



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии
(ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 Программная инженерия

Отчет по лабораторной работе №5 по дисциплине анализ алгоритмов

Тема: Конвейерная обработка

Студент: Серова М.Н.

Группа: ИУ7-55Б

Оценка (баллы): _____

Преподаватель: Волкова Л.Л.

Москва, 2021

Содержание

Введение	2
1 Аналитическая часть	3
1.1 Выделенные стадии конвейерной обработки	3
1.1.1 Загрузка логинов и паролей из базы данных	3
1.1.2 Многократное последовательное хеширование	3
1.1.3 Загрузка в базу данных	4
1.2 Требования к программному обеспечению	4
1.3 Вывод	5
2 Конструкторская часть	6
2.1 Схемы алгоритмов	6
2.2 Структуры данных	6
2.3 Вывод	8
3 Технологическая часть	9
3.1 Средства реализации	9
3.2 Реализация алгоритмов	9
3.3 Выводы	17
4 Экспериментальная часть	18
4.1 Пример работы программы	18
4.2 Технические характеристики	20
4.3 Время выполнения алгоритмов	20
4.4 Тестирование	21
4.5 Выводы	23
Заключение	24
Список литературы	25

Введение

Конвейеризация (конвейерная обработка) – способ организации вычислений, основанный на разделении подлежащей исполнению функции на более мелкие задачи, называемые ступенями (этапами) конвейера, и выделении для каждой из них отдельного блока. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько задач.

В данной работе рассматривается конвейеризация алгоритма медленного хеширования извлеченных из исходной таблицы базы данных паролей с последующим их сохранением в итоговую таблицу базы.

Целью данной работы является реализация асинхронного взаимодействия потоков на примере конвейерной обработки данных.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1. Изучить асинхронное взаимодействие на примере конвейерной обработки данных.
2. Привести схему конвейерных вычислений.
3. Описать используемые структуры данных.
4. Определить средства программной реализации.
5. Реализовать и протестировать ПО.
6. Провести сравнительный последовательной и конвейерной реализации по затрачиваемым ресурсам (времени работы).
7. Изучить время, затрачиваемое на нахождение заявки в очереди к каждому этапу конвейера.

1 Аналитическая часть

В данном разделе рассматриваются принципы и идея конвейерной обработки данных, а также приводится описание решаемой задачи и выделенных стадий конвейерной обработки.

1.1 Выделенные стадии конвейерной обработки

В данной работе в качестве алгоритма, реализованного для конвейеризации, используется шифрование паролей с сохранением в базу данных. Таким образом, было выделено три ленты конвейера:

1. Загрузка логинов и паролей из базы данных.
2. Многократное последовательное (медленное) хеширование полученной строки для большей безопасности.
3. Загрузка полученного значения в базу данных.

1.1.1 Загрузка логинов и паролей из базы данных

Изначальные логины пользователей и пароли для хеширования загружаются из базы данных.

1.1.2 Многократное последовательное хеширование

Существует много алгоритмов хеш-шифрования данных, наиболее распространенные из них приведены ниже [1].

1. MD5 - алгоритм дайджеста сообщений (Message-Digest algorithm), генерирует 128-битное хэш-значение; не может предотвратить коллизии и, следовательно, может быть взломан.
2. SHA-1 - Secure Hash Algorithm 1, может генерировать дайджест сообщения - 160-битное хеш-значение, обычная форма - 40 шестнадцатеричных чисел; не является достаточно безопасным, был обнаружен эффективно-го метод атаки.
3. SHA-2/SHA-256 - Secure Hash Algorithm 2, генерирует 256-битное хеш-значение; существуют алгоритмы SHA-224, SHA-384, SHA-512, названия которых содержат двоичную длину SHA-2.

4. HMAC - Hash-based Message Authentication Code, использует алгоритм хеширования, принимает сообщение М и ключ К в качестве входных данных и генерирует дайджест сообщения фиксированной длины в качестве выходных данных.

Существует несколько способов увеличения надежности хеш-функций [2]:

1. использование "соли" добавление символьной последовательности в конец строки;
2. использование нескольких этапов хеширования, использующих разные хеш-функции;
3. растяжение пароля - итеративный, или рекурсивный, алгоритм, который вычисляет хэш самого себя большое количество раз (такой способ усложняет проведение брутфорс атаки (перебор значений) для нахождения пароля выбранного пользователя ввиду низкой скорости проверки каждого пароля.).

В данной работе для обеспечения безопасности хранения данных используется хеш-функция SHA-512 и способ растяжения паролей.

1.1.3 Загрузка в базу данных

По завершении всех предыдущих этапов обработанные пароли пользователей заносятся в локальную реляционную базу данных.

1.2 Требования к программному обеспечению

На основе приведенного алгоритма можно выдвинуть требования к разрабатываемому ПО:

- выходные данные - время работы последовательного выполнения действий (вещественное число);
- время работы конвейера (вещественное число);
- минимальное, среднее, максимальное времена ожидания заявки в очереди каждой из конвейерных лент (вещественные числа);
- наличие обработки некорректного ввода.

1.3 Вывод

Был рассмотрен алгоритм шифрования пароля пользователя и его сохранение в базу данных. Были рассмотрены шаги алгоритма шифрования, выполняемые каждой лентой конвейера для оптимизации работы алгоритма. Выдвинуты требования к разрабатываемому ПО: выходные данные (включающие в себя время работы конвейера, минимальное, среднее, а также максимальное время ожидания заявки в очереди) и наличие обработки некорректного ввода.

2 Конструкторская часть

Данный раздел содержит схемы конвейерной обработки данных, последовательного и конвейерного алгоритма.

2.1 Схемы алгоритмов

В данном пункте раздела представлены схемы реализуемых в работе алгоритмов.

На рисунке 2.1 представлена схема организации конвейерных вычислений на примере конвейера с тремя лентами.

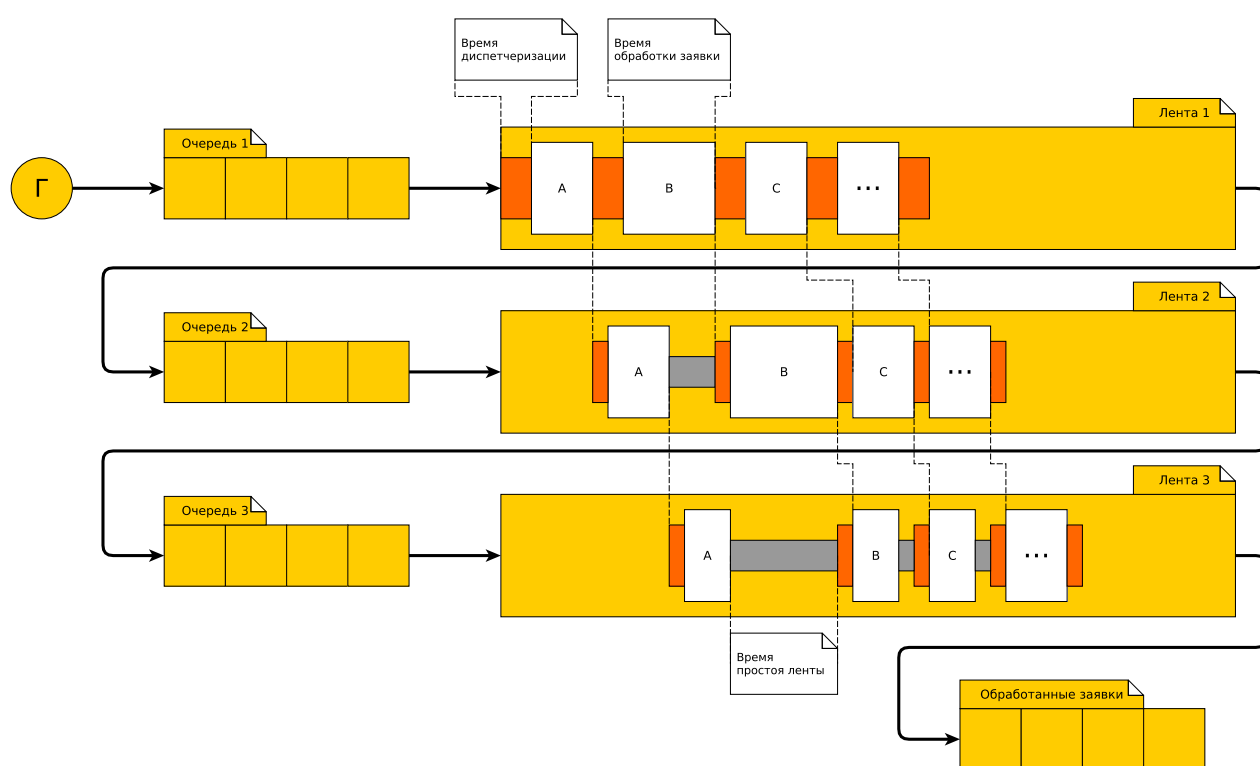


Рис. 2.1: Схема организации конвейера с тремя лентами

2.2 Структуры данных

Для удобства работы были выделены следующие классы:

- UserStats

— поля

* times - массив из объектов класса Times, содержащий время поступления и выхода для каждой ленты конвейера;

- * user - объект класса User, содержащий данные пользователя;
- * current - хэш-значение пароля;
- * number - идентификатор объекта класса;
- методы
 - * set_time - установка времени входа/выхода для конкретной ленты;
 - * get_time - получение времени входа/выхода для конкретной ленты;
- User
 - поля
 - * login - логин пользователя;
 - * password - пароль пользователя;
 - методы
 - * generate_pass - генерирует случайный пароль;
- Time
 - поля
 - * time_in - время поступления на ленту конвейера;
 - * time_out - время выхода с ленты конвейера;
 - методы
 - * set - установка времени поступления/выхода;
 - * get - получение времени поступления/выхода;

Таким образом, программа генерирует заданное в программе количество пользователей и их данных, затем формирует 4 очереди и 3 ленты конвейера и запускает конвейерную обработку. После окончания выводится время конвейерной и последовательной обработки одних и тех же данных, после выводится лог, содержащий время поступления в очередь и выхода задач для каждой ленты конвейера.

2.3 Вывод

В данном разделе приведена схема работы конвейера, выделены структуры данных для дальнейшей реализации программного обеспечения.

3 Технологическая часть

Данный раздел содержит обоснование выбора языка и среды разработки, реализацию алгоритмов.

3.1 Средства реализации

Для реализации программы был выбран язык программирования Python [3]. Такой выбор обусловлен следующими причинами:

- имеется большой опыт разработки;
- имеет большое количество расширений и библиотек, в том числе библиотеку для работы с потоками, измерения времени, построения графиков;
- обладает информативной документацией;

3.2 Реализация алгоритмов

В листингах 3.1 - ?? представлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Класс User

```
1 PC = 'qwertyuiopasdfghjklzxcvbnm1234567890 '
2 PCS = len(PC)
3
4 class User:
5     def __init__(self, login: str = None, password: str = None):
6         if login is None:
7             self.__login = str(uuid.uuid4())
8         else:
9             self.__login = login
10        if password is None:
11            self.__password = self.__generate_pass()
12        else:
13            self.__password = password
14
15    @staticmethod
16    def __generate_pass():
17        size = random.randint(16, 2**9)
18        a = ''
19        for i in range(size):
20            a += PC[random.randint(0, PCS - 1)]
21        return a
22
23    @property
24    def login(self):
25        return self.__login
26
27    @property
28    def password(self):
29        return self.__password
```

Листинг 3.2: Класс Time

```
1 class Time:
2     def __init__(self):
3         self.__time_in = None
4         self.__time_out = None
5
6     def set(self, t, is_in: bool):
7         if is_in:
8             if self.__time_in is not None:
9                 raise Exception("Время поступления на ленту уже записано!")
10            self.__time_in = t
11        elif not is_in:
12            if self.__time_out is not None:
13                raise Exception("Время выхода с ленты уже записано!")
14            self.__time_out = t
15        else:
16            raise Exception("При установке параметр is_in – обязательный!")
17
18    def get(self, is_in: bool = None):
19        if is_in is None:
20            if self.__time_out is None and self.__time_in is None:
21                raise Exception("Время не установлено!")
22            elif self.__time_out is None:
23                raise Exception("Время поступления на ленту не установлено!")
24            elif self.__time_out is None:
25                raise Exception("Время выхода с ленты не установлено!")
26            return [self.__time_in, self.__time_out]
27        elif is_in:
28            if self.__time_out is None:
29                raise Exception("Время поступления на ленту не установлено!")
30            return self.__time_in
31        else:
32            if self.__time_out is None:
33                raise Exception("Время выхода с ленты не установлено!")
34            return self.__time_out
```

Листинг 3.3: Класс UserStats

```
1 class UserStats:
2     cnt = 0
3
4     def __init__(self, login: str = None, password: str = None,
5         do_not_init: bool = True):
6         self.__times = [Time() for _ in range(3)]
7         self.__number = self.cnt
8         if do_not_init:
9             self.user = None
10        else:
11            self.user = User(login, password)
12            self.current = ''
13            UserStats.cnt += 1
14
15    def set_time(self, time_, stage: int, is_in: bool):
16        # is_in — True, если на вход, False, если на выход
17        # stage — номер ленты
18        if stage < 0 or stage > 2 or stage is None:
19            raise Exception("Неправильный номер ленты!")
20        self.__times[stage].set(time_, is_in)
21
22    def get_time(self, is_in: bool = None, stage: int = None):
23        # is_in — True, если на вход, False, если на выход, обязательный
24        # stage — номер ленты, None = выдать все
25
26        if stage is not None and (stage < 0 or stage > 2):
27            raise Exception("Неправильный номер ленты!")
28        if stage is None:
29            if is_in is None:
30                return [self.__times[i].get() for i in range(3)]
31            else:
32                return [self.__times[i].get(is_in) for i in range(3)]
33        else:
34            return self.__times[stage].get(is_in)
35
36    def get_number(self):
37        return self.__number
```

Листинг 3.4: Файл master.py (часть 1)

```
1 def init_db(con: sqlite3.Connection):
2     cur = con.cursor()
3     cur.execute('create table if not exists users_out (login text,
4         password text)')
5     cur.execute('create table if not exists users_in (i int, login
6         text, password text)')
7     con.commit()
8
9 def fill_input_db(con: sqlite3.Connection, count=30):
10     cur = con.cursor()
11     for index in range(count):
12         m = User()
13         cur.execute(f'insert into users_in (i, login, password) values
14             ({index}, "{m.login()}", "{m.password()}")')
15     con.commit()
16
17 def get_list_size_from_db(con: sqlite3.Connection) -> int:
18     cur = con.cursor()
19     cur.execute('select count(*) from users_in')
20     return cur.fetchone()[0]
21
22 def clear_db(con: sqlite3.Connection):
23     cur = con.cursor()
24     cur.execute('drop table if exists users_out')
25     cur.execute('drop table if exists users_in')
26     con.commit()
27
28 def generate_users(users_count: int) -> [UserStats]:
29     users = []
30
31     for i in range(users_count):
32         users.append(UserStats(do_not_init=False))
33
34     return users
```

Листинг 3.5: Файл master.py (часть 2)

```
1 def load_user(u: UserStats):
2     con = sqlite3.connect('app.db')
3     c = con.cursor()
4     c.execute(f'select login, password from users_in where i = {u.
5         get_number()}')
6     login, password = c.fetchone()
7     u.user = User(login, password)
8
9 def get_hashed(u: UserStats):
10
11     for iter in range(2000):
12         u.current = sha512((u.current + str(iter)).encode('Utf-8')).
13             hexdigest()
14
15 def insert(u: UserStats):
16     con = sqlite3.connect('app.db')
17     c = con.cursor()
18     c.execute(f"insert into users_out (login, password) values ('{u.
19         user.login()}', '{u.current}')"
20
21     con.commit()
22     con.close()
23
24 def tex_table(stats, stages):
25
26     print("\\hline")
27     print('Stage N & Task M & Start Time & End Time\\\\\\')
28     print("\\hline")
29
30     for stat_num, stat in enumerate(stats):
31         for stage in range(stages):
32             times = stat.get_time(stage=stage)
33             print(
34                 f'Stage: {stage + 1} & Task: {stat_num + 1} & {times[0] -
35                 start_time:.6f} & {times[1] - start_time:.6f} \\\\\\')
36
37     print("\\hline")
```

Листинг 3.6: Файл master.py (часть 3)

```
1 def job(task: Callable, in_queue: SimpleQueue, out_queue:
   SimpleQueue, stage: int):
2     while True:
3         data: UserStats = in_queue.get()
4
5         if data is None:
6             out_queue.put(data)
7             break
8
9         data.set_time(time.time(), stage, True)
10        task(data)
11        data.set_time(time.time(), stage, False)
12        out_queue.put(data)
13
14 def test_serial(users):
15     for user in users:
16         load_user(user)
17         get_hashed(user)
18         insert(user)
19
20 if __name__ == '__main__':
21     stages_count = 3
22
23     connection = sqlite3.connect('app.db')
24     clear_db(connection)
25     init_db(connection)
26     fill_input_db(connection)
27
28     _users = generate_users(get_list_size_from_db(connection))
29
30     pipeline_time = 0
31     serial_time = 0
32     cnt = 10
33     for i in range(cnt):
34         clear_db(connection)
35         init_db(connection)
36         fill_input_db(connection)
37
38         passwords_queue = SimpleQueue()
39         salt_queue = SimpleQueue()
```


Листинг 3.7: Файл master.py (часть 4)

```
1     hash_queue = SimpleQueue()
2     result_queue = SimpleQueue()
3
4     add_salter = Process(target=job, args=(load_user,
5     passwords_queue, salt_queue, 0))
6     hasher = Process(target=job, args=(get_hashed, salt_queue,
7     hash_queue, 1))
8     inserter = Process(target=job, args=(insert, hash_queue,
9     result_queue, 2))
10    pipeline = [add_salter, hasher, inserter]
11
12    for u in _users:
13        passwords_queue.put(u)
14
15    passwords_queue.put(None)
16    start_time = time.time()
17    for worker in pipeline:
18        worker.start()
19
20    for worker in pipeline:
21        worker.join()
22    end_time = time.time()
23    pipeline_time += end_time - start_time
24
25    start_time_ = time.time()
26    test_serial(_users)
27    end_time_ = time.time()
28    serial_time += end_time_ - start_time_
29
30    pipeline_time /= cnt
31
32    print(f'Pipeline time = {pipeline_time * 1e6} mks')
33
34    serial_time /= cnt
35    print(f'Serial time = {serial_time * 1e6} mks')
36    print('Press to get log')
37    input()
38
39    stats = []
```

Листинг 3.8: Файл master.py (часть 5)

```
1  while not result_queue.empty():
2      stats.append(result_queue.get())
3  stats.pop()
4
5  deltas = [], [], []
6  for stat in stats:
7      for stage in range(stages_count):
8          stage_stat = stat.get_time(stage=stage)
9          deltas[stage].append(stage_stat[1] - stage_stat[0])
10
11 for stage in range(stages_count):
12     print(f'Max time on stage {stage + 1} = {max(deltas[stage]) *
13         1e6} mks')
14     print(f'Min time on stage {stage + 1} = {min(deltas[stage]) *
15         1e6} mks')
16     print(f'Avg time on stage {stage + 1} = {sum(deltas[stage]) /
17         len(deltas[stage]) * 1e6} mks\n')
18
19 connection.close()
20
21 tex_table(stats, stages_count)
```

3.3 Выводы

В данном разделе была реализована конвейерная обработка данных: изменение, хеширование и сохранение паролей пользователей.

4 Экспериментальная часть

В данном разделе сравниваются реализованные алгоритмы, дается сравнительная оценка затрат на время.

4.1 Пример работы программы

Пример работы программы представлен на рисунке 4.1.

```

Pipeline time = 133182.73544311523 mks
Serial time = 228827.65769958496 mks
Press to get log

Max time on stage 1 = 812.7689361572266 mks
Min time on stage 1 = 133.99124145507812 mks
Avg time on stage 1 = 244.4903055826823 mks

Max time on stage 2 = 12578.487396240234 mks
Min time on stage 2 = 3556.489944458008 mks
Avg time on stage 2 = 9844.144185384115 mks

Max time on stage 3 = 3536.4627838134766 mks
Min time on stage 3 = 2316.9517517089844 mks
Avg time on stage 3 = 3023.9184697469077 mks

\hline
Stage N & Task M & Start Time & End Time\\
\hline
Stage: 1 & Task: 1 & 0.002179 & 0.002991 \\
Stage: 2 & Task: 1 & 0.003429 & 0.016008 \\
Stage: 3 & Task: 1 & 0.016438 & 0.019504 \\
\hline
Stage: 1 & Task: 2 & 0.003314 & 0.003629 \\
Stage: 2 & Task: 2 & 0.016270 & 0.026802 \\
Stage: 3 & Task: 2 & 0.027042 & 0.030579 \\
\hline
Stage: 1 & Task: 3 & 0.003776 & 0.004056 \\
Stage: 2 & Task: 3 & 0.026944 & 0.037537 \\
Stage: 3 & Task: 3 & 0.037839 & 0.040872 \\
\hline
Stage: 1 & Task: 4 & 0.004245 & 0.004612 \\
Stage: 2 & Task: 4 & 0.037680 & 0.048180 \\
Stage: 3 & Task: 4 & 0.048373 & 0.051491 \\
\hline
Stage: 1 & Task: 5 & 0.004766 & 0.005046 \\
Stage: 2 & Task: 5 & 0.048321 & 0.058671 \\
Stage: 3 & Task: 5 & 0.058857 & 0.062011 \\
\hline
Stage: 1 & Task: 6 & 0.005198 & 0.005489 \\
Stage: 2 & Task: 6 & 0.058783 & 0.069260 \\
Stage: 3 & Task: 6 & 0.069451 & 0.071884 \\
\hline
Stage: 1 & Task: 7 & 0.005622 & 0.005898 \\
Stage: 2 & Task: 7 & 0.069384 & 0.079809 \\
Stage: 3 & Task: 7 & 0.080004 & 0.083281 \\
\hline
Stage: 1 & Task: 8 & 0.006060 & 0.006336 \\
Stage: 2 & Task: 8 & 0.079923 & 0.090405 \\
Stage: 3 & Task: 8 & 0.090638 & 0.093818 \\
\hline
Stage: 1 & Task: 9 & 0.006461 & 0.006736 \\
Stage: 2 & Task: 9 & 0.090529 & 0.094146 \\
Stage: 3 & Task: 9 & 0.094260 & 0.096577 \\
\hline

```

Рис. 4.1: Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- операционная система: Ubuntu 20.01 Linux x86_64 [4];
- оперативная память: 8 Гб;
- процессор: AMD Ryzen5 4500U [5]:
 - количество физических ядер: 6;
 - количество логических ядер: 6.

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов измерялось на автоматически генерируемых в необходимом количестве пользовательских данных с использованием функции `time` библиотеки `time`. Усредненные результаты 10 замеров реального времени работы приведены в таблице ниже.

На рисунке 4.2 представлена зависимость времени выполнения алгоритма в зависимости от "плана на день количества пользователей для обработки" - на основе таблицы 4.1. Конвейерное выполнение занимает в среднем в 3 раз больше времени, чем многопоточное. Таким образом, проигрыш от использования конвейерной обработки при количестве пользователей от 3 до 80 приблизительно одинаков и составляет около 3 раз.

Таблица 4.1: Время выполнения алгоритма для разного количества пользователей в микросекундах

Размер	Последовательное выполнение	Конвейерная обработка
5	38291.287	36348.152
15	109756.374	73250.5798
25	181147.265	113172.388
35	252654.957	152421.474
45	322310.686	197659.373
55	399007.916	239235.663
65	461043.334	286625.170
75	541337.20	321961.14

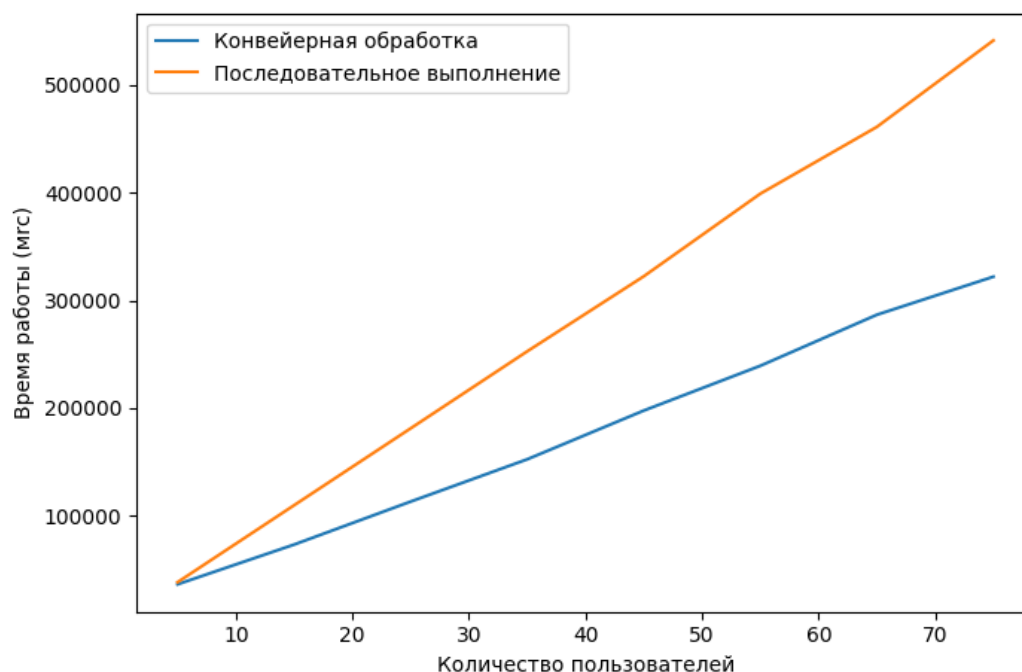


Рис. 4.2: Зависимость времени выполнения от количества пользователей

4.4 Тестирование

Для тестирования корректности работы ПО используется анализ логов. Полученная для последних 6 заявок таблица приведена ниже. На рисунке 4.3 представлены данные о максимальном, минимальном и среднем времени обработки заявок на каждой ленте конвейера.

Таблица 4.2: Полученный лог

Stage N	Task M	Start Time	End Time
Stage: 1	Task: 24	0.010745	0.189995
Stage: 2	Task: 24	0.190242	0.196989
Stage: 3	Task: 24	0.197229	0.204942
Stage: 1	Task: 25	0.190209	0.190594
Stage: 2	Task: 25	0.197179	0.201507
Stage: 3	Task: 25	0.205129	0.214046
Stage: 1	Task: 26	0.190706	0.191005
Stage: 2	Task: 26	0.201600	0.205533
Stage: 3	Task: 26	0.214208	0.219651
Stage: 1	Task: 27	0.191104	0.191360
Stage: 2	Task: 27	0.205596	0.209139
Stage: 3	Task: 27	0.219830	0.226994
Stage: 1	Task: 28	0.191455	0.191704
Stage: 2	Task: 28	0.209199	0.213150
Stage: 3	Task: 28	0.227133	0.232608
Stage: 1	Task: 29	0.191796	0.192047
Stage: 2	Task: 29	0.213213	0.216894
Stage: 3	Task: 29	0.232806	0.241898
Stage: 1	Task: 30	0.192139	0.192398
Stage: 2	Task: 30	0.216962	0.220743
Stage: 3	Task: 30	0.242102	0.247793

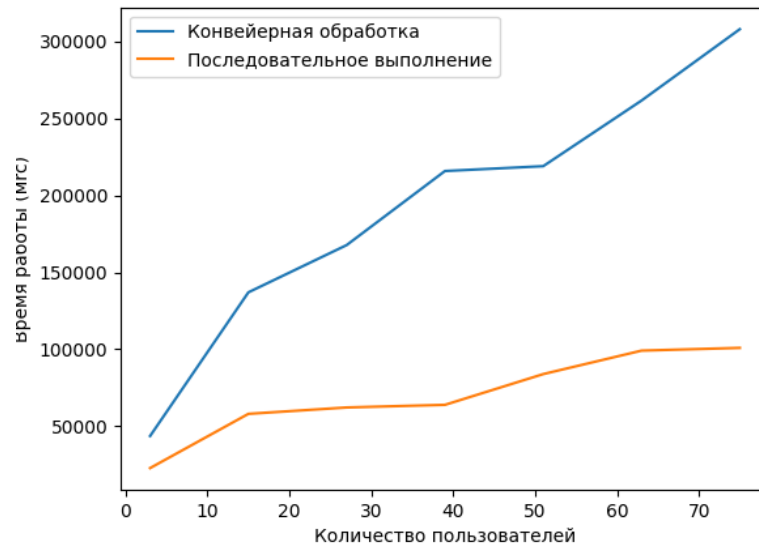


Рис. 4.3: Время выполнения заявок на каждой из лент конвейера

4.5 Выводы

В данном разделе были проведены измерения времени, затрачиваемого на загрузку данных о пользователе из базы данных, хеширование пароля и сохранения в базу данных полученного хэша. Для размера входной очереди конвейера, принадлежащего интервалу $[5, 85]$, проигрыш по сравнению с многопоточной обработкой составил приблизительно 3 раз с небольшими колебаниями при увеличении количества пользователей.

Заключение

В процессе выполнения лабораторной работы было реализовано асинхронное взаимодействие потоков на примере конвейерной обработки данных. В качестве конвейера рассмотрена работа с логинами и паролями пользователей: загрузка из базы данных, многократное хеширование хеш-функцией SHA-512 и сохранение полученных данных в базу данных.

Было исследовано время выполнения выше обозначенного алгоритма. В результате было выявлено, что в среднем для количества пользователей от 5 до 85 проигрыш от использования конвейерной обработки составляет 3 раз с небольшим увеличением при увеличении количества пользователей.

Список литературы

- [1] Подробное объяснение хеш-шифрования и инструментов md5, sha1, sha256, Java. Режим доступа: <https://russianblogs.com/article/80151483863/>. Дата обращения: 21.11.2021.
- [2] Риски и проблемы хеширования паролей. Режим доступа: <https://itnan.ru/post.php?c=1&p=271245>. Дата обращения: 21.11.2021.
- [3] About Python [Электронный ресурс]. Режим доступа: <https://www.python.org/about/>. Дата обращения: 20.10.2021.
- [4] Ubuntu по-русски [Электронный ресурс]. Режим доступа: <https://ubuntu.ru/>. Дата обращения: 20.10.2021.
- [5] AMD Ryzen™ 5 4500U. Режим доступа: <https://www.amd.com/ru/products/apu/amd-ryzen-5-4500u>. Дата обращения: 20.10.2021.
- [6] Параллельная обработка данных. Режим доступа: <https://parallel.ru/vvv/lec1.html>. Дата обращения: 21.11.2021.