МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 2

з дисципліни “*Бази даних. Частина 2*”
на тему “*Практика використання сервера Redis*”

Виконав
студент III курсу
групи КП-81
Бухаленков Дмитро Олександрович

Зарахована: Петрашенко А. В.

Київ 2021

Мета роботи: здобуття практичних навичок створення ефективних програм, орієнтованих на використання сервера Redis за допомогою мови Python.

Завдання роботи полягає у наступному:

Реалізувати можливості обміну повідомленнями між користувачами у оффлайн та онлайн режамах із можливістю фільтрації спам-повідомлень.

Окремі програмні компоненти та вимоги до них

1. Redis server (RS), що виконує наступні ролі:

   1.1. *Сховище,* що містить*:* дані користувачів, їхні групи (звичайний користувач та адміністратор), а також повідомлення, що пересилаються між ними.

   1.2. *Черга повідомлень*, які підлягають перевірці на спам та відправленню адресату.

   1.3. Інструмент *Publish/Subscribe* для ведення та розсилання журналу активності користувачів (див. *Список активностей для журналювання*).

2. Інтерфейс користувача (User Interface)

   2.1. *Звичайний користувач* має змогу виконувати вхід за ім’ям (без паролю), відправляти та отримувати (переглядати) повідомлення, отримувати дані про кількість **своїх** повідомлень, згрупованих за статусом (див. *Статуси повідомлень*).

   2.2. *Адміністратор* має змогу переглядати журнал подій, що відбулись (див. *Список активностей для журналювання),* переглядати список користувачів, які знаходяться online, переглядати статистику (N найбільш активних відправників повідомлень із відповідною кількістю, N найактивніших “спамерів” із відповідною кількістю).

3. *Виконувач* (worker) призначений для:

перегляду черги повідомлень, відбору повідомлення, перевірки його вмісту на наявність спаму (у випадку наявності спаму -- додавання запису в журнал)

Інші вимоги

1. Проаналізувавши матеріали ресурсів, наведений у пункті “Джерела”, обрати та обгрунтувати вибір структур даних Redis щодо реалізації наведених вище вимог, **обов’язково використати наступні структури даних** та інструменти Redis: List, Hash, Sorted List, Set, Pub/Sub.
2. Забезпечити роботу програмних засобів у режимі емуляції із можливістю генерації повідомлень від різних користувачів, налаштування кількості виконувачів та часу затримки обробки на спам з можливістю підключення адміністратора для перегляду подій, що відбуваються.
3. Перевірку на спам можна проемулювати за допомогою затримки на псевдовипадковий час та генерацію псевдовипадкового результату (Так/Ні).

Список активностей для журналювання

Вхід/вихід користувача, наявність спаму у повідомленні.

Статуси повідомлень

“Створено”, “У черзі”, “Перевіряється на спам”, “Заблоковано через спам”, “Відправлено адресату”, “Доставлено адресату”.

Вимоги до інтерфейсу користувача

Використовувати консольний (текстовий) інтерфейс користувача.

Результати:

- List – було використано для черги повідомлень, адже вставка в конець та в початок душе швидка для цієї структури даних.
- Hash – було використано для зберігання даних повідомлень, бо він дозволяє запам'ятовувати набори ключ-значення, і нам треба зберігати відправника, одержувача та саме повідомлення.
- Sorted List – було використано для списків найактивніших відправників та спамерів, адже нам потрібно їх сортувати.
- Set – було використано для списку користувачів, адже порядок нам не важливий, а пошук наявності користувача буде дуже швидким.
- Pub/Sub – було використано для логування подій, адже будь-хто може виступати publisher'ом і отримувачі отримають повідомлення про подію.

Основна логіка роботи користувача з повідомленнями та адміна зі статистикою:

RedisServer,py

```
class RedisServer(object):
   def __init__(self):
       self.__r = redis.Redis(charset="utf-8", decode_responses=True)

   def registration(self, username):
       if self.__r.hget('users:', username):
           raise Exception(f"User with name: \'{username}\' already
exists")

       user_id = self.__r.incr('user:id:')
       pipeline = self.__r.pipeline(True)
       pipeline.hset('users:', username, user_id)
       pipeline.hmset(f"user:{user_id}", {
           'login': username,
           'id': user_id,
           'queue': 0,
           'checking': 0,
           'blocked': 0,
           'sent': 0,
           'delivered': 0
       })

       pipeline.execute()
       logging.info(f"User {username} registered at
{datetime.datetime.now()} \n")
       return user_id
```

```
   def sign_in(self, username):
       user_id = self.__r.hget("users:", username)

       if not user_id:
           raise Exception(f"User {username} does not exist ")

       self.__r.sadd("online:", username)
       logging.info(f"User {username} logged in at
{datetime.datetime.now()} \n")
       self.__r.publish('users', "User %s signed in" %
self.__r.hmget(f"user:{user_id}", 'login')[0])
       return int(user_id)

   def sign_out(self, user_id) -> int:
       logging.info(f"User {user_id} signed out at
{datetime.datetime.now()} \n")
       self.__r.publish('users', "User %s signed out" %
self.__r.hmget(f"user:{user_id}", 'login')[0])
       return self.__r.srem("online:", self.__r.hmget(f"user:{user_id}",
'login')[0])

   def create_message(self, message_text, consumer, sender_id) -> int:

       message_id = int(self.__r.incr('message:id:'))
       consumer_id = self.__r.hget("users:", consumer)

       if not consumer_id:
           raise Exception(f"{consumer} user does not exist, user can't
send a message")

       pipeline = self.__r.pipeline(True)

       pipeline.hmset('message:%s' % message_id, {
           'text': message_text,
           'id': message_id,
           'sender_id': sender_id,
           'consumer_id': consumer_id,
           'status': "created"
       })
       pipeline.lpush("queue:", message_id)
       pipeline.hmset('message:%s' % message_id, {
           'status': 'queue'
       })
       pipeline.zincrby("sent:", 1, "user:%s" %
self.__r.hmget(f"user:{sender_id}", 'login')[0])
       pipeline.hincrby(f"user:{sender_id}", "queue", 1)
       pipeline.execute()

       return message_id

   def get_messages(self, user_id):
       messages = self.__r.smembers(f"sentto:{user_id}")
       messages_list = []
       for message_id in messages:
           message = self.__r.hmget(f"message:{message_id}", ["sender_id",
"text", "status"])
           sender_id = message[0]
           messages_list.append("From: %s - %s" %
(self.__r.hmget("user:%s" % sender_id, 'login')[0], message[1]))
```

```
            if message[2] != "delivered":
                pipeline = self.__r.pipeline(True)
                pipeline.hset(f"message:{message_id}", "status",
"delivered")
                pipeline.hincrby(f"user:{sender_id}", "sent", -1)
                pipeline.hincrby(f"user:{sender_id}", "delivered", 1)
                pipeline.execute()
        return messages_list

    def get_message_statistics(self, user_id):
        current_user = self.__r.hmget(f"user:{user_id}", ['queue',
'checking', 'blocked', 'sent', 'delivered'])
        return "In queue: %s\nChecking: %s\nBlocked: %s\nSent:
%s\nDelivered: %s" % tuple(current_user)

    def get_online_users(self) -> list:
        return self.__r.smembers("online:")

    def get_top_senders(self, amount_of_top_senders) -> list:
        return self.__r.zrange("sent:", 0, int(amount_of_top_senders) - 1,
desc=True, withscores=True)

    def get_top_spamers(self, amount_of_top_spamers) -> list:
        return self.__r.zrange("spam:", 0, int(amount_of_top_spamers) - 1,
desc=True, withscores=True)
```

Виконувач, що псевдовипадково відбирає повідомлення зі спамом

Worker.py

```
class Worker(Thread):

    def __init__(self, delay):
        Thread.__init__(self)
        self.__loop = True
        self.__r = redis.Redis(charset="utf-8", decode_responses=True)
        self.__delay = delay

    def run(self):
        while self.__loop:
            message = self.__r.brpop("queue:")
            if message:
                message_id = int(message[1])

                self.__r.hmset(f"message:{message_id}", {
                    'status': 'checking'
                })
                message = self.__r.hmget(f"message:{message_id}",
["sender_id", "consumer_id"])
                sender_id = int(message[0])
                consumer_id = int(message[1])
                self.__r.hincrby(f"user:{sender_id}", "queue", -1)
                self.__r.hincrby(f"user:{sender_id}", "checking", 1)
                time.sleep(self.__delay)
                is_spam = random.random() > 0.5
                pipeline = self.__r.pipeline(True)
                pipeline.hincrby(f"user:{sender_id}", "checking", -1)
                if is_spam:
```

```
                    sender_username = self.__r.hmget(f"user:{sender_id}",
'login')[0]
                    pipeline.zincrby("spam:", 1, f"user:{sender_username}")
                    pipeline.hmset(f"message:{message_id}", {
                        'status': 'blocked'
                    })
                    pipeline.hincrby(f"user:{sender_id}", "blocked", 1)
                    pipeline.publish('spam', f"User {sender_username} sent
spam message: \"%s\"" %
                                     self.__r.hmget("message:%s" %
message_id, ["text"])[0])
                    print(f"User {sender_username} sent spam message:
\"%s\"" % self.__r.hmget("message:%s" % message_id, ["text"])[0])
                else:
                    pipeline.hmset(f"message:{message_id}", {
                        'status': 'sent'
                    })
                    pipeline.hincrby(f"user:{sender_id}", "sent", 1)
                    pipeline.sadd(f"sentto:{consumer_id}", message_id)
                pipeline.execute()

    def stop(self):
        self.__loop = False
```

Емуляція активності користувачів:

EmulationController.py

```
class EmulationController(Thread):
    def __init__(self, username, users_list, users_count, loop_count):
        Thread.__init__(self)
        self.__loop_count = loop_count
        self.__server = RedisServer()
        self.__users_list = users_list
        self.__users_count = users_count
        self.__server.registration(username)
        self.__user_id = self.__server.sign_in(username)

    def run(self):
        while self.__loop_count > 0:
            message_text = fake.sentence(nb_words=10,
variable_nb_words=True, ext_word_list=None)
            receiver = self.__users_list[randint(0, self.__users_count -
1)]
            self.__server.create_message(message_text, receiver,
self.__user_id)
            self.__loop_count -= 1

        self.stop()

    def stop(self):
        self.__server.sign_out(self.__user_id)
        self.__loop_count = 0
```

Повний код доступний в репозиторії
https://github.com/3A43Mka/db_lab_2/tree/master/lab2

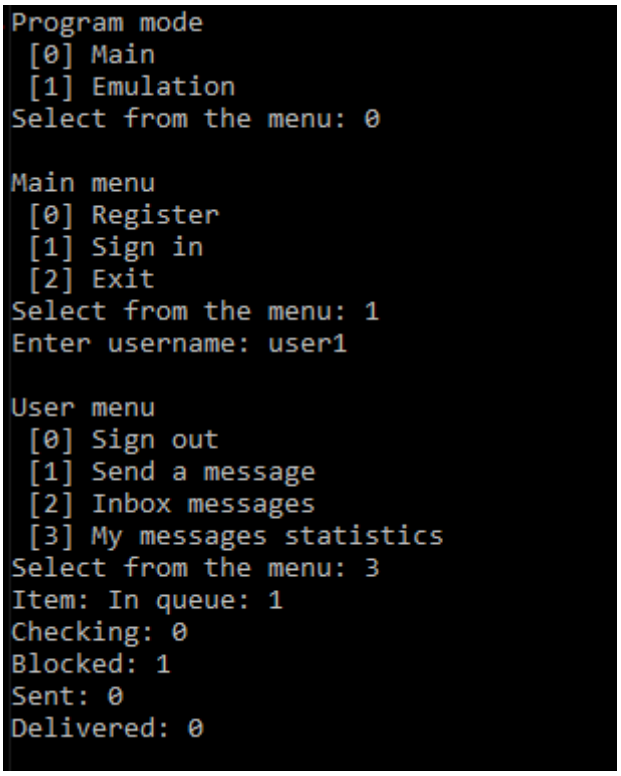

Рис. Користувач заходить в систему і дивиться свою статистику

рис. Виконувач відбирає спам в повідомленнях (псевдовипадково)

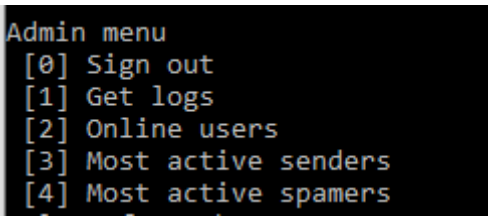

рис. Меню адміна

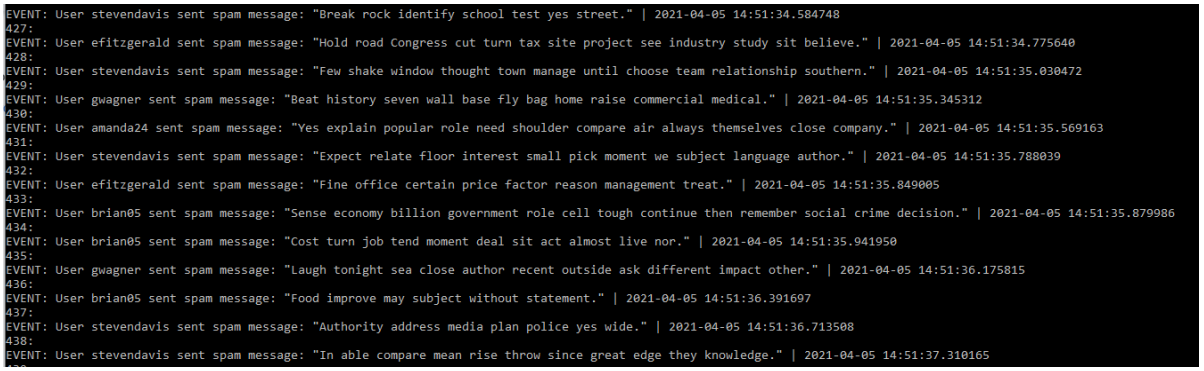

Рис. Перегляд логу подій

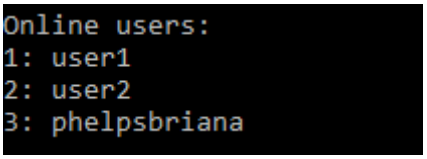

Рис. Перегляд користувачів в системі

```
Top senders:
1: ('user:parkkelly', 4958.0)
2: ('user:whitekimberly', 3870.0)
3: ('user:cmejia', 2825.0)
4: ('user:brian05', 2679.0)
5: ('user:efitzgerald', 2593.0)
6: ('user:phelpsbriana', 2580.0)
7: ('user:hdelacruz', 2297.0)
8: ('user:kweaver', 2215.0)
9: ('user:sharon84', 2178.0)
10: ('user:caldwelldana', 1169.0)
```

Рис. Перегляд найактивніших юзерів

```
Top spamers:
1: ('user:parkkelly', 2003.0)
2: ('user:whitekimberly', 1552.0)
3: ('user:cmejia', 1090.0)
4: ('user:phelpsbriana', 1055.0)
5: ('user:hdelacruz', 920.0)
6: ('user:sharon84', 915.0)
7: ('user:kweaver', 862.0)
8: ('user:caldwelldana', 468.0)
9: ('user:hornalexandra', 290.0)
10: ('user:stevendavis', 227.0)
```

Рис. Перегляд найактивніших спамерів

```
User menu
 [0] Sign out
 [1] Send a message
 [2] Inbox messages
 [3] My messages statistics
```

Рис. Меню користувача

```
1124: From: parkkelly - Late later challenge sign season security least attack believe.
1125: From: sharon84 - Behind standard develop hospital apply radio difference for impact process.
1126: From: sharon84 - Smile nature on hour country move of before own contain.
1127: From: parkkelly - Wait author serve range keep here learn effort start case.
1128: From: parkkelly - Identify available always public interesting only response your discussion American court by.
1129: From: phelpsbriana - Hear season bag between answer what lot indicate reality station animal.
1130: From: sharon84 - Enter teach some school prevent travel final forward despite each environment particular box.
1131: From: phelpsbriana - Within near can enjoy prove social who threat rise fine floor series.
1132: From: parkkelly - Public scene still for despite impact six money property part unit south by air.
1133: From: hornalexandra - Various laugh nothing story join no during wish.
1134: From: sharon84 - Step above successful which nation event data someone goal statement hotel use.
1135: From: parkkelly - Church member network until Democrat ball law break road Democrat decide carry pay.
1136: From: phelpsbriana - Wonder white through color process soldier throw reflect mission challenge hand.
1137: From: parkkelly - Other reduce direction expect look drop beyond move hand own.
1138: From: parkkelly - Condition with will population already carry more employee collection audience.
1139: From: sharon84 - Little continue air present class leg floor.
1140: From: caldwelldana - Design name management try lay around main cover keep.
1141: From: parkkelly - Lawyer herself debate seven involve data.
1142: From: caldwelldana - Everything including information board financial they high hear.
```

Рис. Перегляд отриманих повідомлень

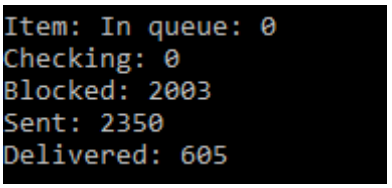

Рис. Перегляд статистики користувача

Основні переваги та недоліки Redis:

+ Дуже швидкий в роботі
+ Легкий в використанні
+ Підтримка багатьох структур даних
+ Використовує свій механізм хешування
+ Відкритий код та стабільність
- При підключенні клієнти повинні знати топологію кластера, що збільшує кількість налаштувань на клієнті
- Потребує великої кількості оперативної пам’яті через особливості реалізації
- Збільшення необхідної кількості ресурсів для роботи при масштабуванні

Висновки: при виконанні даної лабораторної роботи я здобув навички створення програм з використанням Redis та мови програмування Python.