МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №\_\_1\_\_**

по дисциплине«Технология обработки больших данных»

Тема: «Обработка и анализ больших данных на примере статистики сообщений пользователей»

ИСПОЛНИТЕЛЬ: \_\_Журавлев Н. В.\_\_

ФИО

группа ИУ5-44М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"12"\_\_марта\_\_\_2025 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_Сухобоков А.В.\_\_\_\_\_

ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"12"\_марта\_\_\_2025 г.

Москва - 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Описание задачи и алгоритма

Этап 1: Подготовка и изучение основных понятий

- Изучите ключевые понятия:

- Что такое «большие данные» (Big Data)

- Концепция MapReduce

- Обработка потоков данных

- Основные проблемы и ограничения при работе с большими данными

Этап 2: Получение и генерация тестового набора данных

Генерируйте большой CSV-файл или набор текстовых файлов, содержащих сообщения пользователей (например, 1ГБ данных или несколько миллионов строк). Пример структуры файла (CSV):

| user\_id | timestamp | message\_length | message |

|---------|-----------|----------------|---------|

| 12345 | 1705227300 | 46 | Какой-то текст сообщения пользователя |

Студенты самостоятельно создают скрипт на Python (или другом языке), для генерации данных:

- Генерируйте данные случайно

- Используйте разнообразные user\_id, timestamp (в диапазоне нескольких месяцев), тексты сообщений и длину сообщения

Этап 3: Разработка алгоритма обработки больших данных

Необходимо реализовать (по выбору преподавателя или студента) одну из следующих задач:

(A) Анализ наиболее активных пользователей:

- Посчитать количество сообщений от каждого `user\_id`.

- Найти топ-10 самых активных пользователей за заданный интервал времени.

(B) Статистика средней длины сообщения по дням:

- Определить среднюю длину сообщений по дням недели. Найти, в какой день недели сообщения в среднем длиннее.

(C) Поиск популярных слов в сообщениях:

- Реализуйте подсчёт частоты каждого отдельного слова.

- Найдите 20 самых популярных слов (англ.) в наборе сообщений (игнорируя стоп-слова).

Основное требование: Данные должны читаться построчно или блоками, а результаты — накапливаться постепенно. Используйте подходы, избегая загрузки всех данных одновременно в оперативную память.

Этап 4: Реализация алгоритма

- Напишите программу, реализующую выбранную задачу.

- Протестируйте программу сначала на небольших данных, потом на полном наборе больших данных.

- Проверьте эффективность по времени и расходу памяти.

Этап 5: Анализ результатов и составление отчета

Отчёт должен включать:

- Описание задачи и алгоритма

- Реализация алгоритма (фрагменты кода с комментариями)

- Полученные результаты и выводы

- Анализ времени работы и затрат памяти программы

- Сравнение производительности при увеличении объёма данных

# Реализация алгоритма

Необходимо реализовать подсчёт частоты появления каждого отдельного слова. Затем найти 20 самых популярных слов в наборе сообщений (игнорируя стоп-слова). Реализация представлена на листинг 1.

Листинг 1. Алгоритм подсчёта частоты появления каждого слова

|  |
| --- |
| import csv  from multiprocessing import Pool, cpu\_count  from collections import defaultdict  import time  from memory\_profiler import memory\_usage  from statistics import mean  STOP\_WORD = ["a", "no", "of", "A", "in", "at", "on", "or", "it", "me", "he", "be", "to", "her", "us", "if", "for", "so"]  FILE\_NAME = "data.csv"  # Функция Map каждый процесс обрабатывает часть своих данных и возвращает промежуточные результаты  def map\_function(chunk):  local\_result = defaultdict(list)  for row in chunk:  msg = row[3].replace(".", "")  # Получить все слова  all\_words = msg.split()  # Занести все полученные слова в результат  for w in all\_words:  local\_result[w].append(msg.count(w))  return local\_result  # Функция Reduce объединяет промежуточные результаты  def reduce\_function(results):  res = defaultdict(list)  for local\_result in results:  for word, count\_repeat in local\_result.items():  res[word].extend(count\_repeat)  return res  # Чтение файла и разделение на чанки  def read\_chunks(file\_path, chunk\_size):  with open(file\_path, mode='r', encoding='utf-8') as file:  reader = csv.reader(file)  chunk = []  for row in reader:  chunk.append(row)  if len(chunk) == chunk\_size:  yield chunk  chunk = []  if chunk: # Обрабатываем оставшиеся строки  yield chunk  # Основная функция  def main(file\_path, chunk\_size=10000):  # Начало измерения времени  start\_time = time.time()  # Создаем процессы  pool = Pool(processes=cpu\_count())  # Чтение файла по чанкам и параллельное выполнение в pool.imap  chunks = read\_chunks(file\_path, chunk\_size)  map\_results = pool.imap(map\_function, chunks)  # Выполнение Reduce  result\_reduce = reduce\_function(map\_results)  # Выполняем выборку максимального значения среди всех значений  sorted\_repeated\_word = {word: max(count\_repeat) for word, count\_repeat in result\_reduce.items()}  # Сортируем дни по убыванию  sorted\_days = sorted(sorted\_repeated\_word.items(), key=lambda item: item[1], reverse=True)  # Вывод результата  print("Слова и их повторами:")  max\_word = ""  max\_count = 0  i = 0  for word, count\_repeat in sorted\_days:  if word not in STOP\_WORD:  i += 1  print(f"{word}: Повторов {count\_repeat:.2f}")  if i == 20:  break  if i == 1:  max\_word = word  max\_count = count\_repeat  print(f"\nСамое часто употребляемое слово: {max\_word} ({max\_count:.2f} раз)")  end\_time = time.time()  print(f"\nВремя выполнения программы: {end\_time - start\_time:.2f} секунд")  # Запуск программы с измерением памяти  if \_\_name\_\_ == '\_\_main\_\_':  mem\_usage = memory\_usage((main, (FILE\_NAME, )), interval=0.1, timeout=1)  print(f"Пиковое использование памяти: {max(mem\_usage):.2f} MiB")  print(f"Среднее использование памяти: {mean(mem\_usage):.2f} MiB") |

# Полученные результаты

В результате выполнения алгоритма на 3 видах сгенерированных файлов получились следующие результаты.

Таблица 1. Результат выполнения алгоритма

|  |  |  |  |
| --- | --- | --- | --- |
| Размер | 109 МБ | 545 МБ | 1038 МБ |
| Время выполнения | 5.57 с | 31.41 с | 67.24 с |
| Пиковое использование памяти | 203.48 МБ | 818.57 МБ | 1519.38 МБ |
| Среднее использование памяти | 117.12 МБ | 445.31 МБ | 838.03МБ |
| Самое часто употребляемое слово | As (6 раз) | Off (5 раз) | As (6 раз) |

# Анализ времени работы и затрат памяти программы

На маленьком значении размера происходит более быстрое выполнение чем на средней и большой. Так же при маленьком размере используется меньшая по объёму память.

# Сравнение производительности

Минимальное время исполнения достигается при меньшем объёме файла. Максимальное время при максимальном значении, аналогично с объёмом используемой памяти.