Класифікація спама. Існують різні типи завдань машинного навчання, включаючи класифікацію, регресію або кластеризацію, що мають різні цілі. Як простий приклад розглянемо задачу класифікації, метою якої є визначення приналежності елемента до тієї чи іншої категорії, а саме електронного листа на наявність в ньому спаму, на основі маркованих прикладів подібних елементів, електронних листів, про яких точно відомо, містять вони в собі спам чи ні. Всі алгоритми машинного навчання вимагають визначити для кожного елемента набір характеристичних ознак, який буде передаватися функції навчання, в конкретному прикладі, для електронного листа характеристичними ознаками може бути число згадок спам слів. У багатьох випадках визначення правильних характеристичних ознак є найскладнішою частиною машинного навчання. Більшість алгоритмів підтримують тільки числові характерні ознаки, тому потрібно використовувати вектори чисел, що представляють значення ознак, як наслідок, часто важливим кроком є витягнення і перетворення ознак для отримання таких векторів. [7] На рисунку 3.2 показано приклад роботи процесу машинного навчання.

Для реалізації даної програми буде використано наступні алгоритми із бібліотеки MLlib:

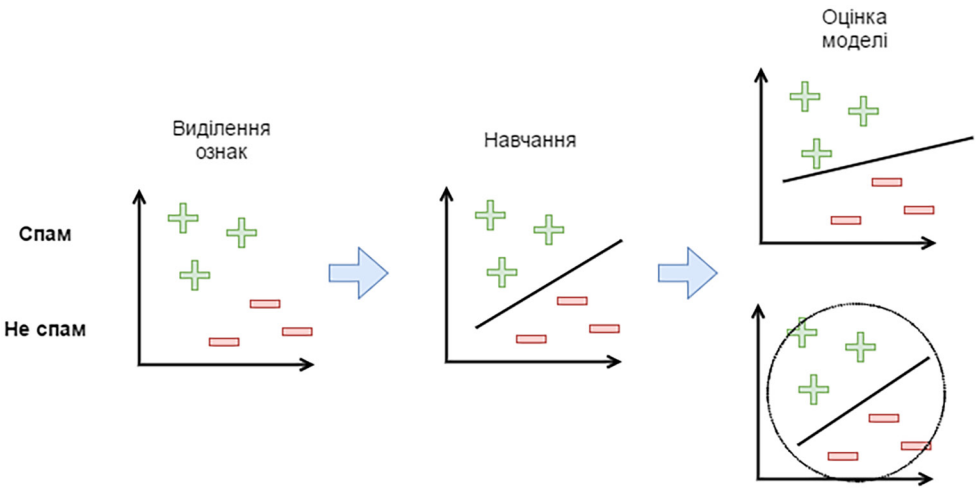

Рисунок 3.2. Етапи процесу машинного навчання

- HashingTF, який створює вектор частот входження обраних термінів (term frequency) в тексті.
- LogisticRegressionWithSGD, який реалізує процедуру логістичної регресії (logistic regression) методом стохастичного градієнтного спуску.

Передбачається, що існують два файли, spam.txt і nonspam.txt, кожен з яких містить приклади електронних листів зі спамом і без спаму, по одному в рядку. Кожен електронний лист в кожному файлі перетворюється в вектор ознак з частотами термінів, і проводиться навчання моделі логістичної регресії для поділу повідомлень двох типів.

Логістична регресія — це метод двійкової класифікації, що ідентифікує лінійну площину, яка розділяє позитивні і негативні зразки. У MLlib реалізація логістичної регресії приймає об'єкти LabeledPoint зі значеннями маркерів 0 або 1 і повертає об'єкт моделі LogisticRegressionModel, здатний передбачати значення для нових точок.

Для початку, потрібно створити екземпляр HashingTF для відображення тексту електронних листів в вектори з 1000 ознак.

```
final HashingTF tf = new HashingTF(1000);
```

Створити набори даних LabeledPoint для прикладів, що дають позитивну реакцію (спам) і негативну (без спаму).

```
JavaRDD positiveExamples = spam.map
(new Function()
    public LabeledPoint call(String
    email) {
        return new LabeledPoint(1,
        tf.transform(Arrays.asList
        (email.split(" ")
    }
});
JavaRDD negativeExamples = nonspam.
map(new Function()
    public LabeledPoint call(String
    email) {
        return new LabeledPoint(0,
        tf.transform(Arrays.asList
        (email.split(" ")
    }
});
JavaRDD trainData =
    positiveExamples.union(negativeEx-
    amples);
```

Виконати логістичну регресію методом стохастичного градієнтного спуску.

```
LogisticRegressionModel model =
new LogisticRegressionWithSGD().
run(trainData.rdd());
```

Після створення моделі за допомогою метода predict() повертається фактично розрахована оцінка для вхідних даних.

```
model.predict(testData);
```

Результати випробувань тестової програми класифікації спаму представлено в порівняльній таблиці 3.2

Таблиця 3.2

Класифікація спаму

Кількість листів	10000	100000	500000	1000000
Час виконання	26 с	49 с	189 с	297 с

За даними таблиці наведено графічні характеристики параметрів на рисунку 3.3.

Висновки та пропозиції. В даній статті було виділено основні вимоги, що висуваються до систем розподіленого оброблення даних; визначено основні підходи до вирішення задач обробки великих за обсягами даних; досліджено існуючі платформи та виокремлено їх особливості.

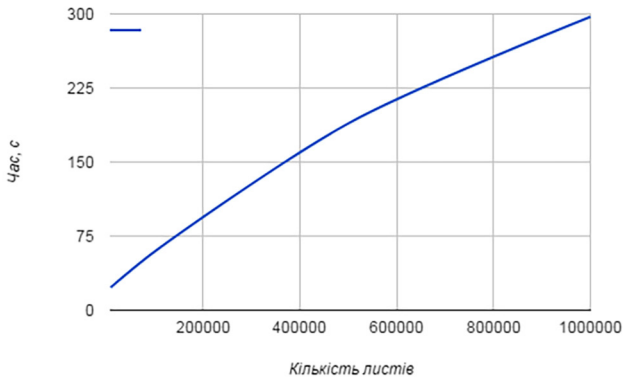

Рисунок 3.3. Відношення часу виконання від кількості листів

Проводиться аналіз отриманих результатів, а саме було розроблено програмний продукт, за допомогою якого можна дослідити роботу системи оброки даних Apache Spark на декількох тестових прикладах. Результати роботи даних тестових прикладів були представлені в роботі. Виявилося, що фреймворк у цілому справляється з поставленою задачею. Також, є алгоритми де Apache Spark у зв'язку із специфічним алгоритмом роботи платформи на рівень ефективніше виконує програму.

Використання Apache Spark для побудови розподілених системи обробки великих даних дозволяє забезпечити автоматичне розпаралелювання і зберігання даних на внутрішніх дисках вузлів кластера. Запропонована архітектура приховує від прикладного програміста деталі внутрішнього устрою Apache Spark і надає простий програмний інтерфейс, який дозволяє сконцентруватися на предметній стороні задачі, а не на її реалізації. Apache Spark дозволяє фахівцям з обробки та аналізу даних і розробникам прикладних програм працювати практично з будь-якими типами даних із будь-яких джерел.

Література

1. Э. Таненбаум. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Ван. Стеен. — СПб.: Питер, 2003. — С. 807.

2. Карпов Л. Е. Архитектура распределенных систем программного обеспечения / Карпов Л. Е. — М.: МАКС Пресс, МГУ, ВМК, 2007. — С. 132.

3. О. А. Литвинов. Розподілена обробка інформації / О. А. Литвинов, В. С. Хандецький — Д.: ТОВ «Баланс-Клуб», 2013. — С. 314.

4. James A. Scott. Getting Started with Apache Spark / James A. Scott. — USA: MapR technologies Inc, 2015. — С. 88.

5. .H. Karau. Learning Spark: Lightning-Fast Big Data Analysis / H. Karau, A. Konwinski, P. Wendell, M. Zaharia. — USA: O'Reilly Media Inc, 2015. — С. 257.

6. Jacek Laskowski. Mastering Apache Spark. — Режим доступа: https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details. — Дата доступа: 25.05.2017.

7. Офіційна документація Apache Spark. — Режим доступа: https://spark.apache.org/. — Дата доступа: 25.05.2017.