🔷 Общая логика

Этот мастер-скрипт управляет распределением данных между воркерами, запрашивает их обработку и собирает результаты, чтобы построить гистограмму распределения.

🔶 Основные шаги:

1. **Разделение данных**
   - Данные (`csv_data`) разбиваются на равные части между воркерами.
2. **Отправка данных воркерам**
   - Каждый воркер получает свою часть данных с помощью команды `store_data_p3` .
3. **Запуск вычислений на воркерах**
   - Мастер отправляет команду `p3_map` , чтобы каждый воркер построил свою локальную гистограмму.
4. **Сбор и объединение гистограмм**
   - Мастер получает локальные гистограммы и складывает их в одну глобальную.
5. **Нормализация гистограммы**
   - Преобразование частот в вероятности (деление на общее число значений).
6. **Отрисовка гистограммы**
   - Гистограмма отображается с помощью `matplotlib` .

🔷 Разбор кода по шагам

1 Определение списка воркеров

python Копировать Редактировать

```
workers = [ ('127.0.0.1', 5000), ('127.0.0.1', 5001), ('127.0.0.1', 5002) ]
```

Здесь у нас 3 воркера, работающие на одном компьютере (`127.0.0.1` — localhost) на портах `5000` , `5001` и `5002` .

2 Функция отправки команд воркерам

python Копировать Редактировать

```
def send_command(address, command_dict): s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM) s.connect(address) message = json.dumps(command_dict) + "\n"
s.sendall(message.encode('utf-8'))
```

🔷 Создается **TCP-сокет** для соединения с воркером.
🔷 Отправляется JSON-команда (`command_dict`) в виде строки.
🔷 `\n` используется как разделитель сообщений.

Получение ответа от воркера

python Копировать Редактировать

```
data = "" while "\n" not in data: part = s.recv(4096).decode('utf-8') if not part: break
data += part s.close() return json.loads(data.strip())
```

🔷 Читаем ответ воркера **пакетами по 4096 байт** до символа `\n` .
🔷 Закрываем соединение.
🔷 Возвращаем распарсенный JSON-ответ.

3 Основная функция `p3_run()`

📌 3.1 Исходные данные

python Копировать Редактировать

```
csv_data = [1.5, 2.3, 2.9, 3.1, 4.7, 5.0, 5.5, 6.2, 7.8, 7.1, 3.5, 4.0]
```

🔷 Это массив чисел, которые будут распределены между воркерами.

📌 3.2 Разделение данных на части

python Копировать Редактировать

```
num_workers = len(workers) chunk_size = len(csv_data) // num_workers chunks = [] for i in
range(num_workers): if i == num_workers - 1: chunk = csv_data[i * chunk_size:] else: chunk
= csv_data[i * chunk_size:(i + 1) * chunk_size] chunks.append(chunk)
```

🔷 Определяем количество воркеров (`num_workers = 3`).
🔷 Делим массив `csv_data` на 3 части (`chunk_size = 12 // 3 = 4`).
🔷 Последний воркер получает остаток (на случай, если `len(csv_data)` не делится нацело).

После выполнения этого кода `chunks` содержит:

python Копировать Редактировать

```
[ [1.5, 2.3, 2.9, 3.1], # Первый воркер [4.7, 5.0, 5.5, 6.2], # Второй воркер [7.8, 7.1, 3.5, 4.0] # Третий воркер ]
```

📌 3.3 Отправка данных воркерам

python Копировать Редактировать

```
responses = [] for i, address in enumerate(workers): command = {"command": "store_data_p3", "data": chunks[i]} res = send_command(address, command) responses.append(res) print("Пункт 3. Данные распределены:", responses)
```

- 🔷 Отправляем каждому воркеру его часть данных.
- 🔷 Воркеры должны сохранить эти данные у себя.

4 Запуск расчета локальных гистограмм

python Копировать Редактировать

```
map_results = [] for address in workers: command = {"command": "p3_map"} res = send_command(address, command) map_results.append(res.get("map_result", {}))
```

- 🔷 Отправляем команду `"p3_map"`, чтобы каждый воркер построил свою гистограмму.
- 🔷 Сохраняем локальные гистограммы в `map_results`.

5 Объединение локальных гистограмм

📌 5.1 Создание пустой глобальной гистограммы

python Копировать Редактировать

```
bins = [1, 2, 3, 4, 5, 6, 7, 8] global_hist = {f"{bins[i]}-{bins[i+1]}": 0 for i in range(len(bins) - 1)} total_count = 0
```

- 🔷 Определяем **диапазоны значений (бины)**.
- 🔷 Создаем пустой словарь `global_hist` для суммирования всех локальных результатов.
- 🔷 Переменная `total_count` хранит общее количество обработанных значений.

📌 5.2 Суммирование данных из воркеров

python Копировать Редактировать

```
for local_hist in map_results: for bin_key, count in local_hist.items(): global_hist[bin_key] += count total_count += count
```

- 🔷 Каждая локальная гистограмма суммируется в `global_hist`.

📌 5.3 Нормализация (расчет вероятностей)

python Копировать Редактировать

```
prob_hist = {bin_key: (count / total_count if total_count > 0 else 0) for bin_key, count in global_hist.items()}
```

- 🔷 Каждое значение переводится в вероятность:

$$P(x) = \frac{\text{число попаданий в бин}}{\text{общее количество значений}}$$

6 Визуализация результатов

📌 6.1 Построение гистограммы вероятностей

python Копировать Редактировать

```
def plot_histogram(prob_hist): bins = list(prob_hist.keys()) probabilities = list(prob_hist.values()) plt.figure(figsize=(8, 5)) plt.bar(bins, probabilities, color='blue', alpha=0.7, edgecolor='black') plt.xlabel("Интервалы значений X") plt.ylabel("Вероятность") plt.title("Гистограмма вероятностей") plt.grid(axis='y', linestyle='--', alpha=0.6) plt.show()
```

- 🔷 Строит гистограмму вероятностей (`prob_hist`).

📌 6.2 Запуск всего процесса

python Копировать Редактировать

```
if __name__ == "__main__": print("\nЗапуск задачи Пункт 3: гистограмма...") p3_result = p3_run() print("Результат задачи Пункт 3:") print(p3_result) plot_histogram(p3_result["probability_histogram"])
```

- 🔷 Выполняем `p3_run()` и строим гистограмму.

📌 Итог

Мастер управляет распределением данных и сбором результатов от воркеров, затем строит глобальную гистограмму.

Если что-то не так со сходящейся гистограммой, проблема может быть в:

- Неправильной логике распределения данных
- Ошибках в коде воркеров
- Ошибке в нормализации

Ты можешь проверить, совпадает ли `total_count` с `len(csv_data)`, чтобы убедиться, что все данные учтены. 🚀

Теперь объясни подробно как работает worker:

```
import socket
import threading
import json
import sys
import os
import numpy as np  # Добавляем numpy


def get_p3_filename(port):
    """ Возвращает уникальное имя файла для хранения данных Пункта 3. """
    return f"p3_worker_data_{port}.json"


def save_data(data, filename):
    """ Сохраняет данные на диск в формате JSON. """
    with open(filename, "w") as f:
        json.dump(data, f)


def load_data(filename):
    """ Загружает данные из файла, если он существует. """
    if not os.path.exists(filename):
        return []
    with open(filename, "r") as f:
        return json.load(f)


def handle_client(conn, addr, port):
    try:
        data_received = ""
        while "\n" not in data_received:
            chunk = conn.recv(4096).decode('utf-8')
            if not chunk:
                break
            data_received += chunk

        if not data_received:
            return

        command = json.loads(data_received.strip())
        cmd = command.get("command")

        if cmd == "store_data_p3":
            d = command.get("data", [])
            save_data(d, get_p3_filename(port))
            response = {"status": "Data stored for p3", "records": len(d)}
        elif cmd == "p3_map":
            stored = load_data(get_p3_filename(port))
            bins = [1, 2, 3, 4, 5, 6, 7, 8]  # Границы интервалов
            hist, _ = np.histogram(stored, bins=bins)  # Используем numpy для корректного распределения

            local_hist = {f"{bins[i]}-{bins[i + 1]}": int(hist[i]) for i in range(len(hist))}

            response = {"status": "p3 map task executed", "map_result": local_hist}
        else:
            response = {"error": "Unknown command"}

        message = json.dumps(response) + "\n"
        conn.sendall(message.encode('utf-8'))
    except Exception as e:
        error_msg = json.dumps({"error": str(e)}) + "\n"
        conn.sendall(error_msg.encode('utf-8'))
    finally:
        conn.close()


def start_server(port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('127.0.0.1', port))
    s.listen(5)
    print(f"Worker Node запущен на порту {port}")

    while True:
```

```
        conn, addr = s.accept()
        t = threading.Thread(target=handle_client, args=(conn, addr,
port))
        t.daemon = True
        t.start()


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Использование: python worker.py <port>")
        sys.exit(1)
    port = int(sys.argv[1])
    start_server(port)
```

И после этого покажи явно, как использовалась парадигма Map-Reduce для всех трех задач и где именно реализован Map, а где Reduce

🔷 Подробное объяснение кода `worker.py`

Этот скрипт представляет собой **воркер**, который принимает запросы от мастера, выполняет вычисления и отправляет результаты обратно.

🔶 Основная логика работы воркера

1. **Запуск сервера**
   - Воркер создает TCP-сервер, который слушает входящие соединения.
   - Принимает соединения и запускает обработку в **отдельном потоке**.
2. **Обработка команд от мастера**
   - `store_data_p3` → сохраняет переданные данные.
   - `p3_map` → строит **локальную гистограмму** и отправляет ее мастеру.
3. **Использование numpy**
   - Для построения гистограммы применяется `numpy.histogram()`, который автоматически распределяет данные по интервалам.

🔷 Разбор кода построчно

1 Функция `get_p3_filename(port)`

python Копировать Редактировать

```
def get_p3_filename(port): """ Возвращает уникальное имя файла для хранения данных Пункта
3. """ return f"p3_worker_data_{port}.json"
```

🔷 Генерирует **уникальное имя файла** на основе порта воркера.
🔷 Это нужно, чтобы разные воркеры не перезаписывали данные друг друга.

2 Функции сохранения и загрузки данных

python Копировать Редактировать

```
def save_data(data, filename): """ Сохраняет данные на диск в формате JSON. """ with
open(filename, "w") as f: json.dump(data, f)
```

🔷 Записывает переданный массив данных в **JSON-файл**.

python Копировать Редактировать

```
def load_data(filename): """ Загружает данные из файла, если он существует. """ if not
os.path.exists(filename): return [] with open(filename, "r") as f: return json.load(f)
```

🔷 Загружает данные из файла.
🔷 Если файла нет, возвращает **пустой массив**.

3 Обработка клиентских соединений (мастера)

python Копировать Редактировать

```
def handle_client(conn, addr, port):
```

🔷 Эта функция выполняется **в отдельном потоке** для каждого соединения.
🔷 Считывает команду от мастера и обрабатывает ее.

📌 **Чтение данных из сокета**

python Копировать Редактировать

```
data_received = "" while "\n" not in data_received: chunk = conn.recv(4096).decode('utf-8')
if not chunk: break data_received += chunk
```

🔹 Читает сообщение **пакетами по 4096 байт**, пока не получит `\n` .

📌 **Разбор команды**

python Копировать Редактировать

```
command = json.loads(data_received.strip()) cmd = command.get("command")
```

🔹 Декодирует JSON и извлекает команду (`command`).

4 Команда `"store_data_p3"` (сохранение данных)

python Копировать Редактировать

```
if cmd == "store_data_p3": d = command.get("data", []) save_data(d, get_p3_filename(port))
response = {"status": "Data stored for p3", "records": len(d)}
```

🔹 Воркер получает массив данных, **сохраняет их в файл** и отправляет мастеру ответ.

Пример ответа мастеру:

json Копировать Редактировать

```
{"status": "Data stored for p3", "records": 4}
```

5 Команда `"p3_map"` (вычисление локальной гистограммы)

python Копировать Редактировать

```
elif cmd == "p3_map": stored = load_data(get_p3_filename(port)) bins = [1, 2, 3, 4, 5, 6,
7, 8] # Границы интервалов hist, _ = np.histogram(stored, bins=bins)
```

🔹 Загружает **ранее сохраненные данные**.

🔹 Использует `numpy.histogram()` , чтобы распределить данные по **интервалам** (`bins`).

python Копировать Редактировать

```
local_hist = {f"{bins[i]}-{bins[i + 1]}": int(hist[i]) for i in range(len(hist))}
```

🔹 Преобразует массив `hist` в словарь `{интервал: частота}` .

Пример результата для одного воркера:

json Копировать Редактировать

```
{ "2-3": 1, "3-4": 2, "4-5": 1, "5-6": 0, "6-7": 0, "7-8": 0 }
```

🔹 Отправляет мастеру результат:

python Копировать Редактировать

```
response = {"status": "p3 map task executed", "map_result": local_hist}
```

6 Обработка ошибок

python Копировать Редактировать

```
except Exception as e: error_msg = json.dumps({"error": str(e)}) + "\n"
conn.sendall(error_msg.encode('utf-8'))
```

🔹 В случае ошибки отправляет **JSON с описанием ошибки**.

7 Запуск сервера воркера

python Копировать Редактировать

```
def start_server(port): s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', port)) s.listen(5) print(f"Worker Node запущен на порту {port}") while
True: conn, addr = s.accept() t = threading.Thread(target=handle_client, args=(conn, addr,
port)) t.daemon = True t.start()
```

🔹 Создает сервер на `127.0.0.1:port` .

🔹 Принимает соединения и обрабатывает их **в отдельных потоках**.

Запуск воркера:

bash Копировать Редактировать

```
python worker.py 5000
```

🔹 Запустит воркера на порту `5000` .

🔷 Реализация парадигмы **Map-Reduce** в коде

MapReduce — это модель для **распределенной обработки данных**, состоящая из двух этапов:

- **Map** (разбиение данных и вычисление частичных результатов)
- **Reduce** (объединение и агрегирование результатов)

🔶 Где реализован этап Map?

📍 В воркере, в обработке команды `"p3_map"` :

python Копировать Редактировать

```
elif cmd == "p3_map": stored = load_data(get_p3_filename(port)) hist, _ =
np.histogram(stored, bins=bins)
```

🔷 Каждый воркер обрабатывает **только свои данные** и строит **локальную гистограмму**.

Map в действии:

- Воркер получает свои данные (например, `[1.5, 2.3, 2.9, 3.1]`).
- Создает локальную гистограмму:

json Копировать Редактировать

```
{ "1-2": 1, "2-3": 2, "3-4": 1 }
```

- Отправляет эту гистограмму мастеру.

🔶 Где реализован этап Reduce?

📍 В мастере, в обработке локальных гистограмм:

python Копировать Редактировать

```
for local_hist in map_results: for bin_key, count in local_hist.items():
global_hist[bin_key] += count total_count += count
```

🔷 **Складывает все локальные гистограммы** в одну **глобальную**.

Reduce в действии:

- Допустим, есть 3 воркера с такими локальными гистограммами:

json Копировать Редактировать

```
Worker 1: {"1-2": 1, "2-3": 2, "3-4": 1} Worker 2: {"4-5": 2, "5-6": 1} Worker 3: {"6-7":
1, "7-8": 1}
```

- После Reduce (суммирования) получается:

json Копировать Редактировать

```
{ "1-2": 1, "2-3": 2, "3-4": 1, "4-5": 2, "5-6": 1, "6-7": 1, "7-8": 1 }
```

- Далее вычисляется **нормированная вероятность** для каждой группы.

📌 Итог

- **Map** выполняется в воркерах: каждый воркер считает **локальную гистограмму**.
- **Reduce** выполняется в мастере: он **объединяет локальные гистограммы** в одну глобальную.

🚀 Это классическая реализация MapReduce для построения распределенной гистограммы!

Покажи Map и Reduce Для всех двух задач: Пункт 2.
Выборочное среднее
1) Вфайлер2.csvданатаблица с единственным столбцом x — наблюдения случайной
величины X
2) Посчитайте выборочное среднее для X
Пункт 3. Гистограмма
1) Вфайлер3.csvданатаблица с единственным столбцом x — наблюдения случайной
величины X
2) Постройте вероятностную гистограмму для X . В качестве разбиения веще
ственной прямой, которое порождает столбцы гистограммы, используйте значения
(12345678)

🔷 MapReduce для двух задач: **Выборочное среднее** и **Гистограмма**

📌 Пункт 2: Выборочное среднее (Mean Calculation)

Цель: Найти выборочное среднее (среднее арифметическое) значений в файле `p2.csv`.

🔷 Как работает MapReduce?

1. **Map (на воркерах)**
   - Каждый воркер получает **часть данных**.
   - Считает **сумму значений** и **количество элементов**.
   - Отправляет результаты мастеру.
2. **Reduce (на мастере)**
   - Получает суммы и количества от всех воркеров.
   - Считает **глобальную сумму** и **общее количество значений**.
   - Вычисляет выборочное среднее:

$$\bar{X} = \frac{\sum X}{N}$$

🔷 Реализация Map (на воркере)

📍 Команда: `"p2_map"`

python Копировать Редактировать

```
elif cmd == "p2_map": stored = load_data(get_p2_filename(port)) local_sum = sum(stored)
local_count = len(stored) response = { "status": "p2 map task executed", "sum": local_sum,
"count": local_count }
```

Что делает этот код?

🔷 Считает **локальную сумму** и **количество элементов** в части данных.

🔷 Возвращает результат мастеру.

🔷 Реализация Reduce (на мастере)

📍 Сбор данных от всех воркеров

python Копировать Редактировать

```
map_results = [] for address in workers: command = {"command": "p2_map"} res =
send_command(address, command) map_results.append(res) total_sum = sum(res["sum"] for res
in map_results) total_count = sum(res["count"] for res in map_results) mean = total_sum /
total_count if total_count > 0 else 0
```

Что делает этот код?

🔷 Получает **локальные суммы** и **количество элементов** от всех воркеров.

🔷 Считает **глобальную сумму** и **общее количество**.

🔷 Вычисляет **выборочное среднее**.

📌 Пункт 3: Гистограмма

Цель: Построить **вероятностную гистограмму** по данным из `p3.csv`.

🔷 Как работает MapReduce?

1. **Map (на воркерах)**
   - Каждый воркер получает **часть данных**.
   - Считает **локальную гистограмму**.
   - Отправляет результат мастеру.
2. **Reduce (на мастере)**
   - Собирает локальные гистограммы.
   - Суммирует значения для каждой группы.
   - Нормирует значения, чтобы получить вероятностное распределение.

🔷 Реализация Map (на воркере)

📍 Команда: `"p3_map"`

python Копировать Редактировать

```
elif cmd == "p3_map": stored = load_data(get_p3_filename(port)) bins = [1, 2, 3, 4, 5, 6,
7, 8] hist, _ = np.histogram(stored, bins=bins) local_hist = {f"{bins[i]}-{bins[i + 1]}":
int(hist[i]) for i in range(len(hist))} response = {"status": "p3 map task executed",
"map_result": local_hist}
```

Что делает этот код?

🔷 Загружает **локальные данные**.

🔷 Разбивает их на **интервалы** с `numpy.histogram()`.

🔷 Возвращает **локальную гистограмму** мастеру.

🔷 Реализация Reduce (на мастере)

📍 Сбор данных от всех воркеров:

python Копировать Редактировать

```
map_results = [] for address in workers: command = {"command": "p3_map"} res =
send_command(address, command) map_results.append(res.get("map_result", {})) bins = [1, 2,
3, 4, 5, 6, 7, 8] global_hist = {f"{bins[i]}-{bins[i+1]}": 0 for i in range(len(bins) - 1)}
total_count = 0 for local_hist in map_results: for bin_key, count in local_hist.items():
global_hist[bin_key] += count total_count += count prob_hist = {bin_key: (count /
total_count if total_count > 0 else 0) for bin_key, count in global_hist.items()}
```

Что делает этот код?

🔷 Получает **локальные гистограммы** от воркеров.

🔷 Складывает их в **глобальную гистограмму**.

🔷 Вычисляет **нормированные вероятности**.

📌 Итог: Где Map и Reduce в каждой задаче?

Задача	Map (на воркерах)	Reduce (на мастере)
Пункт 2: Среднее	Вычисляет сумму и количество	Считает общее среднее
Пункт 3: Гистограмма	Создает локальную гистограмму	Объединяет и нормирует

🚀 **Теперь у нас распределенная система, которая вычисляет среднее и строит гистограмму с помощью MapReduce!**