Титульний аркуш магістерської дисертації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут телекомунікаційних систем

Кафедра Телекомунікацій

«На правах рукопису»
УДК _____

«До захисту допущено»
В.О. завідувач кафедри
_____ Явіся В.С.
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Магістерська дисертація

зі спеціальності 172 Телекомунікації та радіотехніка
(код і назва спеціальності)

на тему: Методи тестування навантаження інформаційних ресурсів Web-сервісів

Виконав: студент ІІ курсу, групи ТЗ-з83мп
Лобода Ілля Іванович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник в.о. завідувач кафедри ТК, к.т.н., доцент Явіся В.С. __ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант_____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент доцент каф. ІТМ к.т.н., доцент Правило В.В.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут
імені Ігоря Сікорського»
Інститут телекомунікаційних систем

Кафедра Телекомунікацій

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність 172 Телекомунікації та радіотехніка
(код і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Явіся В.С.___
(підпис) (ініціали, прізвище)
«___»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Лободі Іллі Івановичу

1. Тема дисертації «Методи тестування навантаження інформаційних ресурсів Web-сервісів»
науковий керівник дисертації Явіся Валерій Сергійович, доцент,
затверджені наказом по університету від «_07_» «_11_» 2019р. № _3854-с_

2. Строк подання студентом дисертації 02.12.2019

3. Об’єкт дослідження методика тестування інформаційних веб-сервісів

4. Предмет дослідження (Вихідні дані – для магістерської дисертації за освітньо-професійною програмою) - інформаційні веб-сервіси

5. Перелік завдань, які потрібно розробити

1. Провести аналіз методів тестування навантаження

2. Визначити методику тестування навантаження web-сервісів

3. Створити web-сервіс, що буде слугувати тестованою системою

4. Провести тестування навантаження згідно методології

5. Отримати висновки тестування

6. Орієнтовний перелік ілюстративного матеріалу

1. Тема, мета, об'єкт та предмет дослідження;
2. Способи надання web-сервісів
3. Методи тестування навантаження
4. Концепт тестування навантаження
5. Методологія тестування навантаження систем
6. Експериментальне використання методології
7. Висновки до роботи

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 01.10.2010

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Аналіз проблематики	01.09.2018 – 30.10.2018	
2	Аналіз методів тестування навантаження	01.11.2018 – 30.12.2018	
3	Вивчення архітектури Web-сервісів	01.01.2019 – 30.02.2019	
4	Побудова інформаційного Web-сервісу	01.03.2019 – 30.04.2019	
5	Визначення методології навантажувального тестування	01.05.2019 – 30.06.2019	
6	Проведення навантажувального тестування побудованого Web-сервісу	01.07.2019 – 30.08.2019	
7	Збір даних та їх аналіз	01.09.2019 – 30.11.2019	

Студент _____ _____

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Тема роботи: Методи тестування навантаження інформаційних ресурсів Web-сервісів

Текстова частина дипломної роботи: с.99, рис. 56, джерел 8.

Метою роботи є проведення багатостороннього аналізу методів тестування навантаження та утворення на основі цього методології щодо проведення навантажувального тестування інформаційних ресурсів веб-сервісів, а також включення його у процес розробки та експлуатації систем.

В наш час швидкість розповсюдження інформації, доступ до неї та можливість нею користуватись є дуже важливою. І при розумінні цього, дуже важливим стає такий параметр системи, як її продуктивність. Саме спираючись на цей параметр існує такий вид тестування систем, як навантажувальне тестування або ж тестування продуктивності.

Було визначено чим є інтернет ресурси та веб-сервіси і яку роль вони займають у інформаційному просторі. Було надано загальну концепцію тестування навантаження, його методи та тестові сценарії на основі цього, а також виведено методологію, спираючись на яку, можливо провести навантажувальне тестування.

Ключові слова: Навантажувальне тестування, продуктивність, веб-сервіси, методологія тестування навантаження, інформаційні ресурси.

ABSTRACT

R & D: Load testing approaches of information resources of web-services

Text part of thesis: p.99, fig. 56, sources 8.

The aim of the work is to carry out a multilateral analysis of load testing and methods based on it and then to make methodology for the testing of information resources of web services and to use it in software development cycle and supporting.

Nowadays, the speed of the dissemination of information, access to it and the ability to use it is very important. And when it comes to understanding this, the system parameter, such as its performance, becomes very important. It is based on this option that there is a kind of testing systems, such as load testing or performance testing.

It was determined what Internet resources and Web services are and what role they occupy in the information space. The general concept of load testing, its methods and test scenarios on the basis of this was provided and a methodology was derived, based on which load testing might be possible.

Keywords: Loading testing, productivity, performance, web services, load testing methodologies, information resources.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

XML - Extensible Markup Language

CSV - Comma-separated values

HTTP – Hyper Text Transfer Protocol

REST - Representational State Transfer

SOAP - Simple Object Access Protocol

ПЗ – Програмне забезпечення

CRUD – Creat Read Update Delete

RPC - Remote procedure call

AWS – Amazon Web Services

БД – База даних

SQL - Structured query language

ЦП – центральний процесор

CPU – Central processor unit

SUT – System Under Test

TS – Test System

ВСТУП

У теперішній час складно собі уявити сучасну людину, яка б не користувалась мережею інтернет. Технології невпинно покращуються та оновлюються і дуже складно знайти сферу їх використання, щоб показувала цей прогрес краще за ІТ.

Все більше і більше бізнесу переходить з оффлайн в онлайн простір. А з великою кількістю представників на ринку, збільшується і прискіпливість споживачі до наданих послуг чи товару.

Одним з найважливіших критеріїв відбору сервісу серед інших в мережі інтернет є швидкість «надання послуги». Але одне діло – надавати послуги декільком користувачам і зовсім інше – мільйонам. Звичайний домашній сайт-візитка не буде спроможним «вистояти» перед кількістю користувачів, для якої він не був побудований, тому необхідно завжди розуміти свою цільову аудиторію та кількість користувачів, що будуть використовувати сервіс. І саме в цей момент нам потрібне тестування навантаження.

Ми кожного дня використовуємо пошукові сайти, інтернет магазини, сайти з прогнозом погоди та новинами, тощо. Проте мало хто з нас знає, як насправді все працює в мережі. За кожним великим сайтом криється величезна кількість веб-сервісів, що допомагають йому виконувати роботу якнайкраще.

Сайт може отримувати інформацію про погоду в місті з одного веб-сервісу, ситуацію на дорогах і новини у світі – зовсім з інших. І кожен цей веб-сервіс може мати свою архітектуру та бути розрахованим на зовсім різну кількість користувачів.

Якщо ви хочете, щоб ваш бізнес розвивався – треба постійно підлаштовуватись до ринку і надавати послуги краще за інших. Саме тому і почали використовувати тестування навантаження, що має безліч методів та підходів.

Саме тому темою даної роботи було обрано «Методи тестування навантаження інформаційних ресурсів веб-сервісів».

Актуальність теми полягає в тому, що все більше і більше користувачів задумуються про швидкість та якість надання послуг сервісом. Якщо сторінка сайту не буде відповідати або відкриватись більше ніж за 2 секунди, вірогідність того, що клієнт піде до конкуретна значно виростає. Ось чому продуктивність сайту та його інфраструктури так важлива. Саме тому потрібно правильно та швидко її тестувати, аналізувати і в подальшому – покращувати. Адже від цього залежить цінність даного сервісу.

Предметом дослідження даної наукової роботи є інформаційні веб-сервіси.

Об’єктом дослідження є методика тестування інформаційних веб-сервісів.

Новизна даної роботи полягає в тому, що тема тестування навантаження веб-сервісів в наш час майже не вивчається, метою даної роботи є збір та аналіз загальноприйнятих методів тестування навантаження, а на їх основі визначення методики навантажувального тестування веб-сервісів.

Можливими галузями застосування результатів цієї роботи є використання їх у розробці ПЗ, мережевих технології в схемотехніці, тощо.

1 ОСОБЛИВОСТІ НАДАННЯ WEB-СЕРВІСІВ

1.1 Визначення явища «інформаційний ресурс» та веб-сервіс

Згідно з Законом України «Про Національну програму інформатизації», інформаційний ресурс – це сукупність документів у інформаційних системах (бібліотеках, архівах, банках даних тощо).

Веб-сервіси – це такі програмні компоненти, чиї інтерфейси підтримуються комп'ютером і чий формат є придатним для їх обробки. Ці інтерфейси взаємодіють між собою або з іншою системою за допомогою HTTP-протоколу. Для всіх веб-сервісів є характерним їх URI адреса, де його функціональність абстрактно представляється за допомогою того самого інтерфейсу, а реалізація сервісу забезпечує відправку і прийняття повідомлень при взаємодії з сервісом. [1]

Сайт інтернет-магазину взаємодіє з базою на складі, щоб показувати кількість доступного товару саме за допомогою веб-сервісів.

Існують такі основні «стандарти» для веб-сервісів:

- SOAP (Simple Object Access Protocol)
- REST (Representational State Transfer)

SOAP є більш складною моделлю при використанні у веб-сервісах і її використовують у більш «важких» архітектурах, де взаємодія між системами неможлива з використанням лише теорії CRUD (Create Read Update Delete), але в тих системах, де ця теорія не відчувається недостатньою – дуже добре підходить архітектура REST, що реалізована доволі легко. При використанні в системі протоколу SOAP сервісу, що користується послугами цієї системи, необхідно знати про «внутрішні» методи цієї системи, аби з ними взаємодіяти. Це, в свою чергу, дозволяє реалізувати за допомогою SOAP більш складну архітектуру та інфраструктуру.[4]

З веб-сервісом можна взаємодіяти використовуючи такі «основні» сценарії:

- однонаправлений запит – повідомлення надсилається до сервісу або до сервісів, які в свою чергу не відповідають на нього;

- синхронний «запит-відповідь» - сервіс-клієнт та сервіс-надавач послуг стають пов’язаними, причому відбувається синхронний обмін повідомленнями між ними, коли клієнт стає заблокованим, на час, поки не отримає відповіді від сервісу;
- асинхронний «запит-відповідь» - в цьому випадку клієнт не блокується після надсилання повідомлення до сервісу і здатен виконувати інші дії, поки в цей час його повідомлення піддається обробці сервісу;
- RPC-виклик – клієнт викликає процедуру інтерфейсу веб-сервіса шляхом серіалізації параметрів процедур у повідомленні і посилки його до сервісу;
- помилки – певний метод або функція/процедура на стороні сервісу закінчується помилкою і про це повідомляється клієнту;
- запит з підтвердженням – після того, як запит був успішно прийнятий або ж в процесі прийняття повідомлення сталась помилка – клієнт про це повідомляється;
- передача через посередників – тут запит не потрапляє до веб-сервісу напряму. Клієнт відсилає його через деяких посередників. При цьому можлива маршрутизація та трекінг;
- кешування – якщо посилання клієнту такого роду вже відсилались, то наступного разу вони відправляються з кешу;
- діалог – з повідомлень збираються певні групи, що являють собою логічні набори, які зберігають свій певний стан;
- додаткова функціональність – обмін повідомленнями проходить з додатковими вимогами, такими як кодування, аутентифікація, цілісність, транзакція тощо. [7]

1.2 Концепт тестування навантаження та його актуальність

В наш час від сайтів і сервісів у мережі інтернет залежать великі гроші, а інколи й життя. Технології швидко розвиваються, сайти стають інтерактивними і все частіше і частіше відходять від концепту сайту-візитки.

Вони вже представляють собою повноцінні додатки, програми та величезну інфраструктуру. І робота всіх цих сервісів повинна не мати збоїв та помилок. Дуже часто «функціональне» тестування (тестування основних функцій сайту), тестування локалізації чи дизайну стає недостатнім. В умовах ринкової економіки якість випущеного продукту повинна бути якомога краща і компанії шукали шляхи вирішення цієї проблеми. Сайт чи додаток може прекрасно працювати, коли ним користується одна людина, але зовсім інша річ – мільйони людей, мільйони запитів. Саме це і зробило тестування навантаження настільки актуальним в реаліях нашого часу.[7]

Отже, тестування навантаження або тестування продуктивності – це тестування, яке проводиться для певної системи чи частини цієї системи для перевірки швидкості її роботи під певним навантаженням. Також в процесі цього тестування можливо визначити такі характеристики системи, як масштабування, надійність та використання ресурсів.

В деяких англомовних ресурсах тестування продуктивності та навантаження можуть розділяти, але мають спільну мету та означення, тому ці значення часто змішуються. В цій роботі обидва ці методи будуть вважатись за один. [3]

Існують такі основні методи навантажувального тестування, як:

- тестування навантаження (Load testing);
- тестування стабільності (Stability/Endurance testing);
- стресове тестування (Stress testing);
- шип тестування (Spike testing).

Також виділяють такі неосновні види навантажувального тестування, як:

- об’ємне тестування (Volume testing);
- тестування масштабованості (Scalability testing). [6]

Метою навантажувального тестування є не пошук помилок, а знаходження та усунення вузьких місць в системі, що тестується та встановлення базової лінії для майбутніх тестів регресії. Провести

випробування продуктивності - це означає здійснити ретельно контрольований процес вимірювання та аналізу. В ідеалі, тестоване програмне забезпечення вже достатньо стабільне, щоб цей процес проходив плавно.

Для будь-якої складної системи та застосування зокрема, дуже важливим є налагодження постійного та раннього тестування (тестування продуктивності). Мається на увазі тестування на ранніх етапах розробки. Це дозволяє знайти проблеми якомога раніше розробки та швидко їх виправляти. Також за допомогою раннього тестування та виявлення проблеми, стає можливим змінити концепцію в розробці та використати інші методи корегування системи з найменшими витратами на розробку та підтримку продукту, що складають великий відсоток від загальних витрат.

Зазвичай тестування продуктивності являє собою набір тестових сценаріїв, що запускають для створення певного навантаження на певну систему. Це роблять за допомогою використання певного навантаження інжекторними машинами, які запускають окремі екземпляри тестових сценаріїв. Але в наш час багато інструментів тестування продуктивності дозволяють створювати певний вплив на систему за допомогою навантаження за допомогою лише однієї машини. Ці інструменти дозволяють запустити тести «розподіленими» і координувати їх та імітувати певну кількість користувачів, використовуючи лише одну інжекторну машину. [7]

Висновок до розділу 1

У першому розділі було визначено, що інформаційні ресурси та веб-сервіси є важливою складовою мережі інтернет. Інформаційним ресурсом є сукупність документів, файлів та інформації загалом у інформаційних системах. А веб-сервісами називають загальні програмні компоненти, чиї інтерфейси підтримуються комп’ютером та іншими сервісами. Вони слугують призначені для спілкування програм з програмами. Тестування навантаження є окремим видом тестування систем, що використовує

навантаження на систему як певний вплив. Цей вид тестування є дуже важливим, хоча він і не відноситься до функціональних видів тестування, адже за допомогою нього можна визначити загальні характеристики системної продуктивності та здатність системи реагувати на навантаження. [7]

2 МЕТОДИ ТЕСТУВАННЯ НАВАНТАЖЕННЯ

2.1 Тестування навантаження

Не всі можуть мати час або ресурси для тестування кожної функції сайту, але розуміння функціональності веб-сайту буде корисним для з'ясування того, які частини сайту дійсно потрібно перевірити на завантаження. Розробити тестові сценарії найбільш простими є дуже хорошою практикою. Теоретично можна завантажити інструмент тестування навантаження і записати тестовий випадок, коли користувач потрапив у кожну окрему сторінку, а потім створить тест навантаження для декількох користувачів, які виконують те ж саме. Проте це не особливо корисно, ані реалістично для більшості веб-сайтів. Дуже гарною концепцією є використання теорії Парето, що визначає 80% навантаження, що впливають на систему як таке навантаження, що генерується лише на 20% сторінках або ж 20% коду цієї системи

Зазвичай навантаження на систему подається протягом 4-8 годин. В цей час збираються такі характеристики продуктивності як: кількість повідомлень в секунду, транзакцій в секунду, час відповіді від сервера, відсоток помилок у відповідях, утилізація апаратних ресурсів тощо. Зібрані метрики проходять перевірку на відповідність заданим вимогам. В результаті можливою є відповідь на питання: чи система відповідає заданим вимогам щодо працездатності?

Зазвичай під час навантажувального тестування використовують навантаження в кількості 80% від результату максимальної продуктивності, виявленої при стрес-тесті.

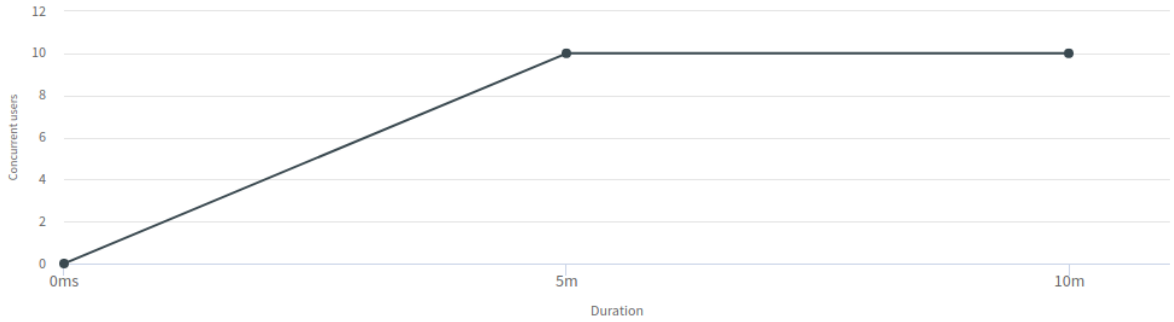

Рисунок **Error! No text of specified style in document.**.1 – графік тестування навантаження в залежності від заданого навантаження та часу

На виході тестування маємо локалізацію вузьких місць в продуктивності програми та дефектів, докладне профілювання всіх компонентів системи і утилізацію апаратних ресурсів під корисним навантаженням. [7]

2.2 Тестування стабільності

Тестування стабільності - це процес тестування ПЗ, систем, сервісів тощо, чиє призначення у можливостей тестованої системи виконувати необхідні їй функції в заданих умовах протягом певного значного періоду часу або для певної кількості операцій. Тобто це випробування, що визначає наскільки система є стабільною.

Найкращим результатом для даного виду тестування є відсутність витоків пам'яті, несподіваних помилок у програмі або системі, знищення даних, перезавантаження серверів під навантаженням та інші аспекти, що певним чином діють на стабільність. Ці речі часто неможливо помітити протягом інших видів тестування, адже цей тип тестування використовує час.

Одним із синонімів до «стабільності» в контексті цього тестування є «витривалість». Адже під час проведення цього тестування спостерігач може об’єктивно оцінити, наскільки тестована система є витривалою.[7]

Отже тестування стабільності визначається, як така методологія тестування програмного забезпечення, що прийнята для перевірки, чи система може працювати безперервно, в межах або трохи вище прийнятного

періоду часу не виходячи за певні визначені (на початку проектування цієї системи або перед проведенням тестування) норми роботи.[6]

Кілька статистичних даних збираються та вимірюються під час випробувань стабільності; ці цифри аналізуються спеціалістами, щоб створити звіт та визначити можливі проблеми продуктивності.

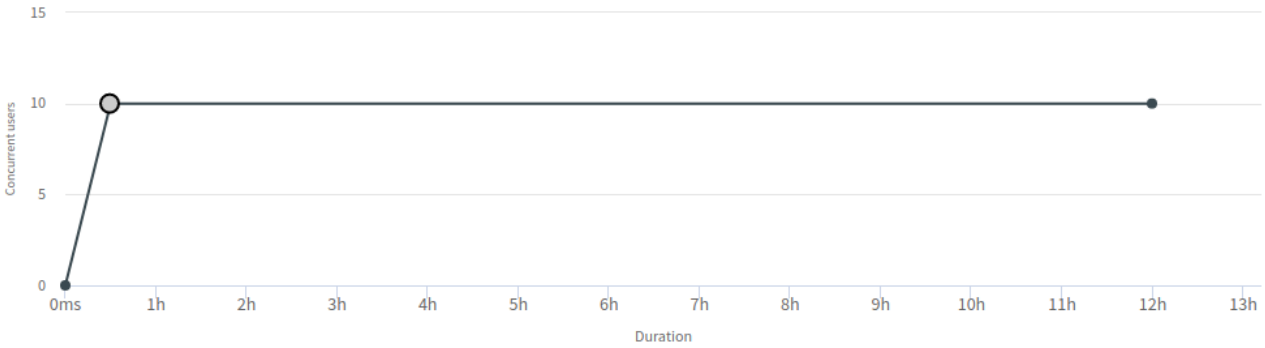

Рисунок **Error! No text of specified style in document.**.2– процес тестування стабільності в залежності від заданого навантаження та часу

2.3 Стресове тестування

Стрес-тестування - це процес визначення чи здатен комп'ютер, мережа чи програми або ж певна система підтримувати певний визначений рівень ефективності в несприятливих умовах. Під час проведення цього тестування, можливе використання певних лабораторних умов для подальшого проведення певних кількісних випробувань, таких як вимірювання частоти помилок або ж аварійних ситуацій в системи. Визначення цього методу тестування також відноситься до оцінки певних аспектів, як стійкість до атак (спрямованих на відмову в обслуговуванні) та доступність системи в цілому. [6]

Під час проведення стрес-тесту навмисно створюється та підтримується несприятлива обстановка. Це може бути:

- Запуск декількох ресурсомістких програм на одному комп'ютері одночасно
- Спроба зламати комп'ютер і використовувати його як зомбі для розповсюдження спаму
- Закидання серверу спамом електронної пошти

- Створення численних одночасних спроб доступу до одного веб-сайту
- Спроби зараження системи вірусами, троянами, шпигунськими програмами та іншими шкідливими програмами.[7]

Як вже було вказано, проведення стрес-тестування дає змогу визначити, наскільки додаток та система працездатні в умовах, що є більш несприятливими для системи, ніж звичайні умови експлуатації. Також можливою стає оцінка регенерації системи та здатності її до відновлення що визначає, наскільки швидко і як повно система повертається до нормального свого стану після стресу.

Також однією з цілей стрес-тестування є визначення, наскільки деградує система, тобто змін її продуктивності, а це прямим чином корелюється з визначенням тестування продуктивності.

Стрес-тестування, з погляду тестування навантаження – це процес вимірювання та аналізу системи під впливом великої кількості запитів користувачів, використання великої кількості даних, тощо. Тобто ціллю цього тестування є визначення змін у продуктивності системи під впливом навантаження, на яке система не була розрахована. [7]

Особливістю даного процесу є визначення певної «критичної точки», коли застосування чи система вже не здатні працювати з очікуваною ефективністю.

Також метою цього тестування є вимірювання здатності системи до регенерації після критичних помилок та здатність виводити технічні повідомлення про свій стан під час їх виникнення.

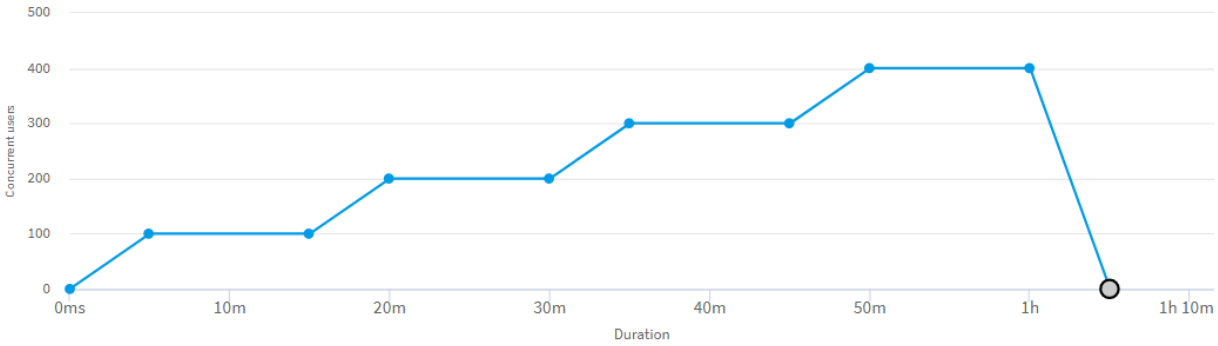

Рисунок **Error! No text of specified style in document.**.3– процес стрес-тестування в залежності від заданого навантаження та часу

Як видно з Рис. 2.3 стрес-тестування проводиться кроками. Кожен крок повинен закінчуватись або певною стабілізацією системи або ж її деградацією. Такий метод дозволяє аналізувати реакцію системи на певне екстремальне навантаження протягом певного часу і визначення її максимальної деградації на цьому етапі. [7]

2.4 Шип-тестування

Назва цього виду тестування походить від англійського слово “Spike”, що в перекладі означає «шип». Це пов’язано з тим, що на графіку використання навантаження відносно часу, в процесі цього тестування з’являється «шип». Тобто збільшення навантаження до певного високого значення та опущення до малого значення за певний короткий проміжок часу.

В наш час дуже складно спрогнозувати, яке навантаження, в який час і в якій кількості припаде на певну систему, сайт тощо. Тому цей вид тестування є дуже важливим, адже від пов’язаний з короткотривалим «викидом» понаднормового навантаження в систему. [7]

В реальності такий профіль навантаження може бути спричиненим відмовою сайту чи системи в «бізнес години» і авторизації великої кількості користувачів після введення її в експлуатацію, це може бути запуск нової моделі смартфону у відомого бренду, навчальної пожежної тривоги на підприємстві і навіть великої кількості повідомлень в соціальній мережі у Новий Рік.

Кожна програмна система має заздалегідь визначене максимальне навантаження. Впродовж шип-тестування система підлягається впливу максимального навантаження (або навіть навантаження, більшого за максимальне) за короткий інтервал часу і раптове зменшення навантаження на систему до прийнятного для неї робочого значення. Неприйнятною для системи поведінкою є отримання користувачем повідомлення про помилку, погіршення загальної працездатності системи, припинення системи

генерувати відповіді для інших сервісів та користувачів або ж зупинка її роботи.

Іншою, не менш важливою метою шип-тестування є визначення часу яке система витрачає на регенерацію та повернення до початкових умов роботи. [7]

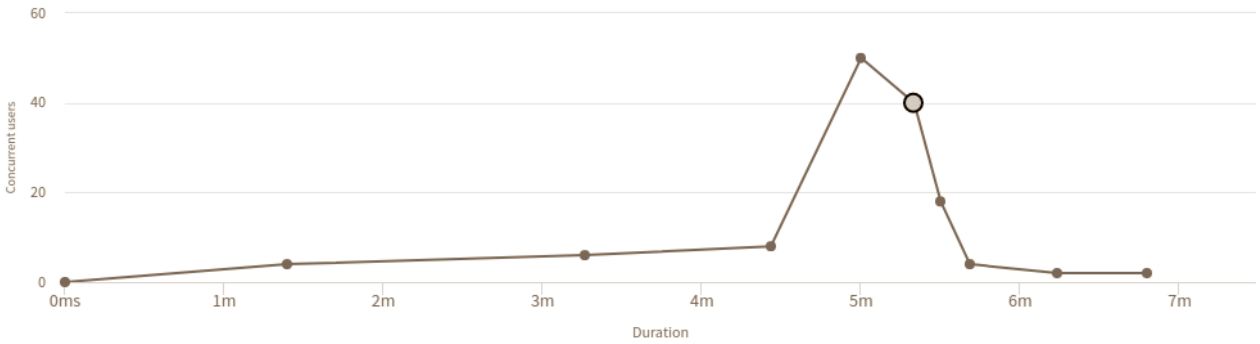

Рисунок **Error! No text of specified style in document.**.4– процес шип-тестування в залежності від заданого навантаження та часу

Як видно з рисунку 2.4, під час шип тестування викид навантаження займає незначний час, відносно всього тесту, проте саме навантаження збільшується в рази від робочого навантаження (10 разів і більше). [2]

2.5 Об’ємне тестування

Об'ємне тестування – це тестування, де система піддається впливу великих обсягів даних.

Наприклад, якщо потрібно протестувати програму з певним розміром бази даних, то необхідно збільшити базу даних до заданого розміру, занести в цю базу певні тестові дані та виконати тестування продуктивності на цій системі.

Насправді, під час об’ємного тестування нас цікавлять ті ж показники, що і при тестуванні продуктивності, а це в основному - час виконання операцій залежно від кількості користувачів, тільки в цьому випадку ми проводимо тестування на великих обсягах даних.

Найперші претенденти на об'ємне тестування, це ті місця, де відбувається вибір великих об'ємів даних з БД або виконуються складні sql запити для вибірки даних

Для проведення об’ємного тестування потрібно:

- обрати архітектуру, що задовольняє заданим вимогам;
- виявити місця, де вибираються великі обсяги даних з БД;
- виявити місця, де виконуються складні sql запити;
- заповнити БД великою кількістю даних;
- виконати тестування. [7]

2.5 Тестування масштабованості

Тестування масштабованості - це тестування навантаження, яке зосереджується на розумінні того, як додаток масштабується, коли він розгортається в більших системах або під час того, коли на нього впливає більше навантаження в цих системах. Мета даного тестування полягає в тому, щоб зрозуміти, в який момент застосування припиняє масштабування та визначати причини цього. Таке тестування масштабованості можна розглядати як своєрідне тестування продуктивності з його певними особливостями та метою.

Хорошим початком для проведення цього тестування є використання комфортного для системи профілю навантаження.[2]

Ідея тестування масштабованості полягає в тому, щоб визначити, в якій точці програма припиняє масштабування і визначає причину, що лежить в основі цього.

З кожним кроком навантаження повинно невпинно збільшуватись, оскільки кожен етап тестує свій певний аспект. Малі навантаження забезпечують функціонування певної системи на базовому, прийнятному рівні. Тест середніх навантажень забезпечує функціонування сервісу чи додатку на його очікуваному рівні. Тестування системи з високим навантаженням перевіряє можливість системи працювати з навантаженнями, більшими за прийнятне.[7]

Тест на масштабованість може дати відповіді на наступні типові питання:

- Як зміни в апаратному та програмному забезпеченні впливають на продуктивність сервера?
- Чи можна підвищити продуктивність програми шляхом модернізації серверного обладнання? [5]

Висновки до розділу 2

Як було визначено у другому розділі – тестування навантаження є дуже важливим напрямком у тестуванні систем та є одним із способів підвищення якості системи. В цьому розділі було наведено основні абстрактні методи тестування навантаження, основна відмінність яких є їх профілі навантаження. Кожен із методів є незалежним та може використовуватись як окремо, так і в сукупності з іншими, адже кожен метод використовується для досягнення окремих цілей та визначення своїх, конкретних метрик. Проте в загальному розумінні, тестування навантаженням перевіряє, наскільки система продуктивна при тих чи інших навантаженнях.[7]

3 МЕТОДОЛОГІЯ ТЕСТУВАННЯ НАВАНТАЖЕННЯ WEB-СЕРВІСІВ

Як вже було сказано в роботі – в наш час інтернет сайти або застосунки можуть отримувати інформацію не з одного, а з великої кількості веб-сервісів. Саме такі, більш складні системи зазвичай фігурують на ринку та є фундаментом побудови середніх та складних систем.

Такі системи називають мультисервісними. Саме через їх розповсюдженість, методологія тестування навантаження в даному розділі буде наводитись для випадку мультисервісної архітектури, а потім буде перевірена для більш простої реалізації веб-сервісу.

3.1 Мультисервісні системи

Перш ніж обговорювати тестування продуктивності мультисервісних систем, необхідно дати визначення мультисервісної системи. Типова мультисервісна система пропонує ряд послуг, які можна отримати за рахунок доступу до неї через точки входу. У цьому контексті сервіс працює на сервісній платформі, яка дозволяє організаціям і приватним особам обслуговувати і споживати контент за допомогою мережі інтернет. Мультисервісні системи пропонують різні види і рівні послуг.

Ця система складається з безлічі підсистем (апаратних і програмних), що зазвичай взаємодіють за допомогою більш ніж одного протоколу. Підсистеми надають різні функції і хост послуги або виступають посередником для їх доступу і адміністрування. Споживачі послуг є кінцевими користувачами які отримують доступ до послуг через сумісні пристрої, які називаються користувацьким обладнанням або периферійними пристроями. Споживання послуг реалізується за допомогою протоколів зв'язку, що включають в себе різні типи транзакцій, наприклад, аутентифікацію, тощо. З урахуванням специфікації IMS (IP Multimedia Systems), існуюча телекомунікаційна інфраструктура швидко рухається до загального підходу для створення, розгортання та управління послугами;

тому очікується, що незабаром в багатьох протестованих системах буде доступна велика різноманітність послуг.[8]

Крім цього аспекту різноманітності послуг, видно ще два аспекти: складність послуги та випадковість попиту на послуги. Сервіси стають все більш складними, вимагаючи більше обчислювальних ресурсів на стороні системи та більше обміну повідомленнями між залученими компонентами. Очікується, що чим складніша і затребувана послуга, тим більше знижується продуктивність системи. Випадковість попиту на послуги стосується налаштувань користувача для сервісів та описує спосіб вибору користувачем варіанту зі списку доступних послуг. Як у результаті - загальне завантаження системи - це сукупність екземплярів різних служб, де кожен вид послуг робить внесок лише у незначній частці.

Служби розміщуються в мережі компонентів, де кожен компонент має чітко визначений функціонал, наприклад, аутентифікацію, реєстрацію, зарядку. Щоб отримати більше продуктивності, деякі компоненти можуть бути навіть розподілені на декількох серверах. Коли послугу викликає користувач, ці компоненти взаємодіють один з одним за допомогою обміну повідомленнями. Ця комунікація вимагає хорошої синхронізації між компонентами та надійного механізму обробки стану обслуговування, що здатний працювати при великих навантаженнях. Серед відомих питань, що виникають при оцінці працездатності складної розподіленої системи, слід враховувати, що не всі програмні компоненти є доступними. Через складність існуючих систем очікується, що деякі компоненти є сторонніми компонентами, що належать третім особам. Тому під час тестування працездатності може бути виявлено, що ці компоненти не взаємодіють належним чином з рештою системи, або що деякі з них є вузькими місцями для всієї системи. Однак виклик залишається тим самим: потрібно зрозуміти продуктивність всієї системи і, врешті-решт, зуміти визначити який компонент є вузьким місцем для працездатності системи. [8]

Послуги можуть бути доступні в двох або більше мережах. Ці мережі можуть з'єднуватися один з одним таким чином, що користувачі з однієї мережі можуть спілкуватися з користувачами з інших мереж.

Цей аспект слід також враховувати при розробці тестів на ефективність. Тестована система повинна вміти розподіляти навантаження в адекватних пропорціях також і на послуги із закордонних мереж. Однак у цій конфігурації тестована система необхідна для імітації користувачів, які підключаються до всіх мереж.

Ще одна характеристика полягає в тому, що різні користувачі демонструють різні навігаційні шляхи для однієї послуги. Зазвичай користувачі дотримуються шляху взаємодії для успішного запиту, але можуть також проходити шляхи відмови, наприклад, користувач недоступний, покинуті запити, тощо. Залежно від послуги, зв'язок між двома сутностями різниться за складністю. Чим складніша взаємодія служби, тим більш можливий негативний сценарій.[8]

3.2 Перевикористання функціональних тестів

Одним із добре відомих методів тестування є метод функціонального тестування. Функціональне тестування систем використовується для перевірки відповідності послуг функціональним вимогам, тобто, що служба функціонально правильна.

Поряд із цією тезою тестування працездатності розглядається як розширення до функціонального тестування для перевірки вимог щодо продуктивності. Основна ідея полягає у визначенні ефективності тестів шляхом повторного використання основних функціональних можливостей тестів для служб. Тестові компоненти використовуються для імітації клієнтів сервісу. Ці компоненти виконують основні функціональні тести для оцінки реакції служб на їх запити.

Поєднання компонентів тестів, що виконують різні основні функціональні випробування, паралельно приводять до різних тестових сценаріїв для сервісів. Параметризація цієї тестової основи дає можливість

гнучкості тестової установки з різними функціональними та продуктивними навантаженнями.[8]

3.3 Дизайн процес навантажувального тестування

Для тестування мультисервісних систем виникає потреба у більш загальній методології яка зосереджена на відносинах користувач-сервіс. Методика повинна стосуватися обох сторін проблеми: аспекти функціональності, наприклад, поведінка користувача та нефункціональність, наприклад, паралельність, якість сервісу.

Для вирішення цих аспектів було введено ряд нових концепцій. Методологія проектування тестів продуктивності базується на процесі проектування тесту на ефективність, який зображено на рисунку 3.1

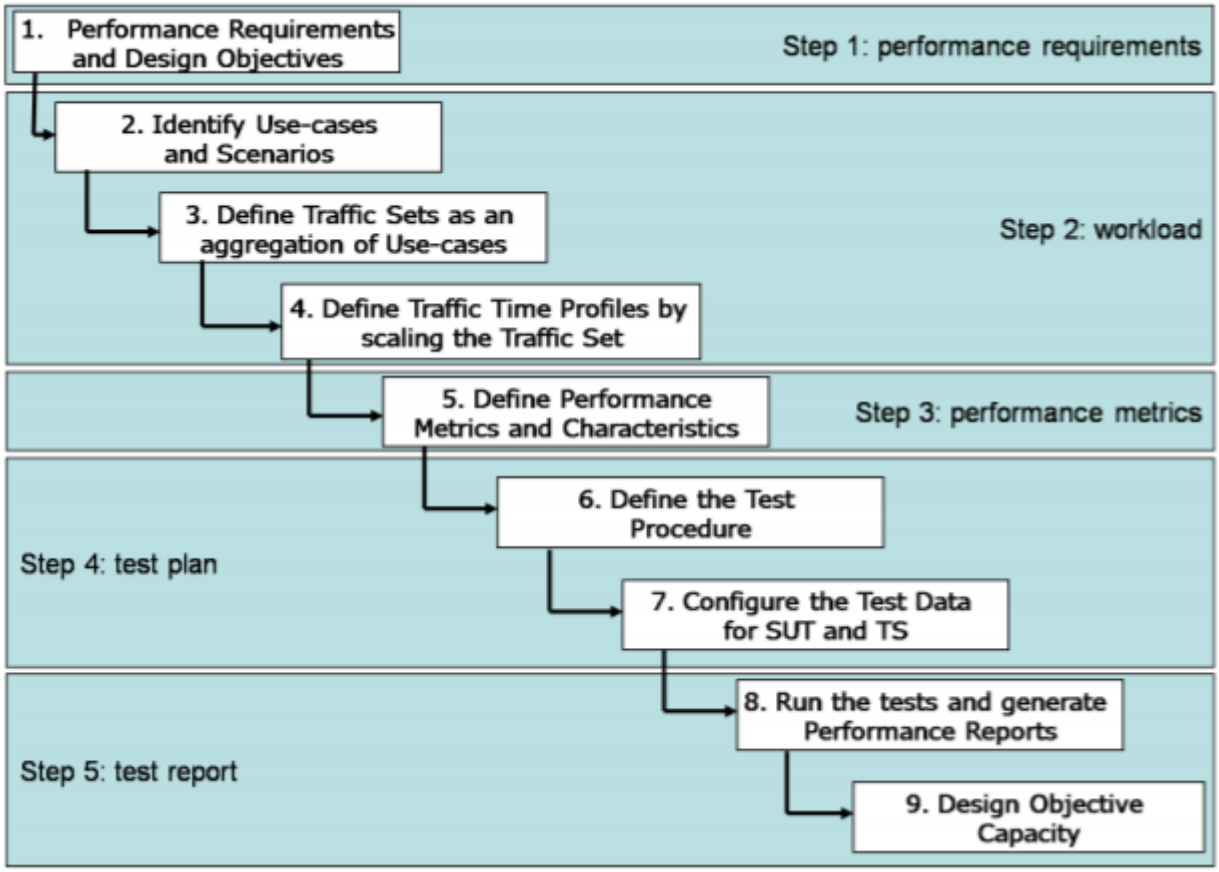

Рисунок **Error! No text of specified style in document.**.5– процес дизайну тестування навантаження

Вимоги до продуктивності. Процес починається з вибору вимог до продуктивності. На цьому кроці інженер-тестувальник повинен визначити, які особливості характеризують ефективність роботи що тестується. Висока кількість транзакцій за одиницю часу, швидкий час відгуку, низька

швидкість помилок в умовах навантаження - приклади загальних вимог до ефективності. Але є й інші конкретні вимоги, які, можливо, враховуються для системи, зокрема: доступність, масштабованість або утилізація.

Навантаження. Другий крок - визначення навантаження. Навантаження складається з користувацьких випадків (use-case) які визначають типи взаємодій між користувачами та тестованою системою для доступу до служб, наприклад, голосового дзвінка, конференц-дзвінка, запиту програми. Оскільки мультисервісна система надає багато послуг для кожного користувацького випадку, робоче навантаження створюється як композиція з декількох тестових сценаріїв для кожної моделі запиту.

Це завдання вимагає систематичного пошуку через специфікаційні документи тестованої системи, щоб розпізнати всі можливі моделі взаємодії. Поняття, яке розширює попередній процес тестування продуктивності - це набір трафіку, який дає можливість параметризувати трафік, компонувати його замість того, щоб мати фіксованим. Таким чином, можна визначити кілька наборів трафіку, наприклад, в місті, області, на рівні країни з урахуванням соціальних та технологічних факторів. Навантаження також пов'язане із заздалегідь визначеним профілем часу-трафіку, який поєднує набори трафіку з існуючим навантаженням, наприклад, швидкість прибуття випадкових дзвінків.[8]

Показники ефективності. На третьому кроці потрібно визначити показники ефективності. В залежності від виконуваної роботи, для перевірки працездатності багатосервісних систем необхідно зібрати інформацію з кожного сценарію. Тому розглядаються два типи показників: глобальні показники схожі, наприклад, CPU, пам'ять, показник швидкості відмов, пропускна здатність та пов'язані зі сценарієм показники, визначені для кожного сценарію, наприклад, коефіцієнт відмов, затримка. Список показників ефективності повинен включати ключові показники ефективності які мають значення для загальної оцінки.

Тест план. На четвертому кроці тестована система та система тестування параметризуються за допомогою первинних параметрів забезпечення, наприклад, користувачів. Тести на ефективність виконуються відповідно до процедури навантажувального тесту, який враховує ефекти тривалості запусків, що використовують багато служб.

Результат тестування. Останнім кроком є виконання тестів відповідно до плану тестування та створення звітів із змістовними даними, наприклад, найбільш стійким навантаженням, кількістю підтримуваних користувачів, тощо. [8]

3. 4 Концепти навантажувального тестування

Елементи перевірки працездатності представлені на рисунку 3.2. Поняття згруповані в три основні блоки:

• користувацькі випадки - випадки використання описують типи поведінки окремого користувача та які по черзі визначають сценарії.

• тести на ефективність - у тесті ефективності використовуються сценарії з різних користувацьких випадків, які створює навантаження роботою, агрегуючи поведінку окремих сценаріїв у контрольований спосіб і збирає лог файли під час тесту.

• звіти про випробування результативності - наприкінці тесту створюється звіт про випробування з метою звітування показників, що корелюються з лог файлами.

Сценарій визначається як потік повідомлень між взаємодіючими сторонами. Для кожного сценарію визначаються показники для вимірювання та його цілі проектування (DOs).

Тест на ефективність поєднує сценарії з різних користувацьких випадків у набір трафіку. У межах набору трафіку, кожен тип сценарію інстанціюється у заздалегідь визначеній пропорції, інтерпретується як його вірогідність виникнення під час виконання процедури перевірки працездатності. Ці пропорції вказані в профілі навантаження.

Звіт про перевірку ефективності формується після виконання, і він містить повний опис конфігурацій тестованої системи, конфігурацію системи тестування, процес, що використовується для створення навантажень системи на кожному опорному пункті тестованої системи та серії даних, що повідомляють про показники як функцію часу.[8]

3.4.1 Випадки використання та сценарії

Перший крок у створенні робочого навантаження - визначити випадки використання та вибрати для кожного випадку використання кілька тестових сценаріїв. Випадок використання зазвичай асоціюється з однією службою, однак, він може бути визначати також як склад послуг. Випадки використання визначають моделі взаємодії між одним або більше

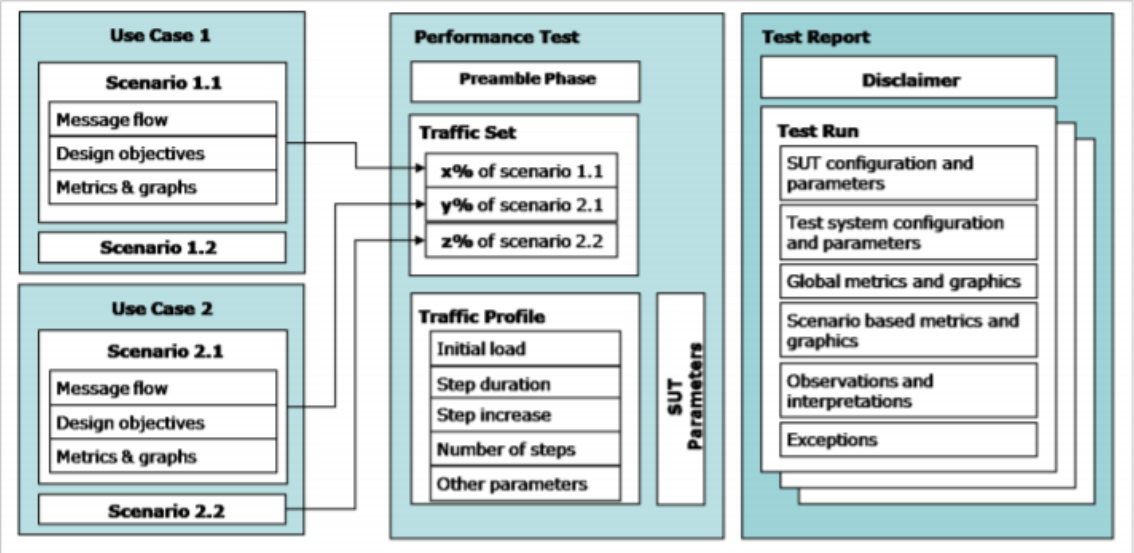

Рисунок **Error! No text of specified style in document.**.6– концепт тестування навантаження

користувачів та тестовану систему, наприклад, у сфері телекомунікацій, є такі послуги як голосовий дзвінок, конференційний дзвінок. На відміну від тестового сценарію, випадок використання - це специфікація загального типу взаємодії між системою тестування та тестованою системою, що відповідає режиму поведінки кінцевого користувача.

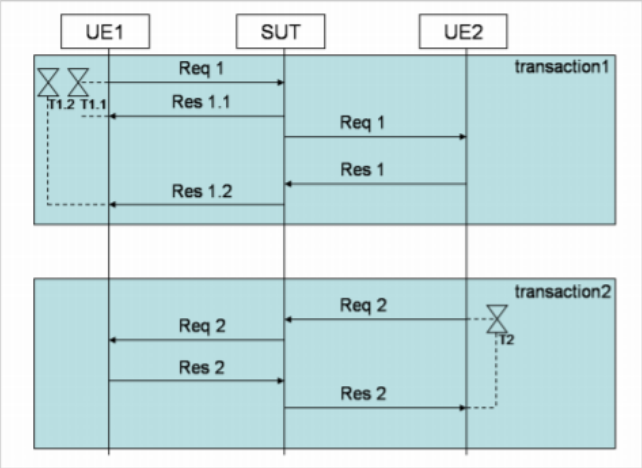

Рисунок **Error! No text of specified style in document.**.7– процес тестування навантаження

Індивідуальний шлях взаємодії називається тестовим сценарієм і описує можливу взаємодію, що визначається поведінкою користувача та інших учасників системи. Типові питання, на які потрібно відповісти в цей момент: Які послуги найбільше використовуватимуться? Який конкретно перебіг характерний для заданого сценарію? Що, швидше за все, буде проблемою?

Приклад тестового сценарію зображений на малюнку 3.3. У цьому прикладі представлена взаємодія між двома UE та SUT. Усі сутності SUT представлені у вигляді одного вікна. Взаємодія заснована на транзакціях запит / відповідь. На кожен запит є щонайменше одна відповідь.

Потік, представлений на малюнку, складається з двох транзакцій. Першу транзакцію ініціює UE1, який надсилає повідомлення запиту Req1 до SUT. На цю транзакцію є дві відповіді. Перша відповідь надсилається SUT, а друга - UE2. Друга транзакція створюється UE2 через довільний проміжок часу, спожитий після першої транзакції. У цій транзакції UE2 посилає Req2, на який відповідає UE1, повідомленням Res2.

Кожна відповідь повинна бути отримана протягом певного часу. Цей час моделюється як таймер, який вимірює витрачений час між запитом та відповіддю. Якщо відповідь не отримана протягом цього часу - час транзакція стикається з помилкою.

Найбільш репрезентативне навантаження реалізується, коли вибір тестових сценаріїв охоплює всі можливі потоки взаємодії, включаючи позитивні і негативні перебіги. Вибрані тестові сценарії повинні фіксувати типові ситуації, такі як успіх або невдачу. Щоб досягти хорошого висвітлення тестових сценаріїв, треба розглянути наступні типи сценаріїв.

- вдалі сценарії - подібний сценарій трапляється більшу частину часу і повинен завжди бути включеним до списку сценаріїв кожного визначеного випадку використання. Голосовий дзвінок називається успішним, коли всі транзакції виконуються без будь-яких станів помилок, викликаних, наприклад, тайм-аутами, неправильним вмістом або ідентифікаторами тощо.
- сценарії відмов - багато причин можуть призвести до відмови служби. Для прикладу служби голосових дзвінків типова ситуація з помилками виникає, коли користувач намагається зателефонувати іншому користувачеві, який недоступний. TS моделює цей сценарій, створюючи дзвінок користувачеві, якого не існує.
- покинуті сценарії - цей тестовий сценарій заснований на занедбаній взаємодії з сервісом. TS повинен імітувати користувачів, які відмовляються від дзвінків, перш ніж вони обриваються. Наприклад, користувач ініціює голосовий дзвінок другому користувачеві, але перш ніж другий користувач відповість, перший скасовує взаємодію служби.
- відхилені сценарії - у сервісі, що включає більше ніж одного користувача, може статися що один із користувачів - не той, хто ініціює послугу - відмовляються від запиту на обслуговування. У голосовому дзвінку це відбувається, коли один користувач викликає другого користувача, проте другий користувач відхиляє виклик.[8]

На рисунку 3.4 представлений стан машини UE1. Загальна схема опису стану машин користувачів, які працюють з послугами зв'язку, може бути описана як у цьому прикладі; поведінку будь-якого користувача, залученого

до тестового сценарію, можна моделювати аналогічно. Запускається UE1 від стану, який називається доступним. Користувач доступний, коли його можна використовувати для створення нового дзвінка. Це три типи дій, які можуть змінити стан цього користувача:

- зовнішні дії - ці дії ініціюються генератором навантаження, який керує всіма користувачами. Генератор навантажень може вирішити, що користувач починає новий сценарій виклику, і робить це, ініціюючи дію для запуску сценарію.

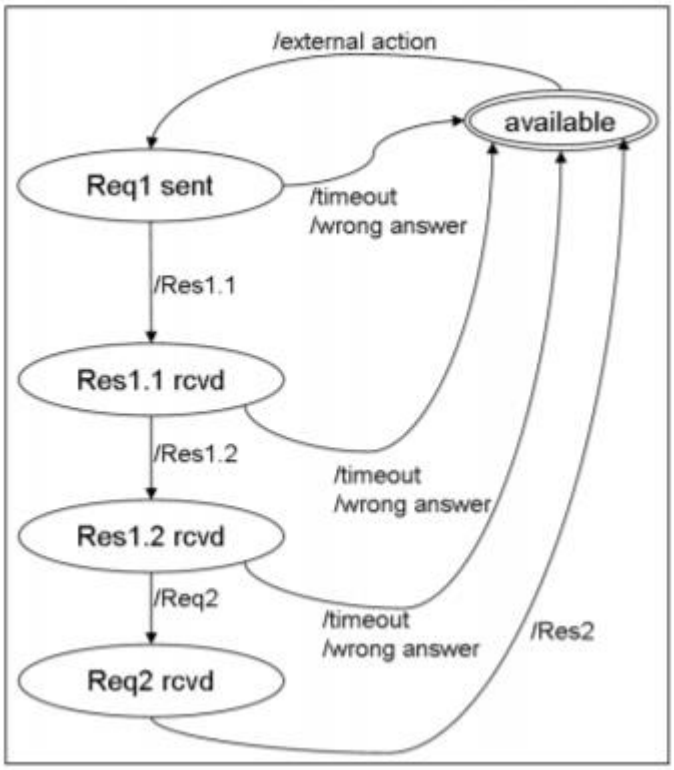

Рисунок **Error! No text of specified style in document.**.8– приклад машини стану користувачача

- події зв'язку - ці дії відповідають повідомленням, отриманим від SUT або надісланим SUT. Повідомлення можуть бути правильними або неправильними. В будь-якому випадку користувач повинен реагувати. Тому машина стану повинна моделювати і такі винятки.
- події тайм-ауту - для кожного запиту запускається таймер транзакцій. Він закінчується через певний проміжок часу. Подія тайм-ауту

означає, що жодне повідомлення для асоційованої транзакції не отримано.

UE на малюнку 4.4 запускає новий сценарій виклику на запит зовнішньої дії, виданий генератором навантаження. UE створює першу транзакцію і відправляє перше повідомлення. Якщо SUT відповідає на повідомлення Res1.1, UE переходить у стан Res1.1 rcvd. В іншому випадку, якщо настає тайм-аут, або якщо SUT відповідає з іншим типом повідомлень, сценарій вважається помилковим і UE переходить в доступний стан, що означає, що користувач може знову використовуватись для іншого дзвінка. З стану Res1.1 rcvd користувач переходить у стан Res1.2 rcvd, якщо отримано повідомлення Res 1.2. Дві транзакції є незалежними та виконуються в різні моменти часу. Це означає що користувач може чекати довільну кількість часу, поки другий UE не надішле повідомлення Req2. Якщо UE2 або SUT надсилає повідомлення іншого типу замість Req2, UE1 переходить у доступний стан. Однак для того, щоб переконатися, що UE1 закінчить стан машини, таймер закінчується після достатнього проміжку часу. Якщо цей таймер закінчиться, це означає, що друга транзакція ніколи не відбудеться. Якщо Req2 отриманий, UE1 переходить у стан rqvd Req2. З цього стану надсилання відповіді Res2, переводить UE1 у доступний стан.

3.4.2 Цілі дизайну

Оцінка ефективності базується на двох етапах: по-перше, зібрати вимірювання щодо кожного поміркованого сценарію тесту під час виконання робочого навантаження та по-друге, вивести показники ефективності щоб оцінити вимоги до продуктивності. Зібрані вимірювання складаються з журналів подій кожного повідомлення пов’язаного з SUT. Крім того, часова мітка, коли повідомлення стає видимий для TS реєструється разом із повідомленням.

Вимоги до продуктивності стосуються частоти помилок та затримок. Вони оцінюються окремо для кожного випадка використання поверх зібраних вимірювань. Для кожної вимоги до продуктивності завдання

дизайну (DO) визначається як порогове значення, яке потім використовується для порівняння показників.

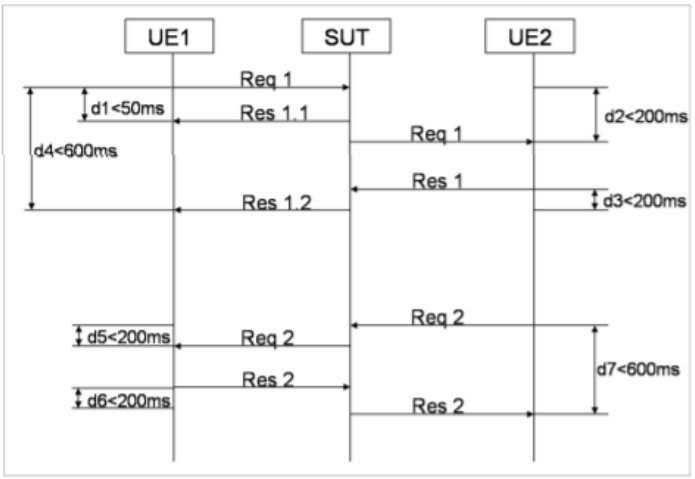

Рисунок **Error! No text of specified style in document.**.9– цілі дизайну

DO для затримок. На рисунку 3.5 зображено один потік сценарію, включаючи пов'язані DO для затримок зв'язку. DO для затримок бувають двох типів:

- затримка DO - визначає максимальний час, необхідний для отримання повідомлення через мережу SUT від абонента UE1 до виклику UE2.
- час обороту транзакції DO - визначає максимальний час, необхідний для завершення транзакції включаючи всі повідомлення.[8]

Ці DO визначають тимчасові межі затримки SUT і можуть бути змодельовані в машині стану користувача (див. рис 3.5) як події таймера. Коли термін дії закінчується, це означає, що DO для цього тестового сценарію перевищено, і тестовий сценарій вважатиметься невдалим.

DO для коефіцієнта помилок. DO для коефіцієнта помилок встановлює поріг допустимого відсотка помилок із загальної кількості сценаріїв. Відмінність від DO для затримок, це те, що DO для показників помилок повинні встановлюватись на рівні випадку використання, такий показник помилок перевіряється для всіх типів сценаріїв, до яких належать цей випадок використання.

Композиція набору трафіку

TS застосовує навантаження на SUT, яка складається з трафіку, генерованого великою кількістю окремо імітованих UE. Кожен UE виконує індивідуальний сценарій. Очевидно, що різні сценарії можуть поєднуватися в одному навантаженні. Концептуальне питання - як дозволити інженерам-тестувальникам можливість визначити композиції сценаріїв. Концепція, що охоплює цей аспект, називається набором трафіку.

Таблиця **Error! No text of specified style in document.**.1– приклад композиції трафіку

Scenario Type	Scenario Label	Scenario ratio	Scenario distribution
registration	s1.1	5%	poisson
re-registration	s1.2	20%	poisson
de-registration	s1.3	5%	poisson
successful voice-call	s2.1	21%	poisson
abandoned voice-call	s2.2	4%	poisson
rejected voice-call	s2.3	4%	poisson
fail voice-call	s2.4	1%	constant
successful page-mode messaging	s3.1	38%	poisson
failed page-mode messaging	s3.2	2%	constant

Приклад набору трафіку представлений у таблиці 3.1. Він складається з поєднання дев'яти сценаріїв, вибраних із різних випадків використання. Назви сценарії використовуються тільки для ілюстрації концепції. У межах встановленого трафіку кожен сценарій має відносну частоту виникнення, що інтерпретується як його ймовірність виникнення під час виконання процедури перевірки ефективності. Ця частота вказує на те, як часто сценарій слід виконувати під час повного виконання тестування на продуктивність.

Поки насправді навантаження є випадковим, щоб уникнути постійної інтенсивності навантаження, сценарії розробляються відповідно до розподілу прибуття, який описує швидкість прибуття подій сценарію з набору трафіку. Коефіцієнт прибуття характеризує еволюцію середньої норми прибуття як функцію часу протягом тривалості процедури перевірки працездатності. Приклад такого процес прибуття - процес Пуассона, який часто використовується при моделюванні телекомунікаційного трафіку.[8]

Приклад гістограми розподілу спроб сценарію в результаті виконання тесту на ефективність представлений на рисунку 3.6 На графіку відображаються частоти різної інтенсивності навантаження, як значення спроб сценарію в секунду, які відбуваються в ході виконання тесту. Наприклад, значення інтенсивності навантаження 86 на осі X має частоту виникнення, близьку до 0,4 на осі Y. Крива показує, що частота значень вище середнього значення і нижча для значень, віддалених до середнього значення, яке відповідає розподілу Пуассона.

Хоча концепція набору трафіку дозволяє використовувати будь-яку комбінацію сценаріїв, релевантний набір трафіку - це сукупність сценаріїв, за якими визначається ймовірність виникнення в реальній ситуації. Джерело інформація для вибору набору трафіку надається статистикою телекомунікацій, зібраною за ці роки.

Однак ця статистика, на жаль, обмежена в просторі та в часі. Статистика також не є репрезентативною для поточного соціального контексту, оскільки кількість користувачів зросла, послуги здешевшали тощо. Ця теза не враховує аспекту вибору набору трафіку, але пропонує концепцію експериментувати з будь-якою можливою комбінацією.[8]

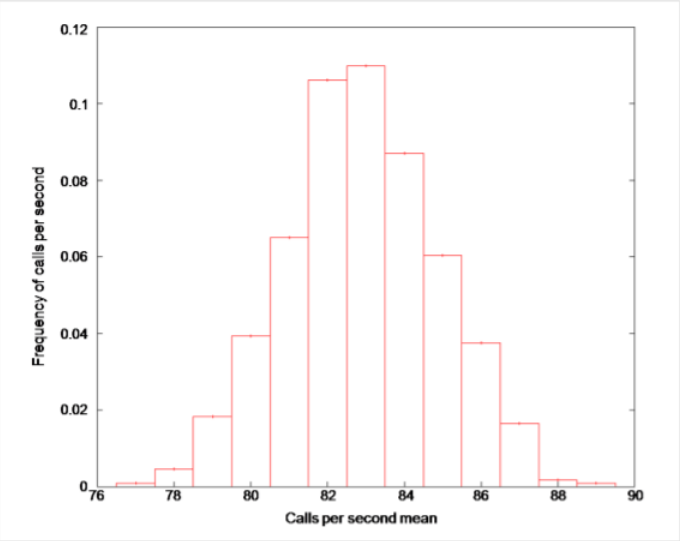

Рисунок **Error! No text of specified style in document.**.10- приклад гістограми з розподілом запитів на сценарій

3.4.2 Профіль трафіку та часу

Наступна концепція - це профіль часу та трафіку, який використовується для опису інтенсивності навантаження, тобто зміною швидкості прибуття сценаріїв, протягом тривалості тесту. Профіль часу трафіку визначає інтенсивність навантаження як функцію сплившого часу під час тесту на працездатність. Функція повинна забезпечити достатню кількість зразків, що формуються протягом певної тривалості часу, щоб можна було збирати показники ефективності з певним рівнем довіри.

Профіль часу трафіку в поєднанні з набором трафіку гарантує, що навантаження складається як поєднання сценаріїв, коли кожен сценарій із специфікацією та тривалістю інтенсивності навантаження на конкретному рівні достатньо довгий, щоб зібрати відповідні вимірювання для оцінки працездатності.

Загальний профіль трафіку та часу - це профіль трафіку та часу, який базується на формі сходинок (див. рис. 3.7). Ширина сходового кроку обрана таким чином, щоб зібрати достатню кількість зразків під час постійного середнього рівня прибуття сценарію. У представленій методології є лише східчаста форма.[8]

Профіль трафіку та часу керується набором тестових параметрів, які регулюють поведінку випробування по часу виконання, наприклад, генерації навантаження. Поки профіль трафіку та часу стосується певної форми трафіку параметри можуть бути визначені глобально для цієї форми. Загальні параметри для профілю трафіку-сходинки часу:

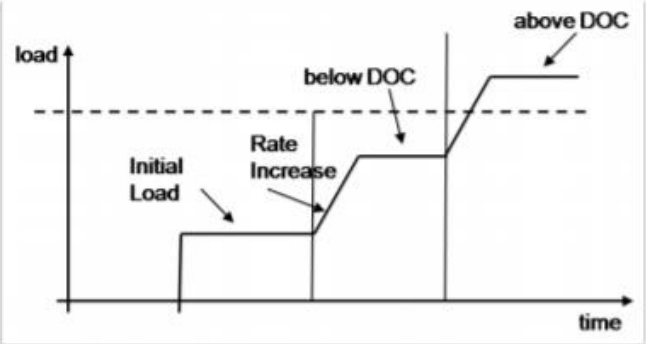

Рисунок **Error! No text of specified style in document.**.11– східчастий профіль трафіку-часу

Таблиця **Error! No text of specified style in document.**.2– приклад конфігурацію профілю трафіку-часу

Stair-Step Traffic-Time Profile Parameters	Value
PX_SimultaneousScenarios	2
PX_TotalProvisionedSubscribers	20000
PX_PercentRegisteredSubscribers	100%
PX_StepNumber	4
PX_StartStepTransientTime	60 sec
PX_EndStepTransientTime	60 sec
PX_StepTime	600 sec
PX_SApSIncreaseAmount	10
PX_SystemLoad	100

- час перемішування - кількість часу, коли завантаження системи представлено на SUT на початку тесту. В цей проміжок часу, як кажуть, записи бази даних "перемішані".
- загальна кількість модельованих користувачів - це опис загальної кількості модельованих користувачів. SUT повинна бути спроможна визначати цих користувачів протягом виконання тесту, але тестова система може імітувати лише їх частину.
- відсоток активних користувачів - цей параметр описує середній відсоток модельованих користувачів, які вважаються активними та використовуються TS. Наприклад, якщо SUT потрібно підтримують один мільйон абонентів, TS можуть попросити моделювати лише 20%.

- збільшення сценарію - кількість спроб сценарію, за допомогою яких використання сценарію збільшується.
- кількість кроків - кількість кроків у тесті на працездатність, як описано в процедурі перевірки працездатності (див. рис 3.8)
- тривалість кроку - кількість часу для виконання тесту з заданим навантаженням на систему (тестовий крок) перед збільшенням навантаженням.
- час початку перехідного періоду - інтервал на початку кроку, під час якого здійснюються спроби сценарію але вони не рахуються. Він повинен змусити TS чекати, поки SUT підготується до нового навантаження так, щоб на показники попереднього кроку не впливав новий крок.
- кінцевий перехідний час - інтервал в кінці кроку, щоб змусити TS чекати завершення поточних транзакцій. TS підтримує поточне навантаження, але не враховує показники для жодної з нових створених транзакцій.
- завантаження системи - початкова швидкість, з якою надходять спроби сценарію.

Таблиця 3.2 ілюструє профіль часу та трафіку для проведення тесту на ефективність з 20000 користувачів. Усі користувачі активні на початку тесту. Початкове завантаження системи - 100 спроб сценаріїв в секунду (SAPS), а потім збільшується в чотири рази за допомогою 10 SAPS. Тривалість кожного кроку - 600 секунд. [8]

3.4.3 Визначення цілей дизайну ємності

Основним показником порівняння, про який повідомляє TS, є проектна об'єктивна ємність (DOC) і він представляє собою найбільше навантаження, яке може витримати SUT за визначених умов, наприклад, рівень помилок не повинен перевищувати заданий поріг. DOC призначений для характеристики загальної продуктивності SUT. Його не слід плутати з DO, які описують вимоги щодо затримок або частоти помилок сценарію. Це число повинно

слугувати для порівняння між версіями SUT, різних апаратних платформ, різних продуктів тощо.

Це число визначається як інтенсивність навантаження, для якого SUT все ще може переносити навантаження при певних умовах якості. Підвищення інтенсивності навантаження автоматично вплине на SUT, щоб перевищити потрібні вимоги до продуктивності, наприклад, час реакції повинен бути нижче заздалегідь визначеної межі. Виконання тесту на ефективність означає, що вибрані сценарії є різними випадками використання, що виконуються одночасно. Кожен розпочатий сценарій стає спробою сценарію. Будь-який із сценаріїв що неправильно обробляється, зараховуються як IHSA. IHSA використовуються для обчислення показника IHS%, який є порівняно з попередньо визначеним порогом, який обирає тестовий інженер. Останнє значення інтенсивності навантаження до IHS% перевищує поріг, приймається як значення DOC для цієї системи. [8]

3.4.4 Процедура навантажувального тесту

Тест на працездатність виконується відповідно до вибраного профілю трафіку-часу. Виходячи з цієї форми, кілька тестових етапів із збільшенням навантажень виконуються з ціллю вимірювання DOC SUT. Тест закінчується, коли досягається поріг частоти IHSA. Це робиться шляхом знаходження навантаження, при якому коефіцієнт помилок знаходиться нижче порогового значення, інше навантаження, при якому рівень помилки перевищує поріг і сковуючи DOC між цими двома навантаженнями. Тест починається з недовантаженої системи, яка поступово підводиться до свого DOC і підтримується при цьому навантаженні певний час. Час, коли система працює на своєму DOC, повинен бути достатньо довгим, щоб отримати змістовні дані та висвітлювати можливі проблеми з продуктивністю, такі як витоки пам’яті та перевантаження черги на повідомлення.[8]

Інтервал DOC шукається відповідно до процедури перевірки працездатності, описаної на рисунку 3.8. Випробування починається з початкового навантаження, і збільшується в кілька етапів. Кожен крок

виконується за встановлену тривалість часу. Результати аналізуються після кожного кроку для того, щоб оцінити, чи було досягнено DOC. Якщо ні, новий тест на працездатність виконується з підвищеним значення навантаження. Після ряду ітерацій ситуація, зображена на рисунку 3.7, де DOC оточений між двома значеннями навантаження. Однак DOC враховує нижнє значення навантаження. Для того, щоб знайти DOC з більш тонкою деталізацією, цей інтервал можна розділити на додаткові значення навантаження та провести ще один тест.

```
1.      configure the system load for an initial load
2.      run the test for TshortStep duration
3.      if the step has IHS<0.1 increase the load with
        SApSIncreaseAmount and go to 2 else go to 4
4.      decrease the load with SApSIncreaseAmount
5.      execute confirmation run for Tconfirmation duration
6.      if the confirmation run has IHS>0.1 go to 4
7.      the DOC is the current load
```

Рисунок **Error! No text of specified style in document.**.12– процедура навантажувального тестування

Для кращої впевненості в результатах процедура закінчується кампанією підтвердження запусків. На відміну від коротких кроків, щоб визначити значення DOC, запуски підтвердження виконуються протягом досить тривалого періоду часу, такого як кілька мільйонів транзакцій. Якщо під час запусків підтвердження, SUT виходить з ладу, тоді DOC повинен бути нижчим, ніж перша оцінка, і, отже, новий цикл підтвердження виконується для нижчого значення. Тест закінчується, коли запуски підтвердження закінчується успішно, тобто IHS% знаходиться нижче порогового значення.[8]

3.4.5 Звіт про перевірку продуктивності

Звіт про перевірку продуктивності збирає та візуалізує всі відповідні показники та статистику. Основна структура звіту про перевірку продуктивності показана на рисунку 3.9. Це документ із супровідним файлам, що забезпечує повний опис виконання тесту на продуктивність. SUT і TS, а також їх параметри описані досить детально, щоб інша людина могла

повторити тест. Результати тестування повинні максимально інтуїтивно представляти обчислювані показники та набори даних у вигляді діаграм, графіків чи іншого візуального формату. Проста перевірка цієї інформації зображує поведінку SUT протягом усього часу виконання тесту.

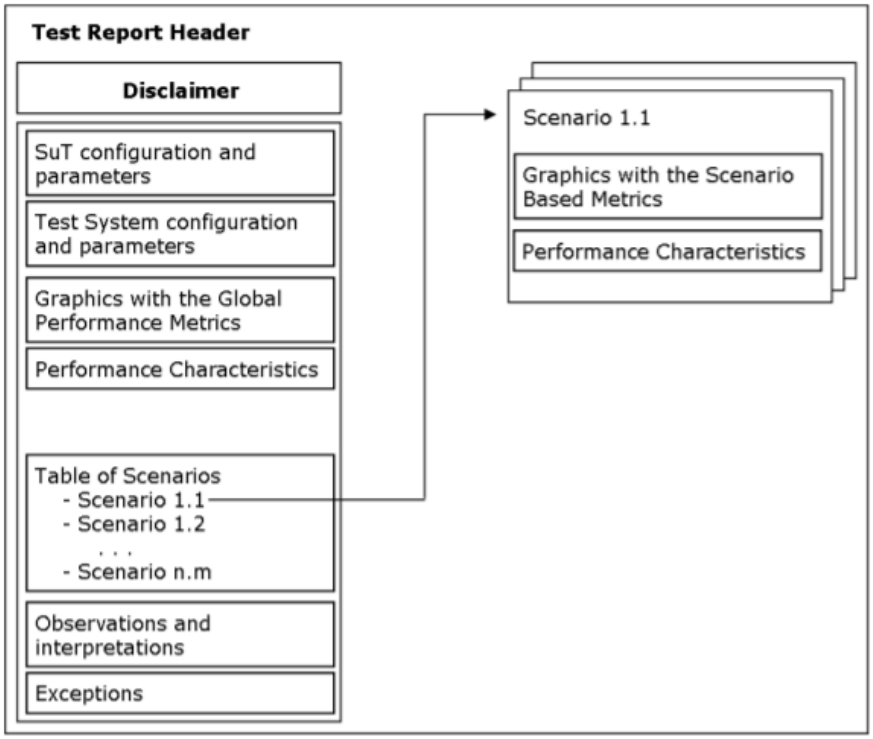

Рисунок **Error! No text of specified style in document.**.13– структура результатів навантажувального тестування

Звіт про перевірку продуктивності повинен візуалізувати глобальні показники ефективності та сценарії на основі показників відповідно. Показники представлені у вигляді графіків, на яких відображається показник значення за одиницю часу. Крім того, для кожної метрики повинні бути обчислені характеристики продуктивності роботи. Що стосується показників на основі сценарію, тестовий звіт містить перелік окремих звітів для кожного сценарію. Кожна сторінка сценарію включає графіки, сформовані з зібраних показників для цього сценарію та їх характеристики. [8]

Частота запитів та кількість помилок

На малюнку 3.10 наведено приклад візуалізації показників SAPS та IHS. Тест складається з: двох кроків навантаження. Перший рядок вказує на інтенсивність навантаження. Додатково, пункти, що вказують кількість сценаріїв, створених у кожну секунду. Пунктирна лінія (другий рядок)

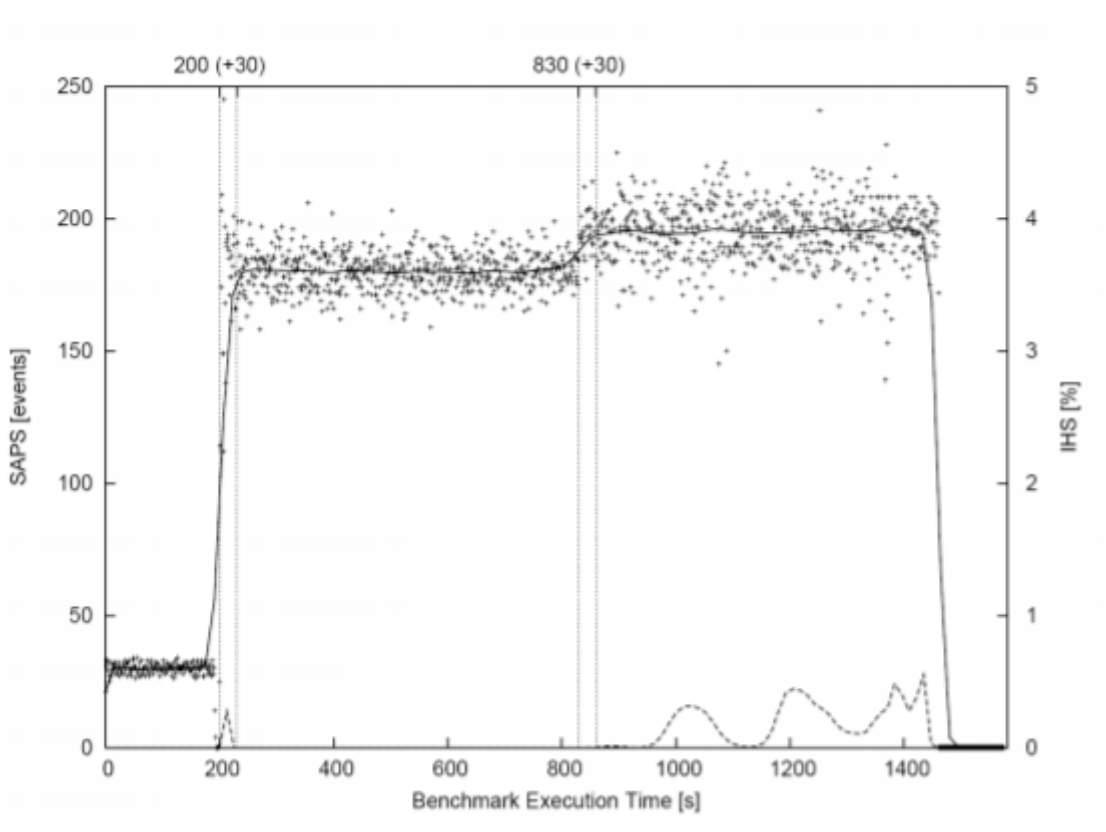

Рисунок **Error! No text of specified style in document.**.14– візуалізація кількості запитів та помилок

вказує на IHS у відсотках відмови від загальної кількості запитів. Цей графік візуалізує дві метрики паралельно, де одна з метрик - сама інтенсивність навантаження. Така ж ідея може бути застосовується до всіх інших показників, і навіть більше двох показників можна візуалізувати паралельно.

Затримка реакції

На рисунку 3.11 показана середня затримка встановлення виклику. Цей показник вимірює зворотний час між відправленням запиту повідомлення до отримання відповідного повідомлення відповіді. Час між двома подіями позначає фактичний час обчислення, необхідний SUT для обробки запиту і доставлення відповіді. Цей графік вказує на залежність між затримкою та рівнем навантаження: під час другого кроку навантаження затримка, очевидно, вище.

Використання ресурсів

Крім метрик, пов'язаних з протоколом, TS повинен контролювати споживання ресурсів SUT. На рисунку 3.12 показана залежність системного

часу процесора (пунктирна лінія) та часу користувача (пунктирна лінія) від швидкості завантаження. Попит на цей ресурс залежить від завантаженості системи. Рисунок 3.13 вказує потребу в пам'яті (пунктирною лінією) в ході процедури тестування продуктивності. [8]

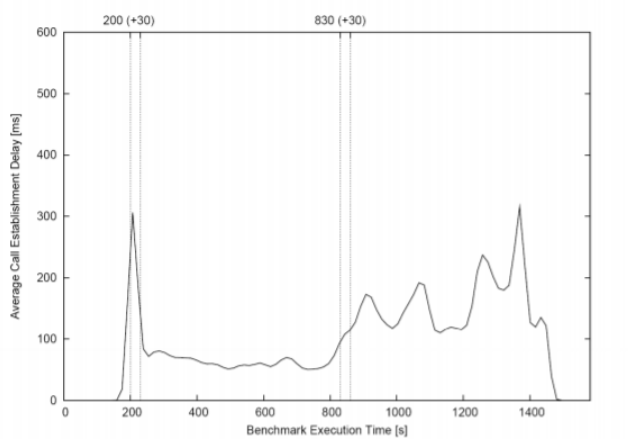

Рисунок **Error! No text of specified style in document.**.15– візуалізація затримки реакції тестованої системи

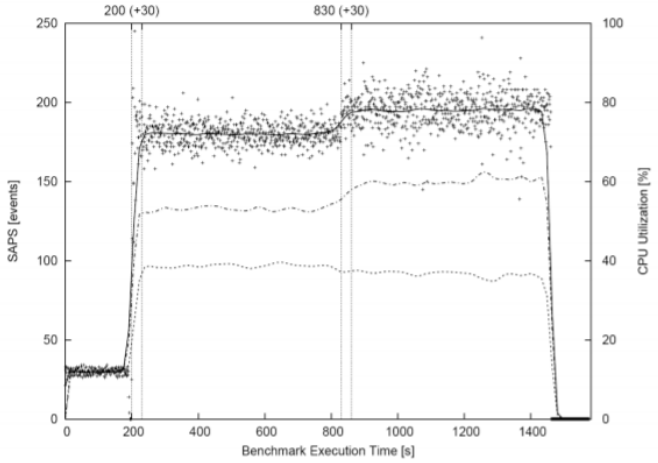

Рисунок **Error! No text of specified style in document.**.16– приклад візуалізації користування процесора

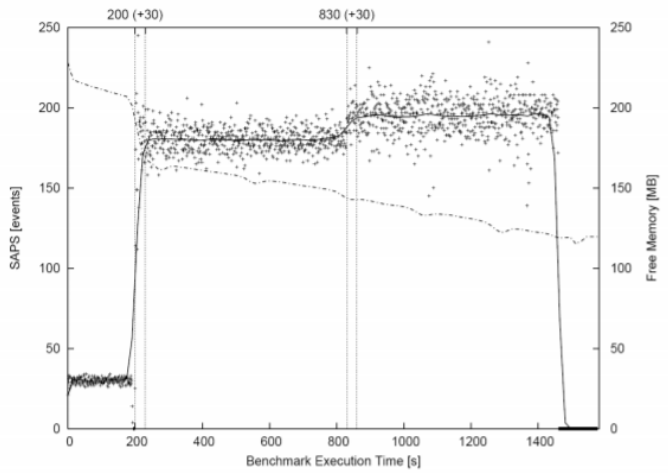

Рисунок **Error! No text of specified style in document.**.17– приклад візуалізації користування пам’яті

3.5 Каталог шаблонів реалізації робочого навантаження

Тестові шаблони є загальними, розширюваними та пристосованими реалізаціями тесту. Тестові шаблони є багаторазовими і, як аналогія програмних моделей, отримані з методів тестування та тестових рішень. Вони доступні у вигляді методів проектування, бібліотек програмного забезпечення та / або генераторів коду, які надають інженеру-випробувачу готовий до використання код.

Хоча мови реалізації тесту дуже гнучкі і дозволяють писати тести різними способами, дуже часто декілька моделей архітектурного дизайну можна розпізнати в нефункціональному тесті реалізації. Поширена практика в розробці програмного забезпечення є шаблонно-орієнтований архітектурний дизайн. Аналогічно можна визначити і архітектурні шаблони розробки тесту на продуктивність. Метою цього розділу є аналіз моделей реалізації які можуть бути використані при реалізації TS.

Загальна специфікація поведінки тесту реалізується як сукупність паралельних процесів виконання. Більшість операційних систем надають функції, що дозволяють процесу містити декілька потоків управління. На цьому рівні загальною проблемою є розв'язання ресурсних суперечок, планування, уникнення тупикових ситуацій, інверсія пріоритету та умови перегонів.

На тестовій платформі потоки використовуються для тестування конкретних завдань, таких як генератор навантажень, обробники стану користувачів, але в межах платформи виконання, також можуть бути потоки, присвячені завданням, які не тестуються, наприклад, потік збору сміття на платформі на базі Java співіснують. Основна увага в цій тезі приділяється моделям, пов'язаним лише з тестовою поведінкою.

Зайняті потоки відповідають різним вимогам. Вони можуть бути згруповані в типи дій, наприклад, потоки для генерації навантаження, потоки для обробки стану. Тип потоку може бути ініційований для довільної кількості разів. Крім того, для кожного типу дій можна створити більше одного потоку, щоб збільшити паралелізм.[8]

Активність потоків орієнтована на взаємодію з SUT, наприклад, підготовку повідомлень, спілкування, перевірку стану. Поведінка потоку може складатися з простих протоколів обміну даними або з більш складних машин стану, заснованих на сеансах з кількома транзакціями зв'язку. Складність складається з інструкцій з обробки даних, інструкцій зв’язку, умов часу. Набір операцій, пов'язаних з тестуванням, та їх потік виконання:

- створення користувача - стосується створення користувача та його ініціалізації. На цьому кроці тестовий процес створює новий запис у сховищі користувачів та встановлює початковий статус цього користувача.
- термінація користувача - після припинення користувача запис у сховищі користувачів повинен бути видалений. Крім того, якщо комунікація базується на транзакціях, пов'язані транзакції або таймери також потрібно видалити.
- обробка даних - в процесі тестування з'являються різні обчислення, наприклад, підготовка даних, оцінка відповідей на SUT.

- кодування - дані кодуються у форматі даних SUT. Ця операція стосується лише інструментів, які не працюють безпосередньо над вихідними даними.
- відправка - операція, під час якої TS зобов'язується надсилати подразники до SUT. Ця операція також розуміє основні операції зв'язку, наприклад, операції сокета.
- ставання в чергу - операція, яку виконує TS при отриманні повідомлення від SUT. Кожне отримане повідомлення ставиться у чергу, звідки воно буде оброблено на подальшому етапі.
- декодування - зворотня операція кодування; неочищене повідомлення, що становить повернуту SUT інформацію, перетворюється на структурну сутність, яку можна додатково дослідити за допомогою TS. Цей крок також характерний лише для тих інструментів, які не працюють безпосередньо над необробленим повідомленням, а швидше створюють структурне уявлення повідомлення.
- фільтрування - отримані повідомлення фільтруються за правилами, що визначають тип повідомлень. На практиці ТС має перевірити більше одного фільтра. Якщо один з фільтрів відповідає повідомленню, TS виконує дії, ініційовані цим фільтром.
- тайм-аут таймера - це операція перевірки того, чи відбувається подія, наприклад, реакція SUT, синхронізація. Таймер - це окремий процес, який видає події для процесів поведінки.
- ведення журналів - це операція з отримання даних журналу, коли відповідна подія відбувається в ТС.

Взаємодія між тестовою поведінкою тесту та даними користувачів зображена на рисунку 3.14. Репозиторій даних користувачів складається з інформації про особу користувача та списку активних сценаріїв, в яких кожен користувач бере участь. Потік імітує поведінку користувача, яка

складається в цьому прикладі з операцій зв'язку, представлених на рисунку як надсилання та отримання операції. Для спрощення потік, який обробляє стан машини користувача, називається головним потоком. При будь-якій операції прийому або відправлення потік поновлює стан відповідного сценарію. Коли сценарій закінчується, ідентифікатор сценарію видаляється зі списку активних сценаріїв, і користувачеві стає доступним для нового сценарію. [8]

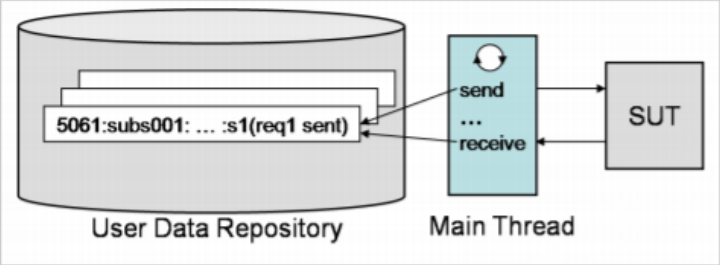

Рисунок **Error! No text of specified style in document.**.18– обробка стану користувача потоком

У наступному каталозі представлені визначені схеми дизайну. Вони структуровані в групи та підгрупи, які характеризуються загальною метою.

Машина стану користувача

- Машина стану з певним обробником: Машина стану призначена лише для одного користувача.
- Машина стану з загальним обробником: Машина стану може працювати паралельно станам більше одного користувача.

Використання потоків в обробці користувачів

- Поведінка одного користувача на потік: один потік обробляє лише одного користувача.
- Послідовна поведінка користувачів на потік: один потік обробляє більше одного користувача в послідовному порядку.
- Поведінка користувача з перемежованою поведінкою на потік: один потік обробляє одночасно більше ніж одного користувача.

Обробка таймерів

- Обробка таймером на основі режиму сну: таймер обробляється в основному потоці за допомогою режиму сну.

- Обробка таймером на основі окремого потоку таймера: таймери обробляються окремим потоком.

Надсилання повідомлення

- Відправка в основний потік: операція надсилання обробляється безпосередньо головним потоком.
- Відправка окремим потоком: операція надсилання обробляється окремим потоком.
    - Відправка окремим потоком за запитом: окремий потік відправки обробляє лише один запит; після цього він гине.
    - Надсилання з окремим потоком за сеанс: окремий потік відправлення обробляє всі запити сеансу.
    - Відправка за допомогою потоку: Група потоків, тобто пул потоків, обробляє всі запити всіх користувачів.

Отримання повідомлень

- Прийом у головний потік: очікування отриманих даних виконується безпосередньо в основному потоці.
- Прийом з окремим потоком: Очікування отриманих даних виконується окремим потоком.
    - Прийом з окремим потоком за сесію: очікування отриманих даних реалізується в окремому потоці.
    - Прийом з басейном потоків
        - Шаблон реактора: очікування отриманих даних реалізується пулом потоків, який використовує схему реактора.
        - Шаблон проактора: Очікування отриманих даних реалізується пулом потоків, який використовує шаблон проактора.

Контроль навантаження

- Генерація одноразового навантаження: навантаження створюється одним генератором навантаження.

- Кілька генераторів навантаження з централізованими даними
    - Метод натискання: навантаження генерується декількома генераторами з використанням централізованих даних. Дані підштовхуються до генераторів навантаження.
    - Метод витягнення: навантаження генерується декількома генераторами з використанням централізованих даних. Дані витягуються генераторами навантаження.
- Генератор декількох навантажень з децентралізованими даними: навантаження створюється декількома генераторами навантаження, які використовують окремі дані.

Інкапсуляція даних

- Шаблон буферного рядка: повідомлення обробляються у вигляді буферних рядків.
- Зменшення вмісту: повідомлення представлені у вигляді структури з мінімальним вмістом.
- Структуроване представлення вмісту повідомлення: Весь вміст повідомлень представлений у вигляді структури.
- Структуроване представлення покажчиків на зміст повідомлення: Зміст повідомлень представлений у вигляді структури покажчиків на місця всередині повідомлення.

Населення користувачів

- Одномісне населення: Населення підтримується одним басейном.
- Кластери користувачів: населення розділене на кластери. Для кожного сценарію існує один кластер.
- Мінімальна кількість кластерів: кількість населення розділена на мінімальну кількість кластерів. [8]

3.5.1 Шаблони дизайну машин стану

Поведінка користувача, як правило, реалізується як машина стану, яка зберігає стан користувача в даний момент часу і може працювати на вході, щоб змінити стан та / або викликати дію або висновок для будь-якої зміни.

Елементи машини стану поведінки користувача в тестовій поведінці розпізнаються:

- початковий стан - користувач спочатку незареєстрований.
- набір можливих подій введення - будь-яке повідомлення, отримане від SUT, є входом у поведінку користувача і може змінити його поточний стан.
- може ввести новий стан відповідно до вводу даних - багато типів повідомлень з SUT можуть змінити поточний статус.
- набір можливих дій або вихідних подій, що є результатом нового стану - для більшості входів з SUT потрібно прийняти реакцію, наприклад, створити нове повідомлення і відправити назад в SUT як відповідь.

З точки зору реалізації, стан машини користувача може бути виконаний або певним чином, або загальним способом.

Шаблон: Машина стану з конкретним обробником (SM SpecHdl)

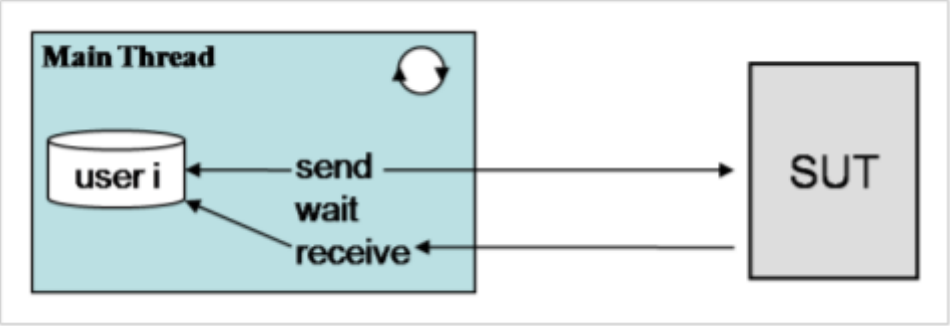

Рисунок **Error! No text of specified style in document.**.19– конкретний обробник подій

Опис

Конкретний підхід поводження представлений на рисунку 3.15. Машина стану користувача повністю обробляється одним потоком, і всі дані, пов’язані з його машиною стану, зберігаються локально. При отриманні або

відправці подій дані користувача змінюються локально, потік не повинен взаємодіяти із зовнішнім сховищем. Це дозволяє уникнути часу синхронізації з рештою потоків.

Переваги

Що стосується реалізації цієї дизайнерської моделі, то її легко реалізувати, оскільки вся логіка стосується лише одного користувача. Поведінка не повинна виконувати складні перевірки, щоб визначити, якого користувача потрібно оновити для певного повідомлення. Події таймера також можуть бути імітовані локально всередині потоку, використовуючи подібні до сну операції. Потік перебуває у режимі очікування протягом короткого періоду часу, поки не закінчиться час очікування або настане дійсна подія, наприклад, отримана відповідь від SUT.

Крім того, ця модель має перевагу в тому, що машина стану реалізована дуже ефективно, оскільки в кожному стані дозволено лише допустимий вибір. Все несподіване вважається недійсним, і ситуація передається оброблювачу помилок. Однак це добре працює лише в тому випадку, якщо поведінка складається лише з одного сценарію. Якщо поведінка складається з більш ніж одного паралельного сценарію, то машина стану може бути більш складною. **Недоліки**

На жаль, для величезної кількості користувачів ця схема не є практичною, оскільки потрібно створити багато потоків. Найбільш використовувані операційні системи (засновані на Unix або Windows) стикаються з усіма серйозними проблемами в умовах, що включають величезну кількість потоків. Чим більше потоків створено, тим більше контекстних комутаторів потрібно використовувати, і тим менше процесорних прорізів призначено для потоку. Також синхронізація багатьох потоків може спричинити проблему, оскільки потребує занадто багато часу. Якщо поведінка тесту потребує багатьох точок синхронізації, витрачати час на синхронізацію може бути занадто дорого. [8]

Шаблон: Машина стану з загальним обробником (SM GenHdl)

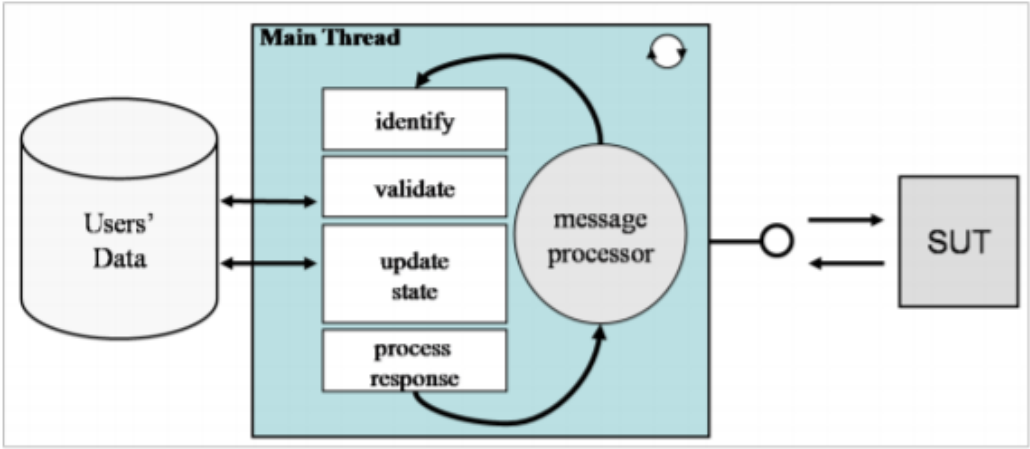

Рисунок **Error! No text of specified style in document.**.20– загальний обробник подій

Опис

Конкретний шаблон обробника подій страждає від проблем із продуктивністю, коли створюється занадто багато потоків. Тому кращим рішенням буде створити менше потоків, використовуючи один потік для обробки більш ніж одного користувача..

Ця закономірність забезпечує управління даними в усьому світі, і стан машини можна розглядати як процесор повідомлень. Функціональність цього шаблону зображена на малюнку 3.16. У будь-якому новому отриманому повідомленні процесор повідомлення ідентифікує користувача, якому належить нове повідомлення, та оновляє його стан необхідним чином. Якщо подія вимагає також створення нового повідомлення, яке має бути відправлено назад до SUT, потік діє відповідно.

Переваги

З точки зору програмування, ця модель дещо складніша за попередню, але застосування деяких конвенцій програмування полегшує достатню технічну реалізацію, наприклад, шаблонів, індексування.

Ця модель має перевагу в тому, що обробник може використовуватися для обробки довільної кількості користувачів паралельно, оскільки це залежить тільки від нового отриманого повідомлення. Поки дані користувача

зберігаються поза потоком, потік не контролює потік виконання; натомість він виконує свої дії лише тоді, коли вони викликані зовнішніми подіями.

Ще одна перевага, запропонована цією схемою, полягає в тому, що інформацією одного користувача може управляти більше ніж один потік. Оскільки потік не управляє інформацією користувача локально, він може бути призначений для обробки будь-яких довільних подій. Однак усі потоки повинні підтримувати інформацію користувачів. Ця концепція також пропонує можливість більш ефективного балансування між потоками, оскільки будь-який потік може обробляти будь-яку подію.

Недоліки

Недоліком цієї схеми є те, що обробник повинен перевіряти тип кожного нового повідомлення на предмет усіх очікуваних типів повідомлень. Це може бути неефективним, якщо застосовувати його до великої кількості типів, але, використовуючи спеціалізовані методи пошуку, такі як метод пошуку на основі дерева, ефективність та оптимізація можуть значно підвищитися.[8]

3.5.2 Шаблони використання потоків в обробці користувачів

У попередньому розділі були розглянуті дві моделі для реалізації машини стану користувача. Обидва підходи мають переваги та недоліки. У цьому розділі представлені ще три моделі та дизайн використання потоків. Ці шаблони засновані на тому, що не всі користувачі повинні бути активними одночасно. Це дозволяє уникнути існування неактивних потоків, створюючи нові потоки лише тоді, коли вони потрібні.

Шаблон: Обробник користувача з одним користувачем на потік (UH SingleUPT)

Опис

Найпростіший спосіб реалізувати користувача - це створити екземпляр потоку, що імітує лише цього користувача, тобто основний потік, як

показано на малюнку 3.17. Ця закономірність підходить до конкретної підходу управління машини стану (SM SpecHdl), але також може поєднуватися із загальним підходом. Однак якщо вибрано SM GenHdl, рекомендується скористатися одним із наступних двох шаблонів обробки користувачів.

Переваги

Цей шаблон легко застосувати до опису поведінки користувачів. Можна також повторно використовувати код з функціональних тестів.

Недоліки

Незважаючи на простоту написання тестів за допомогою цієї методики, існують два основні недоліки. По-перше, контроль навантаження важко реалізувати, коли тестовий інженер хоче підтримувати постійну кількість паралельних користувачів. Тестовий контролер повинен постійно контролювати кількість потоків, що діють паралельно, і кожного разу, коли потік закінчується, необхідно створити новий. По-друге, операції створення, запуску та завершення, застосовані до потоків, є дуже дорогими операціями щодо процесора. [8]

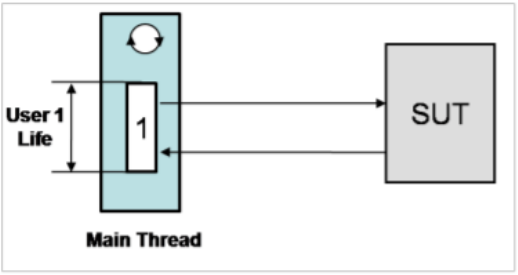

Рисунок **Error! No text of specified style in document.**.21– один користувач на потік

Шаблон: Обробник користувача з послідовними користувачами на потік (UH SeqUPT)

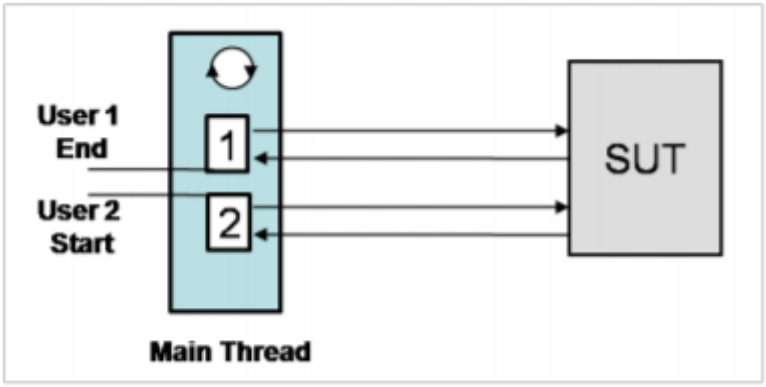

Рисунок **Error! No text of specified style in document.**.22– послідовність користувачів на потік

Опис

Через витрати на створення та знищення потоків бажано використовувати потоки для подальших користувачів, як тільки інші користувачі припиняють роботу. Ця картина проілюстрована на рисунку 3.18. Цей шаблон вимагає, щоб ідентифікатори користувачів зберігалися поза потоком і завантажувалися в потік лише тоді, коли користувач повинен стати активним. Ця закономірність усуває проблему створення та руйнування потоків. Потоки використовуються як пул потоків, де кожен потік може приймати будь-яку особу користувача. Під час випробування кількість потоків може бути збільшена або зменшена відповідно до інтенсивності навантаження. Таким чином, кількість активних користувачів підтримується та контролюється з поточних потоків.

Переваги

Потоки повторно використовуються для виконання нової поведінки користувачів у послідовному порядку. Таким чином буде витрачено менше ресурсів процесора для створення та знищення потоків.

Недоліки

Цю закономірність можна поєднувати з конкретною стратегією реалізації машини стану користувача, а також із загальним підходом. Однак якщо застосувати до загального підходу, процес зберігає стан не локально, а у сховищі зовнішніх користувачів.

Ще один недолік полягає в тому, що кількість потоків все ще залежить від інтенсивності навантаження. Чим більше навантаження, тим більша кількість потоків буде. [8]

Шаблон: Обробник користувача з перемежованими користувачами на потік (UH InterleavedUPT)

Опис

Кращим підходом до використання потоків є переплетення поведінки користувачів одночасно на тому ж потоці, як це показано на рисунку 3.19. Цей зразок може розглядатися як розширення попереднього шаблону з додаванням того, що користувачі одночасно обробляються. Таким чином, потік здатний паралельно моделювати довільну кількість користувачів.

Переваги

У поєднанні із загальним підходом до обробки стану (SM GenHdl) ця модель є найбільш гнучким та масштабованим рішенням. Один потік може працювати одночасно з багатьма користувачами. Більше того, немає потреби асоціювати користувачів з потоком, тобто різні події одного користувача можуть оброблятися довільно більш ніж одним потоком.

Недоліки

Якщо їх поєднувати з конкретною моделлю, суміш паралельних форм поведінки на одному потоці складно задати, і більшу частину часу код втрачає читабельність і стає важко підтримувати. [8]

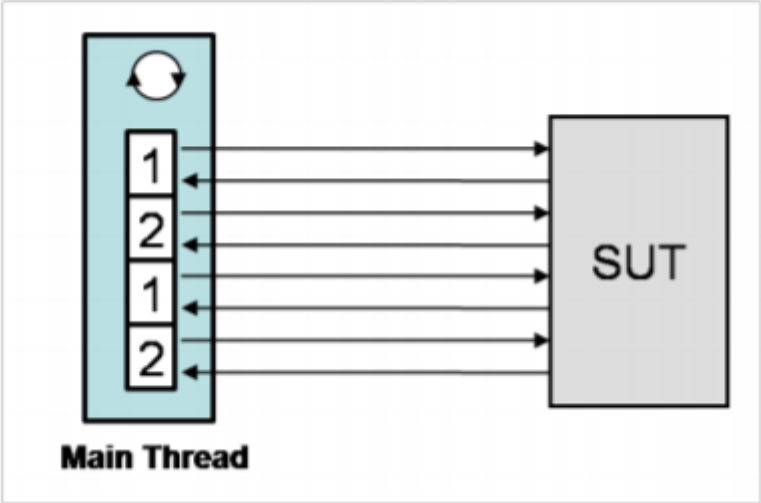

Рисунок **Error! No text of specified style in document.**.23– перемежовані користувачі на потік

3.5.3 Шаблони для таймерів

Всі протоколи, що використовуються сьогодні в телекомунікаціях, включають специфікації обмеження часу для максимального часу відгуку. Багато протоколів включають також специфікацію повторної передачі на рівні протоколу. Крім того, взаємодія користувача з SUT передбачає різні користувацькі часи, такі як час розмови або час дзвінка, який у тестах на ефективність також повинен бути імітований.

Усі ці технічні характеристики були враховані також у запропонованій методиці випробувань. DO посилаються на максимальний час відгуку. Ці значення виймаються із специфікацій протоколу або з Угоди про рівень обслуговування (SLA). TS повинна перевірити, якщо ці часові обмеження виконані, інакше сценарій вважається неадекватним опрацьованим. Якщо потрібно, час повторної передачі виймається безпосередньо із специфікацій протоколу. Вони також повинні бути імітовані TS. Якщо згідно із специфікацією, після ряду повторних передач SUT не реагує, сценарій також вважається недостатньо обробленим. Час користувача - це параметри тестування, які, як правило, випадковим чином розподіляються навколо середніх значень. Середні значення отримують із статистичних оцінок щодо поведінки реального користувача, наприклад, середня тривалість голосового дзвінка становить три хвилини.[8]

Виявлено два способи реалізації таймерів у тестових специфікаціях.

Шаблон: Таймер з режимом сну (T Sleep)

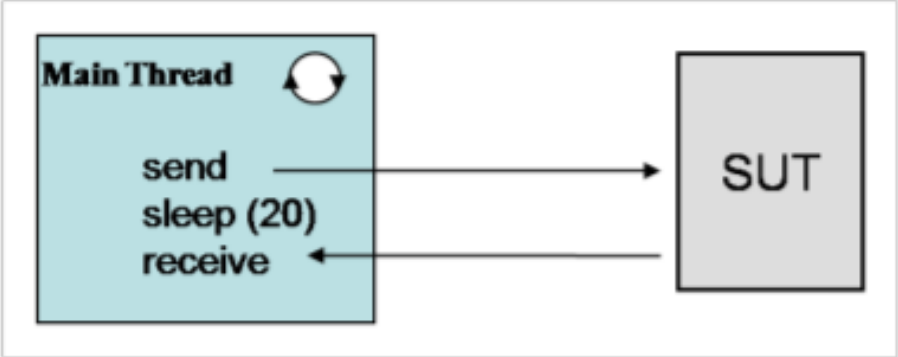

Рисунок **Error! No text of specified style in document.**.24– використання таймеру використовуючи сон

Опис

Операція сну - це функція потоку, яка змушує спливаючий потік спати протягом визначеного часу. Потік може зачекати, поки спливе час або сповіститься зовнішнім процесом, щоб прокинутися.

Операція сну може бути використана для реалізації таймерів, які використовуються в тестовій поведінці. Операцію можна запустити або для очікування відповіді SUT всередині потоку, або для перевірки, відповідає SUT чи ні. Ця картина проілюстрована на малюнку 3.20.

Переваги

Цю закономірність застосувати просто, і концепція таймера може бути реалізована в тому ж потоці, що й решта тестової поведінки.

Недоліки

Цей підхід дуже обмежений через час сну потоку, коли потік не може робити щось інше паралельно. Таким чином, потік не може імітувати більше одного користувача, оскільки він повинен чекати відповідей одного користувача. Аналогічно, потік не може бути залучений до паралельних викликів, оскільки він повинен відстежувати поточний виклик. Цей підхід обмежує можливості проектування станової машини лише шаблоном UH SingleUPT.

Таймер з окремим потоком таймера (T SepTT)

Опис

Підхід, який пропонує більшу гнучкість, передбачає використання зовнішнього потоку таймера, який керує всіми таймерами, що беруть участь у сценаріях користувачів. Функціональність цієї схеми проілюстрована на малюнку 3.21. Коли потік обробника події доходить до того, що потрібно запустити таймер, він просить потік таймера створити новий таймер, який помітить його, коли він закінчиться. Потоку таймера надається інформація про особу користувача та час дії. Потік таймера керує чергою таймерів та виконує їх у їх тимчасовому порядку. Коли створюється новий таймер, потік таймера планує нову подію в потрібному місці. Оскільки події таймера впорядковані на основі часу, потік таймера повинен переходити лише до наступного тайм-ауту.

Коли настає новий час очікування, потік таймера помічає потік обробника подій, відповідальний за користувача, який створив таймер. Це може просто статися, надсилаючи події тайм-ауту для цього користувача. Однак якщо користувач тим часом отримує дійсну відповідь від SUT, потік таймера повинен видалити подію тайм-ауту з черги планування.

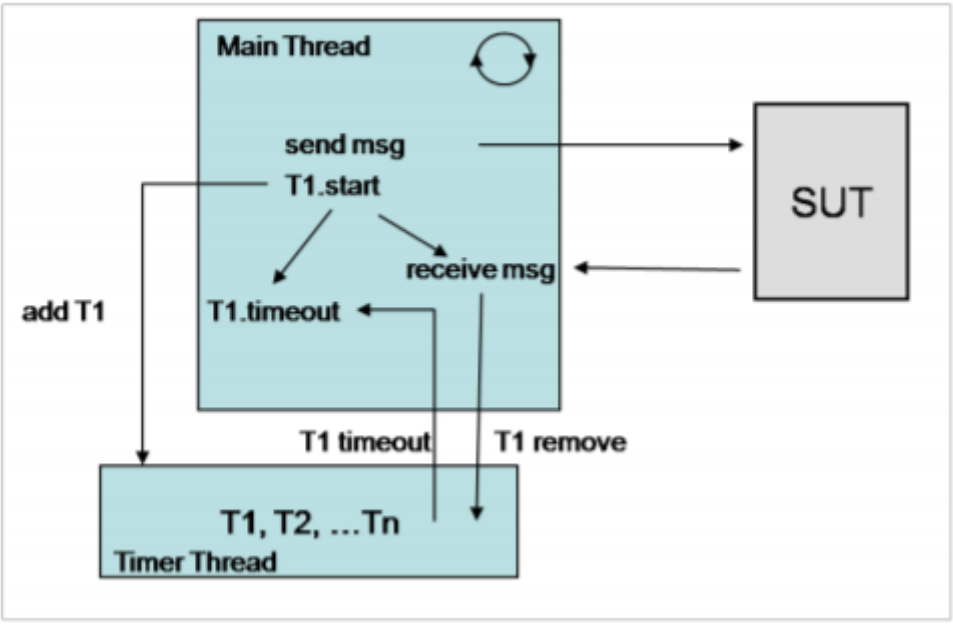

Рисунок **Error! No text of specified style in document.**.25- використання потоку для таймера

Переваги

Цей підхід сумісний з усіма описаними раніше моделями. Це дуже гнучко, оскільки дозволяє потокам обробки подій обробляти події для довільної кількості користувачів паралельно або навіть паралельно виклику того ж користувача. Більше того, події тайм-ауту обробляються як звичайні події, що робить концепцію більш загальною. Однак для більшої гнучкості можна створити більше одного таймерного потоку.

Недоліки

Одним з недоліків є те, що лише один таймер повинен обробляти всі таймери, створені в ході тестового виконання. Це може спричинити проблеми, коли через синхронізацію потрібно створити занадто багато таймерів. Однак цю проблему можна усунути, створивши кілька потоків таймера. [8]

3.5.4 Шаблони відправки повідомлень

Операції надсилання та прийому, як правило, вимагають тривалих термінів виконання через кодування даних, черги та мережевий зв'язок. Потоки, які їх виконують, блокуються до завершення операції. З цього погляду, як стверджувалося в попередніх зразках, не зручно дозволяти машині стану, що обробляє потік, тобто головний потік, витрачати занадто багато часу на ці операції.

Шаблон: Відправка головним потоком (S MainThread)

Опис

Найпростіший метод реалізації потоку викликів сценарію - це реалізувати його повністю як один єдиний потік. Цей потік піклується про всі операції, включаючи надсилання та отримання подій. Ця картина проілюстрована на малюнку 3.22

Цей підхід зазвичай використовується при тестуванні на відповідність, коли поведінка тесту є єдиним потоком. Нитка може надсилати стимули до SUT або чекати відповідей від SUT, але ніколи одночасно. Очікуючи реакцій SUT, технічно тестова нитка спить короткий проміжок часу і час від часу прокидається, щоб перевірити, відповів чи ні SUT. Якщо SUT відповідає,

тестовий потік підтверджує відповідь і слідкує за ходом тестових дій. Якщо SUT не реагує, тестові потоки можуть прийняти рішення про встановлення тайм-ауту після декількох пробуджень.

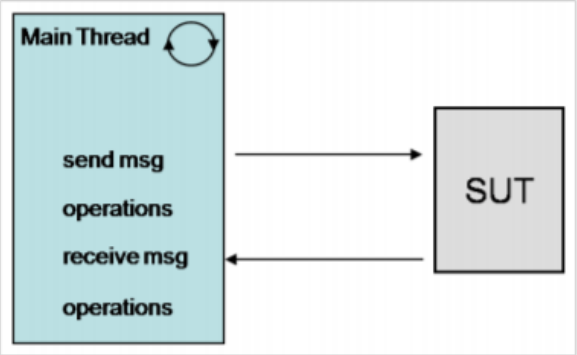

Рисунок **Error! No text of specified style in document.**.26– відправка повідомлень основним потоком

Переваги

Цей шаблон простий у використанні, доки всі операції надсилання обробляються основним потоком.

Недоліки

Ця модель не підходить для тестування працездатності з двох причин. По-перше, він пов'язаний із виконанням одного сценарію, тобто потік може обробляти лише повідомлення, які належать до одного сценарію через часові події, на які він повинен чекати. Після того, як потік повинна чекати відповіді, він повинен спати, поки потік не отримає повідомлення або поки не закінчиться час очікування. По-друге, для моделювання величезної кількості сценаріїв ця модель не є ефективною, оскільки кожен сценарій повинен бути примірником окремого потоку. Для великої кількості потоків TS стає повільним, оскільки слоти процесора повинні бути зарезервовані і для контекстних комутаторів потоків. [8]

Шаблон: Надсилання з окремим потоком за запитом (S SepThreadPerRequest)

Опис

Кращий підхід - використовувати окремого потоку для відправки операцій, коли ці операції вимагають занадто багато часу. Цей шаблон має перевагу в тому, що основний потік може працювати паралельно з

відправним потоком. Потік відправки або відмирає відразу після відправлення повідомлення, або також може бути використаний як потік очікування для обробки відповідей SUT. S SepThreadPerRequest, проілюстрований на рисунку 3.23, передбачає, що кожен запит відправлення обробляється окремим потоком відправки.

Переваги

Ця схема корисна для TS, які моделюють декілька користувачів, які обробляють тривалий запит / відповідь, наприклад запити до бази даних. Основний потік може виконувати багато інших операцій під час обробки операції надсилання.

Недоліки

Ця закономірність є менш корисною для короткочасних запитів через накладні витрати на створення нового потоку для кожного запиту. Він також може споживати велику кількість ресурсів ОС, якщо TS повинен імітувати багато користувачів, які одночасно роблять запити.

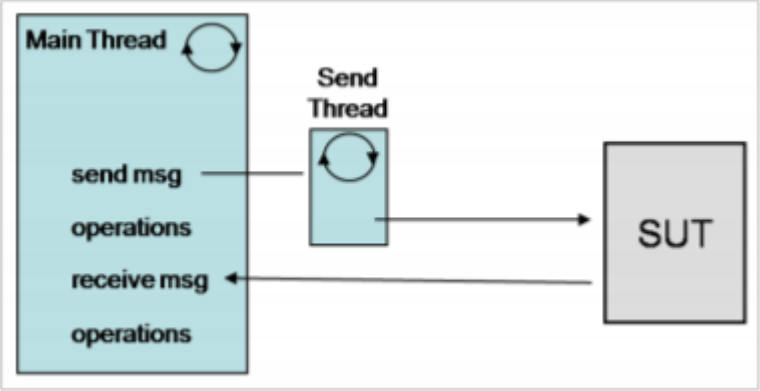

Рисунок **Error! No text of specified style in document.**.27– відправка за допомогою окремого потоку на запит

Відправка з окремим потоком на сеанс (S SepThreadPerSession)

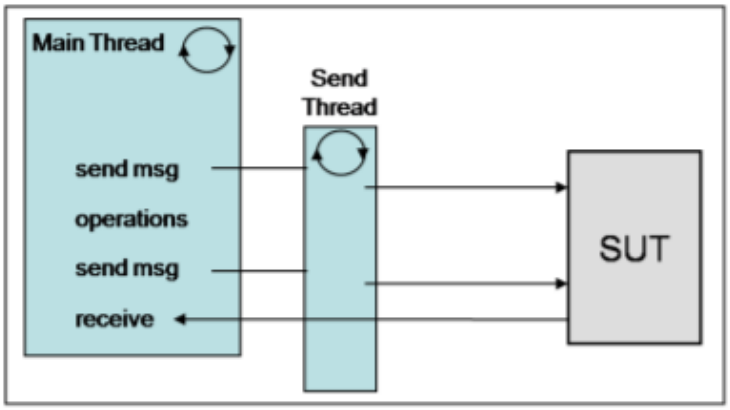

Рисунок **Error! No text of specified style in document.**.28– відправка за допомогою окремого потоку на сесію

Опис

S SepThreadPerSession є варіантом S SepThreadPerRequest, який компенсує витрати на створення потоку в декількох запитах. Це означає, що кожен користувач, модельований TS, обробляється окремим потоком протягом усієї тривалості сеансу, тобто операції надсилання обробляються одним і тим же потоком.

Переваги

Ця схема корисна для TS, які імітують декількох користувачів, які ведуть тривалі розмови.

Недоліки

Цей зразок не корисний для ТС, що імітують користувачів, які роблять лише один єдиний запит, оскільки це фактично модель "за потоком на запит".[8]

Шаблон: Надсилання з басейном потоків (S ThreadPool)

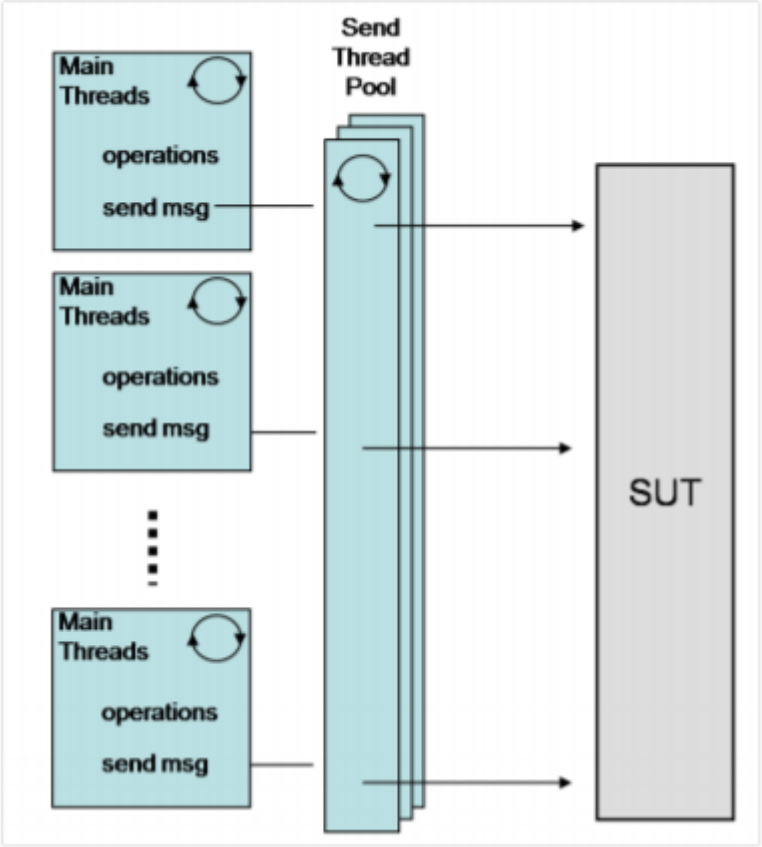

Рисунок **Error! No text of specified style in document.**.29– басейн потоків для відправки повідомлень

Опис

У схемі S ThreadPool, проілюстрованій на малюнку 3.25, створюється ряд N потоків для виконання ряду M завдань, як правило, організованих у чергу. Як правило, N набагато менший за M. Як тільки потік виконає своє завдання, він запитає наступне завдання з черги, поки всі завдання не будуть виконані. Потім потік може закінчуватися або спати, поки не з’являться нові завдання.

Переваги

Маючи на увазі дві описані вище схеми, модель пулу потоків може розглядатися як інша зміна потоку за запитом, яка також компенсує витрати на створення потоку за допомогою попереднього нерестування пулу потоків. Це корисно для TS, яким потрібно обмежити кількість ресурсів ОС, які вони споживають. Запити / відповіді SUT можна обробляти одночасно, поки кількість одночасних запитів не перевищить кількість потоків у пулі. У цей

момент додаткові запити повинні бути в черзі, потік нитка не стане доступним.

Перевага використання пулу потоків над створенням нового потоку для кожної задачі полягає в тому, що уникнення створення та знищення потоку, може призвести до кращої продуктивності та кращої стабільності системи. Реалізуючи цю схему, програміст повинен забезпечити безпеку потоку черги.

Недоліки

Однією з потенційних проблем є те, як налаштувати правильну кількість потоків (N), щоб час очікувань на виконання завдань у черзі був мінімальний. [8]

3.5.5 Шаблони отримання повідомлень

Подібно до надсилання повідомлень, існує кілька моделей для здійснення операцій прийому. Для прийому повідомлень потік приймача створює чергу повідомлень для того, щоб читати з неї щоразу, коли в неї вступають нові повідомлення.

Шаблон: Прийом з основним потоком (R MainThread)

Опис

У цій схемі, яка традиційно використовується для функціонального тестування, потік, який реалізує потік виклику сценарію, може використовуватися також для прийому повідомлень. При такому підході потік повинен зупинятися, поки не буде отримано повідомлення, тому його не можна використовувати для більш ніж одного потоку викликів. Отримавши нове повідомлення, потік перевіряє, очікується це чи ні.

Переваги

Цю модель легко застосувати до тих пір, поки операція прийому описана в основній поведінці тесту.

Недоліки

Цей підхід є простим, але не може бути використаний для реалізації великомасштабних TS через велику кількість потоків, необхідних для великої кількості користувачів.[8]

Шаблон: отримання з окремим потоком за сеанс (R SepThreadPerSession)

Опис

Для критичних завдань необхідно повернути управління потоком, а не чекати відповідей SUT. Таким чином, тестовий потік може вирішувати інші дії, наприклад, інші потоки сценарію. У цій ситуації створюється новий потік, який лише чекає відповідей SUT та повідомляє головний потік у випадку, коли щось отримано.

Переваги

Цей підхід потрібен особливо в тих випадках, коли перший потік повинен обробляти більше ніж один користувач. Перший потік надсилає запит для одного користувача і замість того, щоб чекати відповіді від SUT, він створює ще один запит для подальшого користувача. Другий потік слухає канали зв'язку для відповідей SUT.

Недоліки

Цей шаблон незручний тим, що для великої кількості користувачів потрібно створити величезну кількість потоків прийому.

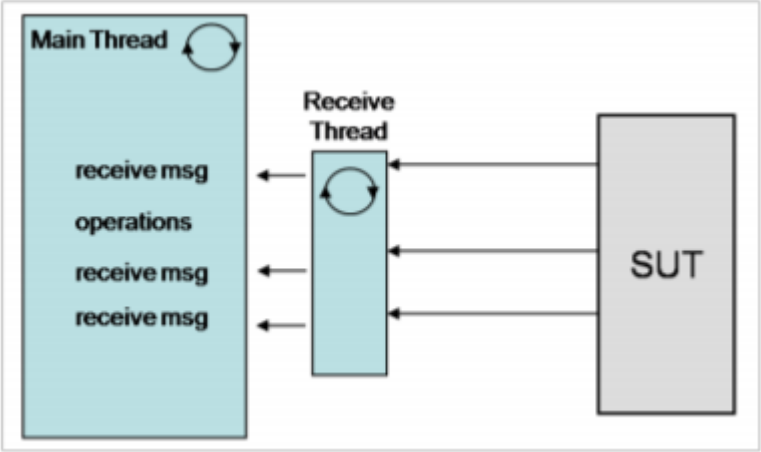

Рисунок **Error! No text of specified style in document.**.30– отримання повідомлень потоком на сесію

Шаблон: Прийом з басейном потоків (R ThreadPool)

Опис

Залежно від кількості відповідей, багато різних потоків можуть бути одночасно присутніми, що ускладнює управління потоком. Оскільки будь-який основний потік поведінки вимагає від другого потоку перевірки відповідей SUT, один потім може бути спільним між декількома основними потоками. Спільний потік паралельно прослуховує більше сокетів і кожного разу, коли щось отримано, він повідомляє відповідний основний потік.

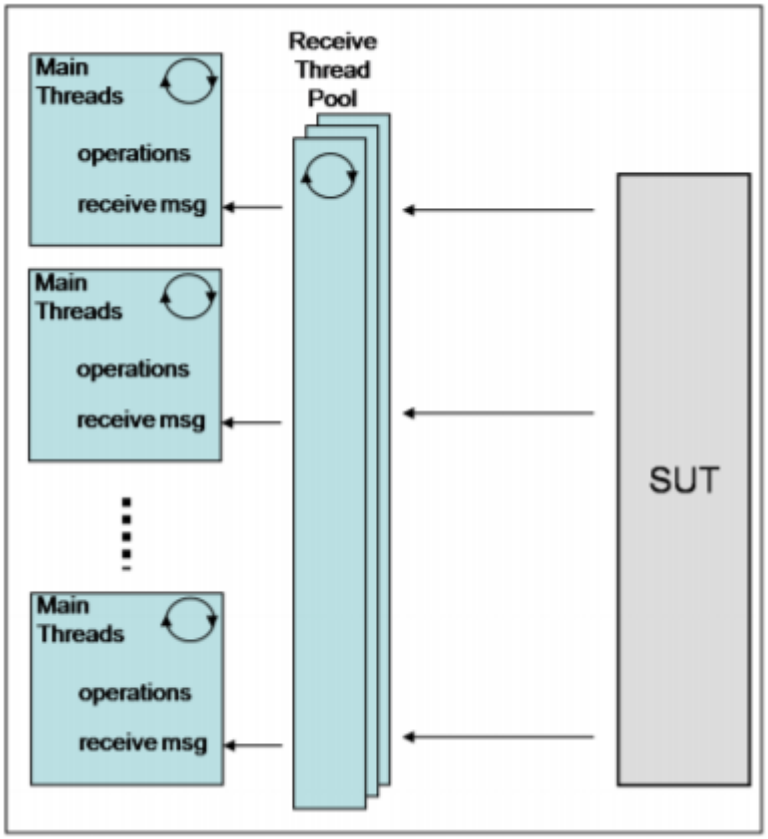

Рисунок **Error! No text of specified style in document.**.31– басейн потоків для отримання повідомлень

Ця закономірність вимагає механізму ідентифікації, що зазвичай називається демультиплексором події між отриманими повідомленнями та основними потоками. Демультиплексор події - це об'єкт, який передає події вводу-виводу з обмеженої кількості джерел відповідним обробникам подій. Розробник реєструє інтерес до конкретних подій та надає обробникам подій

або зворотній зв'язок. Демультиплексор події доставляє запитувані події обробникам подій.

Дві схеми, що включають демультиплексори подій, називаються Reactor, тобто, Receive with Thread Pool with Reactor (R ThreadPool Reactor) і Proactor, тобто, Receive with Thread Pool with Proactor (R ThreadPool Proactor).

Схема реактора підтримує демультиплексування та диспетчеризацію декількох обробників подій із синхронними подіями. Ця закономірність спрощує додатки, керовані подіями, інтегруючи синхронне демультиплексування подій та диспетчеризацію відповідних обробників подій. Демультиплексор передає цю подію відповідному обробнику, який відповідає за виконання фактичної обробки. Ця комунікація заснована на синхронних операціях, які повертають управління абоненту лише після завершення обробки.

Шаблон проактора підтримує демультиплексування та диспетчеризацію декількох обробників подій, які викликаються асинхронними подіями. Демультиплексор події ініціює асинхронні операції та відправляє ці події вперед відповідним обробникам, але не чекає завершення.

Переваги

Ця модель ефективно використовує апаратні ресурси, значно скорочуючи кількість потоків.

Недоліки

Цю закономірність складно реалізувати, оскільки вона потребує механізму ідентифікації, тобто демультиплексора події між отриманими повідомленнями та основними потоками.[8]

3.5.6 Шаблони керування навантаженням

Щоб контролювати інтенсивність навантаження, TS повинен контролювати кількість транзакцій, що працюють паралельно. Такий механізм називається керуванням навантаженням, і він зазвичай реалізується

у вигляді одного або декількох окремих процесів, які взаємодіють один з одним з метою збільшення, утримання або зменшення кількості взаємодій з SUT.

Повертаючись до загальної моделі потоку сценарію, на малюнку 3.3 роль контролера навантаження полягає у виборі користувачів та створенні нових викликів для тих користувачів, які виконують сценарії, представлені в цьому потоці. Це передбачає надсилання першого запиту, тобто Req1, для кожного користувача. Цей крок приведе користувача з наявного стану у стан, надісланий Req1, як представлено на малюнку 3.4. Щоб контролювати інтенсивність навантаження, контролеру навантаження потрібно керувати всіма подіями надсилання повідомлень типу Req1. З цієї причини регулятор навантаження також називають генератором навантаження.

Синхронізація паралельних потоків реалізується шляхом передачі повідомлень про координацію асинхронно або через синхронні віддалені виклики процедур. Взагалі потоки управління навантаженням потребують синхронізації на початку або на зупинці та при збільшенні або зменшенні рівня навантаження.

Загальним питанням контролю навантаження є точність часових позначок подій. Багато тестів на ефективність вимагають точного контролю часу кожної відправленої події та швидкості надсилання подій. Деякі тести на працездатність навіть вимагають точного розподілу часових позначок протягом часового інтервалу, наприклад, розподілу Пуассона.

Механізми контролю навантаження можуть мати централізовані дані або децентралізовані дані. Підхід до централізованих даних передбачає, що дані, необхідні для генерації навантаження, централізовані в одному місці, тому потоки генерації навантаження повинні отримувати дані з центрального місця. Підходи, засновані на децентралізованих даних, усувають накладні витрати синхронізації, так що кожен потік генерації навантаження здатний вирішити, як генерувати запити.

Шаблон: Генерація навантаження з одним генератором (LG SGen)

Опис

Найпростіший спосіб здійснити контроль навантаження - це мати лише один потік для створення навантаження. Один генератор навантажень може сам керувати навантаженням, дбаючи про час видачі подій.

Переваги

Це працює дуже добре, доки цей потік може генерувати все навантаження.

Недоліки

Недоліком цієї картини є те, що точність зменшується при більших навантажень.

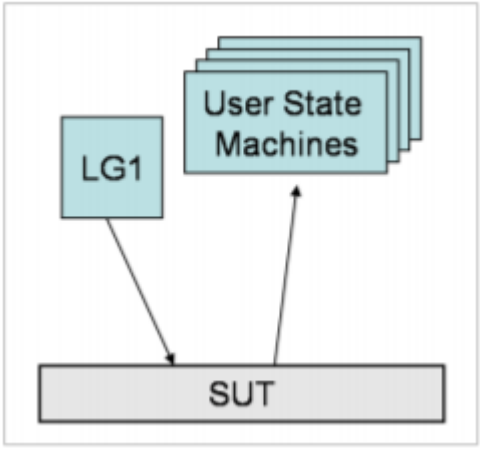

Рисунок **Error! No text of specified style in document.**.32– один генератор навантаження

Шаблон: генерація завантаження за допомогою декількох генераторів та централізованих даних (LG MGenCtrl)

Опис

У централізованому підході інформація про навантаження підтримується центральним об'єктом, який називається контролером навантаження. Контролер навантаження зберігає глобальний вигляд швидкості завантаження, що має бути реалізований, і контролює всі часові позначки, коли потрібно створювати події.

Механізм контролера навантаження може працювати або в режимі витягування, тобто завантаження генерації з декількома генераторами та централізованими даними за допомогою шаблону методу витягування (LG MGenCtrl Pull), в якому користувацькі потоки запитують контролер часових міток, коли надсилати нові події, або в режимі push. , тобто завантаження генерації з кількома генераторами та централізованими даними за допомогою шаблону методу Push (LG MGenCtrl Push), в якому контролер запитує користувацькі потоки, коли потрібно генерувати нові події.

Шаблон LG MGenCtrl Pull вимагає, щоб кожен потік генератора навантажень запитував контролер навантаження для міток часу, коли надсилати події. Контролер навантаження працює в цьому випадку як сервер, який забезпечує інтерфейс для вимогливих часових позначок.

LG MGenCtrl Push передбачає, що кожному генератору навантаження надається інформація, коли потрібно випустити нову подію. Це вимагає, щоб кожен генератор навантажень був готовий відправляти події на вимогу контролера навантаження. Контролер навантаження працює в цьому випадку диспетчером часових позначок.[8]

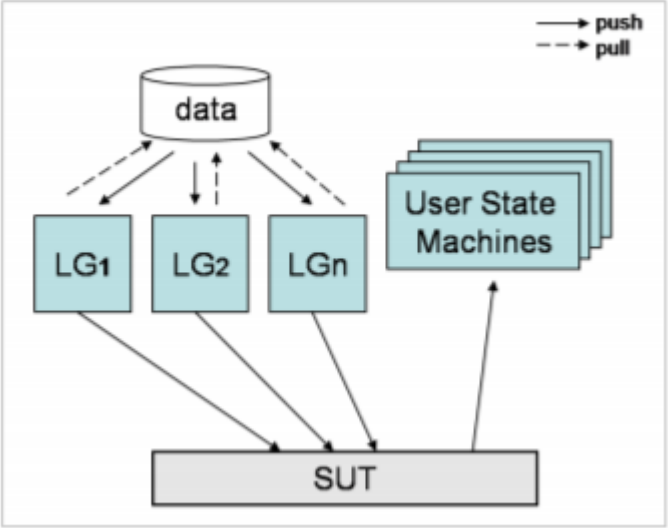

Рисунок **Error! No text of specified style in document.**.33– декілька генераторів навантаження з централізованими данними

Переваги

Генерація навантаження розподіляється на кілька контролерів навантаження, які врівноважують зусилля по створенню запитів. Ця закономірність забезпечує хорошу точність в часі.

Недоліки

Недоліком цих моделей є те, що дані зберігаються централізовано, тому потрібна синхронізація між потоками генератора навантаження.

Шаблон: Генерація завантаження за допомогою декількох генераторів та децентралізованих даних (LG MGenDectrl)

Опис

Управління навантаженням може бути реалізовано також децентралізованим способом, так що кожен потік генерації навантаження діє самостійно. Основна ідея полягає в інструментації потоку кожного покоління для того, скільки навантаження генерувати і як воно повинно генерувати це навантаження.

Переваги

Цей підхід є адекватним для розподіленого виконання, щоб уникнути накладних комунікацій, необхідних централізованим методом.

Недоліки

Одним з недоліків цієї схеми є те, що дані користувачів повинні бути розділені на менші кластери, щоб кожен генератор навантажень міг працювати зі своїми власними даними користувача.

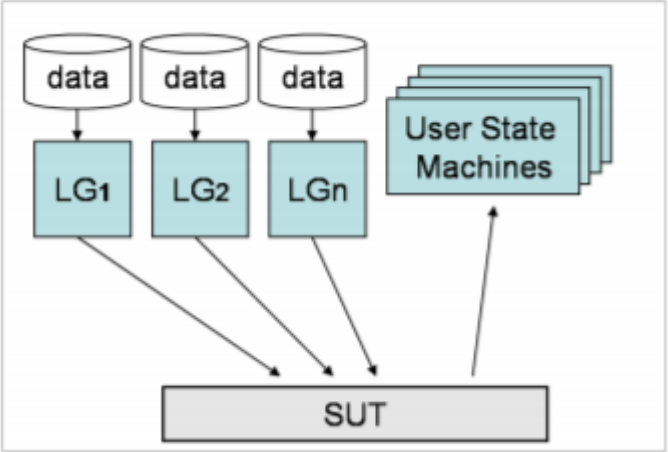

Рисунок **Error! No text of specified style in document.**.34– декілька генераторів навантаження з децентралізованими даними

3.5.7 Шаблони інкапсуляції даних

Тестові операції діють на повідомлення, обміняні з SUT. Зміст повідомлення зазвичай обробляється не в сирому вигляді, а у вигляді структури, яка забезпечує засоби доступу до його інформації. Механізм називається інкапсуляцією даних. До вмісту повідомлення можна отримати доступ через інтерфейс, який надає покажчики на менші частини повідомлення. У цьому розділі обговорюється кілька стратегій інкапсуляції та доступу до вмісту повідомлення. [8]

Шаблон: Представлення даних за допомогою буферного рядка (D StringBuffer)

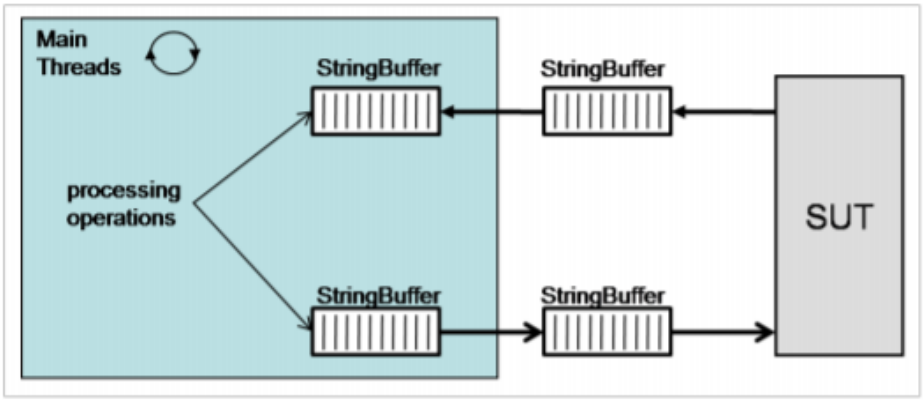

Рисунок **Error! No text of specified style in document.**.35– представлення даних за допомогою буферного рядка

Опис

Найпростіший метод інкапсуляції даних повідомлень - це зберігання їх у буфері рядків. Тому такий підхід обмежується протоколами, заснованими на рядках, наприклад, діаметром SIP. Буфер рядків дозволяє легко виконувати пошук та модифікацію через регулярні вирази. Завдяки своїй простоті цей шаблон легко інтегрувати в будь-яке середовище виконання. Він також не потребує подальшого кодування або розшифровки вмісту.

Переваги

Буфер рядків дозволяє легко виконувати пошук та модифікацію через регулярні вирази.

Недоліки

На жаль, обробка рядків коштує багато часу процесора. Тому шаблон не слід використовувати для великих повідомлень. Проста оптимізація - це розділити повідомлення на кілька рядків, які потім обробляються окремо.[8]

Шаблон: Представлення даних із мінімальним вмістом структури (D MinStrContent)

Опис

Згідно з цим шаблоном, вміст зводиться до мінімальної кількості інформації. В його основі лежить ідея витягти з повідомлення лише корисну інформацію та обробляти її окремо від решти повідомлення. Коли інформація повинна бути використана для створення нового повідомлення, її потрібно ввести лише в нове повідомлення. Ця закономірність вимагає двох операцій: декодування, тобто витяганя використаної інформацію з повідомлення та кодування, тобто ставить інформацію в нове повідомлення. Рис 3.32 ілюструє механізм цієї схеми.

Переваги

Цей шаблон застосовний до будь-якого типу протоколів, а не лише на основі рядків. Він працює ефективно, коли використана інформація невелика порівняно з розміром повідомлення.

Недоліки

Цей шаблон діє ефективно лише тоді, коли витягнута інформація є достатньою для створення нових повідомлень, тобто подальших повідомлень відповідно до протоколу. В іншому випадку, коли для створення відповіді потрібна додаткова інформація з вихідного повідомлення, оригінальне повідомлення також має бути збережене. Це трапляється, коли TS не потребує додаткової інформації, але може вимагати SUT.

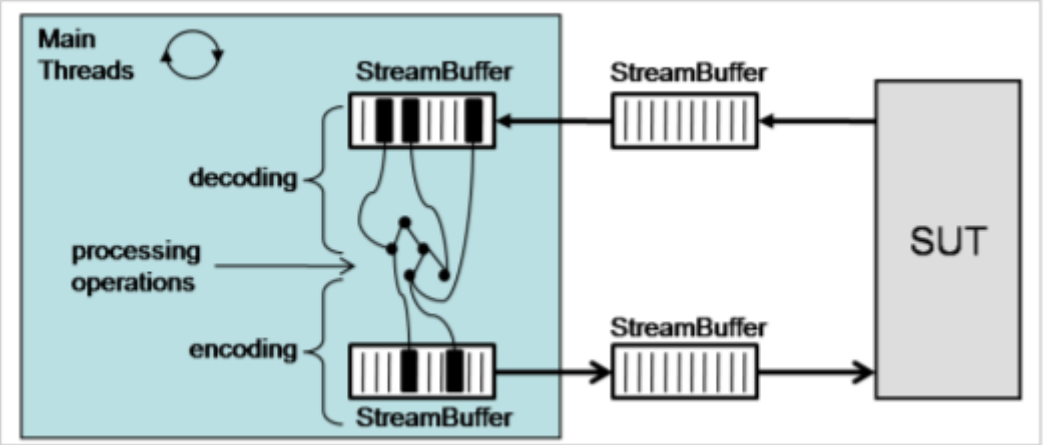

Рисунок **Error! No text of specified style in document.**.36– представлення даних за допомогою структури з мінімальним вмістом

Шаблон: Представлення даних з повним структурованим вмістом (D StrContent)

Опис

Ця модель вимагає, щоб вміст повідомлення був представлений у вигляді структури дерева, яка базується на специфікації повідомлення протоколу. Інформація витягується декодером, який проходить все повідомлення і будує дерево зображення. При операції надсилання дерево перетворюється назад в неочищене повідомлення шляхом проходження кожного вузла дерева.

Переваги

Великою перевагою, з точки зору тестування, є те, що представлення дерева полегшує доступ до кожної інформації. Крім цього, обробник подій може підтвердити будь-яку деталь, необхідну протоколом, наприклад, перевірити відповідність специфікації протоколу.

Недоліки

Операції з кодером та декодером можуть коштувати багато часу процесора, тому його не слід використовувати, якщо вся інформація не потрібна для тестування. Крім того, операція кодування може потребувати створення безлічі коротких живих об'єктів. У середовищі, що базується на Java, це може спричинити дуже щільне збирання сміття, завдяки чому всі потоки зупиняються, тим самим перериваючи інші важливі дії.[8]

Шаблон: Представлення даних як структуру покажчиків (D StrPointers)

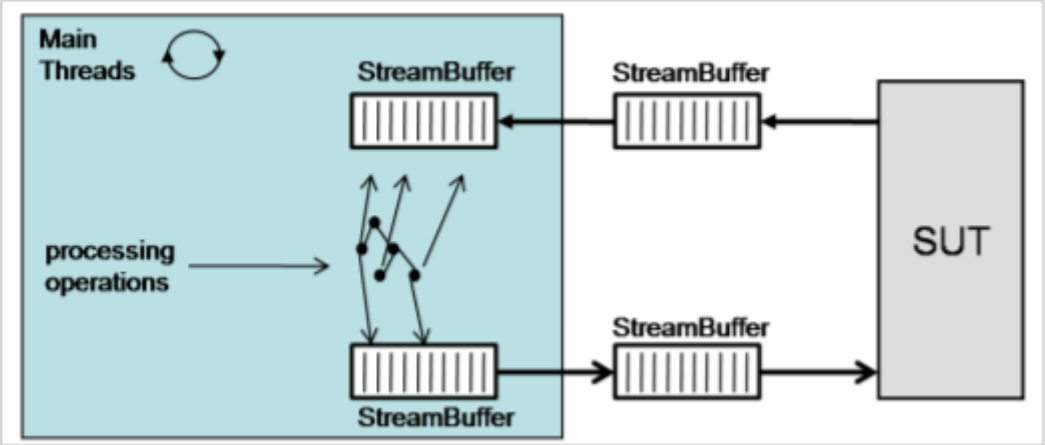

Рисунок **Error! No text of specified style in document.**.37– представлення даних як структури покажчиків

Опис

Цей шаблон є оптимізацією шаблону D StrContent. Ідея полягає в тому, щоб у структурі дерева зберігалися лише вказівники на місця, де вміст може бути доступний у вихідному повідомленні. Рисунок 3.33 ілюструє механізм цієї схеми. Цей шаблон все ще потребує CoDec, але він може бути реалізований ефективніше до тих пір, поки не потрібні короткі живі об'єкти, оскільки єдиним завданням є ідентифікація та зміна даних у визначених місцях у вихідному повідомленні.

Цей шаблон можна використовувати ще ефективніше, коли місця визначені заздалегідь. У тестовому навантаженні, де всі дані створюються на основі шаблонів генерації даних, локації можна обчислити заздалегідь лише на основі можливого вмісту повідомлення. Однак для цього потрібно, щоб

кожне поле мало фіксований розмір, наприклад, номер користувача завжди має сім цифр.

Переваги

Основна перевага цієї схеми - зменшене споживання процесора та пам'яті для кодування та декодування.

Недоліки

Цю закономірність складно реалізувати для багатьох протоколів.[8]

3.5.8 Шаблони басейну користувачів

До цього моменту поведінка тесту на ефективність описувалася як композиція всіх поведінок користувачів, де кожна поведінка користувача розглядається як певний тестовий сценарій. Однак все ж слід враховувати правила вибору користувачів та призначення їх певному типу сценарію, якщо користувач може запускати будь-який тип сценарію.

Користувач характеризується своєю особою та своїм станом. Користувач може перебувати у стані "доступного", що означає, що під час запуску сценарію може бути обраний користувач для запуску нового сценарію або іншого стану, відмінного від доступного. Коли користувач виконує сценарій, це означає, що всі транзакції, що беруть участь у сценарії, параметризовані відповідно до цієї ідентичності користувача.

Модельований користувач може створювати дзвінки різного типу. Крім того, користувач може запускати одночасні сценарії, наприклад, здійснювати голосовий дзвінок та одночасно надсилати повідомлення. При створенні нового дзвінка вибір користувача повинен бути довільним. Обраний користувач повинен мати можливість зателефонувати будь-якому іншому користувачеві, включаючи самого себе, наприклад, користувач може відправити миттєве повідомлення самому собі. Користувач може бути використаний кілька разів, якщо пробіг тесту досить довгий. Для коротких пробігів користувач може використатись двічі.

Набір користувачів, які беруть участь у тесті на працездатність, називається популяцією користувачів, а кількість користувачів може бути

розділена на кілька пулів користувачів. Пул користувачів групує набір користувачів, які призначені виконати подібне завдання, наприклад, певний тип сценарію, групу сценаріїв, випадок використання.

Будь-який тест на ефективність повинен визначати і підкорятися набору правил щодо способу вибору користувачів із пулу користувачів. Ось приклад кількох правил:

- Користувач - це машина стану, здатна імітувати повну поведінку обладнання користувача
- Користувачем може викликати або / і бути викликаним
- Користувач може створити більше одного дзвінка
- Користувача можна повторно використовувати для створення інших дзвінків
- Користувач може викликати випадковим чином будь-якого іншого користувача

"Активні" користувачі, які ініціюють сценарій та реагують на спробу сценарію, вибираються випадковим чином з одного пулу користувачів. Оскільки користувачів може бути вибрано довільно, логіка тестування повинна уникати побічних ефектів між різними сценаріями, що впливають на вимірювання, як це відбувається, наприклад, у Домен IMS, коли викликана сторона була незареєстрована, але вона була обрана для створення нового голосового дзвінка; таким чином, виклик буде невдалим, навіть якщо це не викликано SUT. [8]

Існує кілька моделей для впорядкування користувачів у пулах користувачів.

Шаблон: Населення з одним басейном (P SinglePool)

Опис

Найпростіша модель - використовувати єдиний пул, як представлено на рисунку 3.34. Усі екземпляри сценарію поділяють цей пул користувачів.

На жаль, при такому підході важко уникнути конфліктів між різними сценаріями, виконаними паралельно. Наприклад, для IMS SUT, який вимагає

попередньо зареєструвати користувачів, необхідні незареєстровані користувачі для сценарію реєстрації. Якщо користувач, який уже зареєстрований, знову обраний для реєстрації, це неправильний вибір.

Щоб уникнути конфліктів, ТС потребує більших обчислювальних ресурсів для проведення різних перевірок випадкових вибраних користувачів. Якщо користувач не виконує передумов, він відкидається, і буде обраний інший.

Цей шаблон впливає і на тестовий розподіл, оскільки штати користувачів повинні постійно оновлюватися, коли виклик переходить у новий стан. Якщо припустити, що кожен сервер зберігає локальний сховище для інформації про користувачів, потрібна додаткова комунікація між серверами для синхронізації та постійного оновлення локальних репозиторіїв.[8]

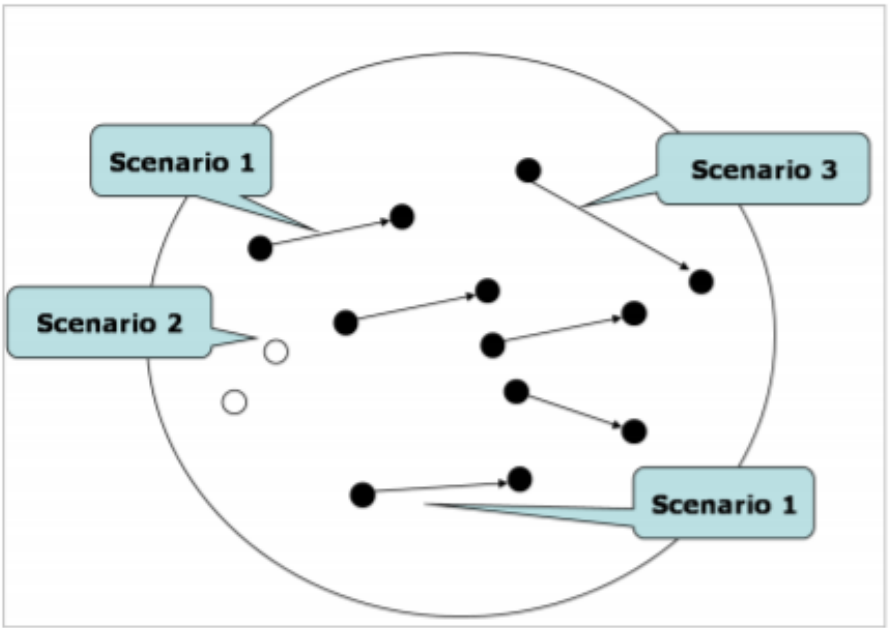

Рисунок **Error! No text of specified style in document.**.38– один басейн користувачів

Переваги

Цей шаблон легко реалізувати.

Недоліки

Цей шаблон може спричинити конфлікти між сценаріями. Щоб уникнути цих конфліктів, ТС потрібно проводити різні перевірки, щоб

перевірити, чи можуть обрані користувачі використовуватися чи ні в новому сценарії.

Шаблон: Населення з кластерами басейнів (P Clasters)

Опис

Кращий шаблон, який дозволяє уникнути конфліктів, описаних для одного шаблону пулу, визначає окремий пул для кожного сценарію, як проілюстровано на рисунку 3.35. На жаль, це менш реалістично, оскільки користувач обмежується лише одним типом сценарію протягом тривалості тесту.

Переваги

Ця модель має перевагу в тому, що вона дозволяє уникнути конфліктів між сценаріями з використанням одних і тих же користувачів.

Недоліки

Цей шаблон викликає нереальну поведінку через асоціацію користувачів до типового сценарію. Це означає, що користувач, пов'язаний з одним сценарієм, не може бути причетним до інших типів сценаріїв, крім того, з яким він був пов'язаний.

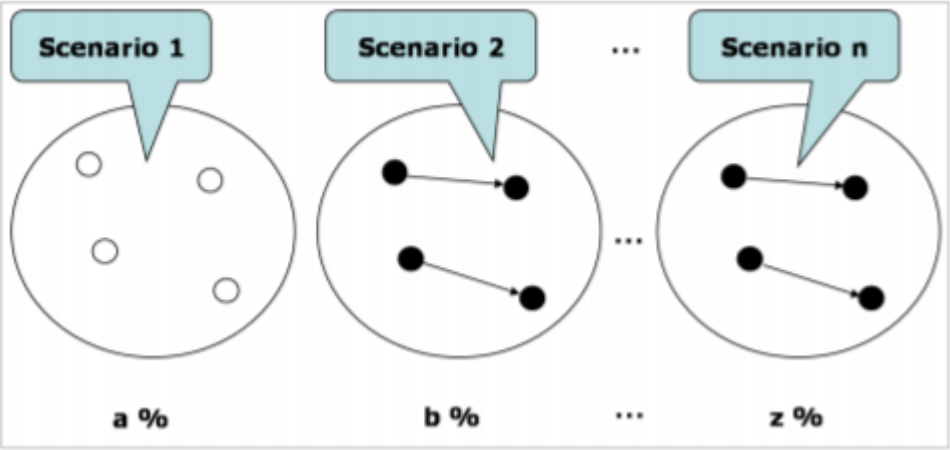

Рисунок **Error! No text of specified style in document.**.39– кластери басейнів користувачів за сценарієм

Шаблон: Населення з мінімальною кількістю кластерних басейнів (P MinClusters)

Опис

Цей шаблон заснований на поєднанні попередніх двох шаблонів, як це представлено на малюнку 3.36. Ця схема вирішує конфлікти між різними сценаріями, які використовують одні й ті самі користувачі, наприклад, скасування реєстрації та реєстрації в мережі IMS, створюючи окремі кластери для користувачів, які беруть участь у сценаріях, які можуть спричинити конфлікти. Усі інші сценарії, які не викликають конфліктів, мають один і той же кластер.

Переваги

Перевага цього шаблону полягає в тому, що він створює менше кластерів, ніж шаблон P кластерів.

Недоліки

Цей шаблон не є реалістичним, якщо всі сценарії повинні використовувати окремі кластери (вони можуть конфліктувати один з одним). Крім того, ця модель є більш складною для реалізації, оскільки необхідне спеціальне управління кластерами.[8]

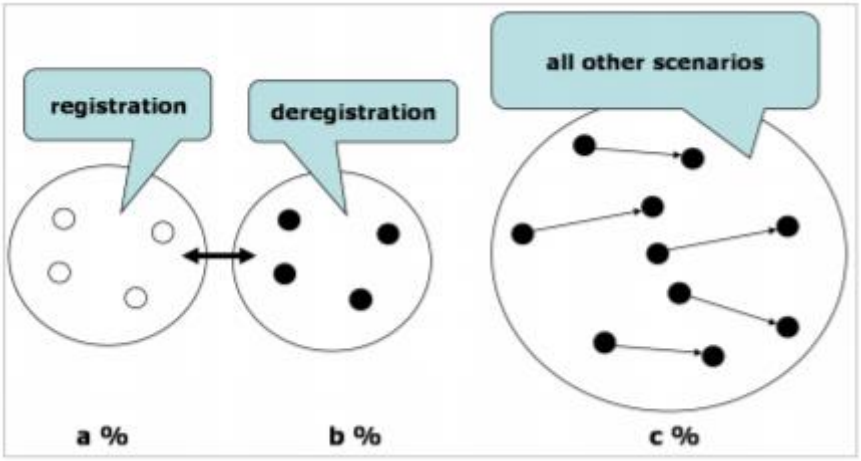

Рисунок **Error! No text of specified style in document.**.40– використання мінімальної кількості кластерів басейнів користувачів

3.5.9 Таблиця сумісності шаблонів

У попередніх розділах було введено та обговорено декілька моделей реалізації навантажень. Шаблони були згруповані в категорії, які залежать від функції, яку вони повинні виконувати, наприклад, шаблони для реалізації обробки подій. Як можна помітити, є моделі з однієї категорії, які не сумісні

з іншими з іншої категорії. Сумісність між моделями є важливим аспектом, який необхідно проаналізувати, перш ніж вибрати певну схему, щоб реалізувати навантаження.

Таблиця **Error! No text of specified style in document.**.3– таблиця

_	SM_SpecHdl	SM_GenHdl	UH_SingleUPT	UH_SeqUPT	UH_InterleavedUPT	T_Sleep	T_SepTT	S_MainThread	S_SepThreadPerRequest	S_SepThreadPerSession	S_ThreadPool	R_MainThread	R_SepThreadPerSession	R_ThreadPool_Reactor	R_ThreadPool_Proactor	LG_SGen	LG_MGenCtrl_Push	LG_MGenCtrl_Pull	LG_MGenDectrl
SM_SpecHdl	*		+	+		+	+	+	+	+	+	+	+	+	+	+	+	+	+
SM_GenHdl		*	+	+	+		+	+	+	+	+	+	+		+	+	+	+	+
UH_SingleUPT			*			+	+	+	+	+	+	+	+	+	+	+	+	+	+
UH_SeqUPT				*		+	+	+	+	+	+	+	+	+	+	+	+	+	+
UH_InterleavedUPT					*		+	+	+	+	+		+		+	+	+	+	+
T_Sleep						*		+	+	+	+	+	+	+	+	+	+	+	+
T_SepTT							*	+	+	+	+	+	+	+	+	+	+	+	+
S_MainThread								*				+	+	+	+	+	+	+	+
S_SepThreadPerRequest									*			+	+	+	+	+	+	+	+
S_SepThreadPerSession										*		+	+	+	+	+	+	+	+
S_ThreadPool											*	+	+	+	+	+	+	+	+
R_MainThread												*				+	+	+	+
R_SepThreadPerSession													*			+	+	+	+
R_ThreadPool_Reactor														*		+	+	+	+
R_ThreadPool_Proactor															*	+	+	+	+
LG_SGen																*			
LG_MGenCtrl_Push																	*		
LG_MGenCtrl_Pull																		*	
LG_MGenDectrl																			*

сумісності шаблонів

На таблиці 3.3 проаналізовано сумісність між представленими шаблонами. Використовується просте позначення: знак "+" позначає пари шаблонів, сумісних. Очевидно, що шаблони з однієї категорії не сумісні один з одним, оскільки їх не можна використовувати одночасно в одній і тій же реалізації.

Оскільки структури генерації навантаження складаються з окремих потоків, які не заважають решті потоків, що використовуються для обробників стану користувачів, таймерів, відправників, приймачів тощо, вони сумісні з усіма категоріями. Тому вибір схеми генерації навантаження не залежить від способу реалізації поведінки користувача.

Конкретний шаблон обробки машини стану не може бути використаний з переплетеною схемою обробки користувачів, тоді як загальний підхід до обробки стану не сумісний ні з схемою прийому повідомлення реактора, ні з схемою таймерів на основі сну.

Інший набір несумісностей помітний для впорядкованого шаблону обробки користувачів. Це не сумісно з тимчасовою схемою на основі сну, що приймає в основній схемі потоку та схемою прийому повідомлення на основі реактора.

Висновки до розділу 3

У цьому розділі виділено складові частини TS. У першій частині описаний спосіб проектування відповідних робочих навантажень та перевірок працездатності для мультисервісних систем. У другій частині глави досліджуються шаблони, що застосовуються при здійсненні тестів на ефективність. Ці шаблони розглядаються як загальні вказівки щодо того, як здійснити тест, спираючись лише на абстрактні артефакти, такі як потоки, машини стану користувачів та операції з обробки даних. Незважаючи на те, що в цьому розділі методологія, як такова, наводилась для мультисервісних систем – її використання для веб-сервісів буде описане в наступному розділі. Це говорить про те, що ця методологія навантажувального тестування з її шаблонами підходить для тестування різних систем. Також наведені шаблони не є кінцевими і можуть бути доповнені виходячи з нових розробок процесорів тощо.

4 ЕКСПЕРИМЕНТАЛЬНЕ ВИКОРИСТАННЯ МЕТОДОЛОГІЇ ТЕСТУВАННЯ НАВАНТАЖЕННЯ WEB-СЕРВІСІВ

4.1 Веб-сервіс

Мовою програмування для створення Веб-сервісу для подальшого тестування була вибрана мова Java, як одна з найпопулярніших на сьогоднішній час мов програмування у світі. Java – це об’єктно-орієнтовна мова програмування. Ця мова програмування є незалежною від платформ та архітектури. Під «незалежністю» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Цього можна досягти, компілюючи початковий Java код у байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

В даній роботі було побудовано REST веб-сервіс. Інтерфейсом веб-сервіса було обрано стандартний JAX-RS інтерфейс для REST застосунків. Цей інтерфейс описує стандартні REST API запити. Для реалізації цього інтерфейсу використовується бібліотека Jersey. Проект використовує Maven як засіб автоматизації зборки та управління. Контейнером сервлетів виступає Apache Tomcat. Веб-сервіс буде розташований на локальному комп’ютері, де і буде проходити тестування. А це означає, що ми усуваємо будь-які залежності від мережі тощо.

Як бачимо – в цьому проекті використані стандартні відкриті бібліотеки та застосунки, що базуються на мові програмування Java.

Застосунком для тестування навантаження було обрано Jmeter через те, що він є безкоштовним, має великий рівень використання серед розробників та тестувальників та через його гнучкість.

В системі клієнт-сервіс саме Jmeter буде фігурувати клієнтом, що виконує запити до сервісу сподіваючись на відповідь.

```
@Path("/index")
public class JMeterTest {
    @GET
    @Produces("text/html")
    public String checkECV() throws InterruptedException, FileNotFoundException, IOException {
        String result = "<br><div align='center'><h2>Hey This is JMeter Test...</h2></div>";

        System.out.println(result);
        Thread.sleep(1000);
        return result;
    }
}
```

Рисунок **Error! No text of specified style in document.**.41– лістинг основного методу веб-сервісу

Створений веб-сервіс буде мати в своєму арсеналі лише один метод, що зображений на Лістингу 4.1. При спробі зайти на http://localhost:8080/RESTJerseyExample/tz-z83mp/index/ ми будемо бачити те, що зображено на рисунку 4.2.

Hey This is JMeter Test...

Рисунок **Error! No text of specified style in document.**.42– зображення роботи веб-сервісу

4.2 Експериментальне використання методології навантажувального тестування для Веб-сервісу

В цій роботі буде використовуватись звичайний користувацький випадок. За допомогою Jmeter буде імітуватись поведінка користувачів, створюватись профіль навантаження тощо.

Через простоту створеного Веб-сервісу будуть використані такі шаблони тестування навантаження:

Шаблони машини стану користувачів – відсутня як така. Користувачі, як такі, не мають станів і лише виконують 1 запит.

Шаблони обробки користувачів – буде використаний шаблон одного користувача на один потік. Це є стандартна пропозиція застосунку Jmeter. Вона нас задовольняє через нескладну архітектуру сервісу.

Шаблони таймерів – таймери не будуть використовуватись під час цього тестування. Сценарій не буде відправлятись до статусу «відмова» при закінченні певного проміжку часу.

Шаблон відправки та отримання повідомлення – при цьому тестуванні буде використовуватись шаблон відправки та отримання повідомлення в головному потоці.

Шаблон керування навантаження – буде використовуватись шаблон з використанням лише одного генератора навантаження у вигляді Thread Group в Jmeter.

Шаблон інкапсуляції даних – не використовується. Дані не інкапсулюються в тестованій системі.

Шаблон басейну користувачів – не використовується, бо в цьому навантажувальному тесті використовується лише один сценарій, що немає поняття користувача як такого.

Для даного тестового сценарію для початку буде використовуватись крововий профіль навантаження, наведений на рисунку 4.3. Максимальна кількість потоків – 200. Будемо використовувати 4 кроки, кожен по хвилині.

Для початкових результатів використаємо графік транзакцій за секунду, де вказано кількість помилок та вдалих транзакцій, як видно на рисунку 4.4.

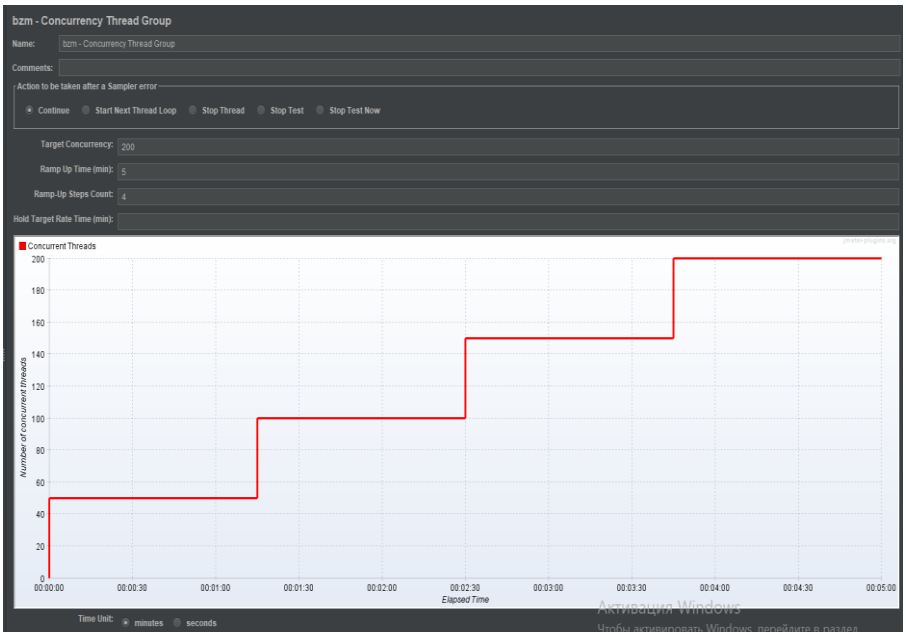

Рисунок **Error! No text of specified style in document.**.43– кроковий профіль навантаження що використовується у тестуванні

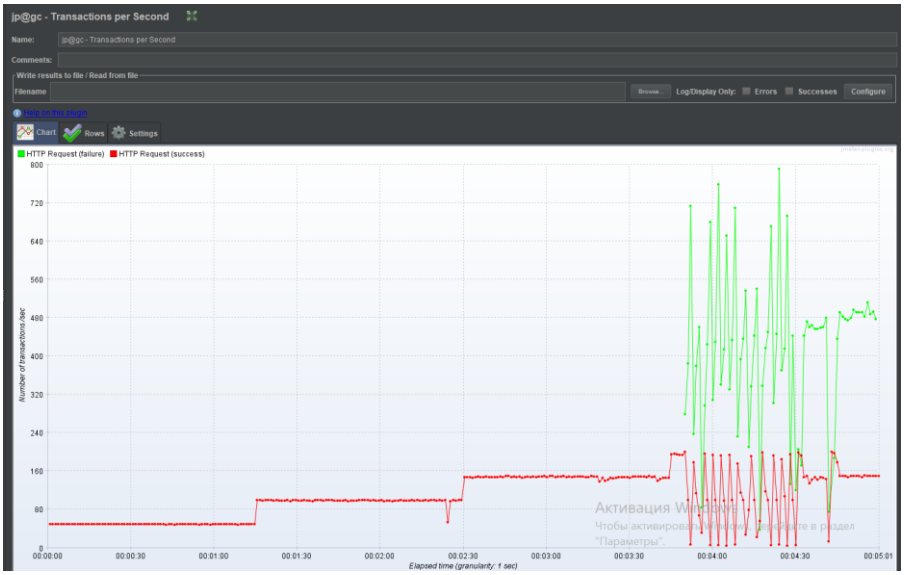

Рисунок **Error! No text of specified style in document.**.44– графік кількості транзакцій за секунду

Як видно при 200 потоках система виходить за ІНС<0.1% (в нашому випадку це відсоток помилок серед усіх запитів). Спробуємо виконати процедуру навантажувального тестування для визначення DOC. Було запущено декілька тестів на 180, 160, 140 та 130 потоків згідно з кроками, описаними з процедурою у розділі 3.4.4. Емпірично було визначено, що побудована система не витримує навантаження більше 130 потоків протягом

чотирьох хвилин. Всі досліди вказували, що система починає неадеквотно реагувати після 1-2 хвилини. У випадку, коли навантаження знаходилось на рівні 140-180, утворювалось 2 інтервали, де система переставала адекватно відповідати на запити, проте через деякий час система відновлювалась.

Рисунок **Error! No text of specified style in document.**.45– графік кількості транзакцій при 180 потоках і 57% помилок

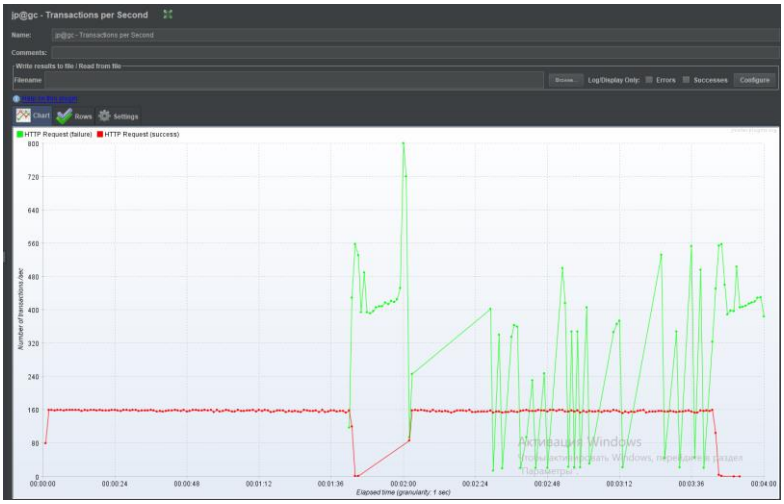

Рисунок **Error! No text of specified style in document.**.46- графік кількості транзакцій при 160 потоках і 43% помилок

Рисунок **Error! No text of specified style in document.**.47- графік кількості транзакцій при 140 потоках і 8% помилок

Рисунок **Error! No text of specified style in document.**.48- графік кількості транзакцій при 130 потоках і 0 помилок

Навантаження, яке дозволяє системі мати ІНС<0.1% є навантаженням у 130 потоків. Саме таке навантаження є оптимальним для роботи системи, тобто таким, що буде задовольняти користувачів. При наступних етапах розробки цього веб-сервісу, необхідно слідкувати за цим показником, адже якщо він буде падати – це означало б, що система деградує.

Під час навантажувального тестування побудованого веб-сервісу також проводилась оцінка роботи центрального процесору. Було видно, що інтервали тесту, коли тестована система починала у великій кількості збільшувати коефіцієнт помилок, припадали на збільшення відсотку використання центрального процесору процесом Java на комп’ютері, а саме з 1-2% до 70%. Тому можливо припускати, що створення такої кількості

користувачів використовуючи вибрані шаблони не є максимально ефективним для процесора і системи тестування загалом, як і було описано в недоліках до них.

Були проведені досліди з більшим навантаженням на систему, проте з меншим часом проходження тесту і було виявлено, що при часі проходження тесту меншим за хвилину при навантаженні в 1000 потоків можливо отримати результати при відсотку помилок в 0%, проте при такому часі проходження це не є репрезентативним.

Також був проведений другий експеримент з сайтом google.com. Було виявлено, що після невеликого навантаження протягом двох хвилин сайт почав блокувати всі запити від Jmeter. Таким чином вони блокують можливі DDOS атаки.

Проведемо третій експеримент, коли тестована і тест системи знаходяться не на одній машині. Для цього скористаємось безкоштовним хостингом від Google – Google Cloud. Необхідно модифікувати створений сервіс, підлаштувавши його до стандартного вигляду Google App Engine. Суть сервісу не змінилась, його вихідний код – також. Були додані лише файли для того, щоб «збудувати» проект в хмарі. Тепер шлях до нашого сервісу знаходиться по посиланню https://perf-tst.appspot.com/index.

Це було зроблено для того, щоб розділити тестовану і тест системи, коли неможливо зрозуміти, кому з них не вистачає потужностей ЦП.

В цьому експерименті використані всі ті самі кроки, що і в першому експерименті. Для початку – східчастий профіль навантаження. Як видно на рисунку 4.9 – результат є дуже схожим.

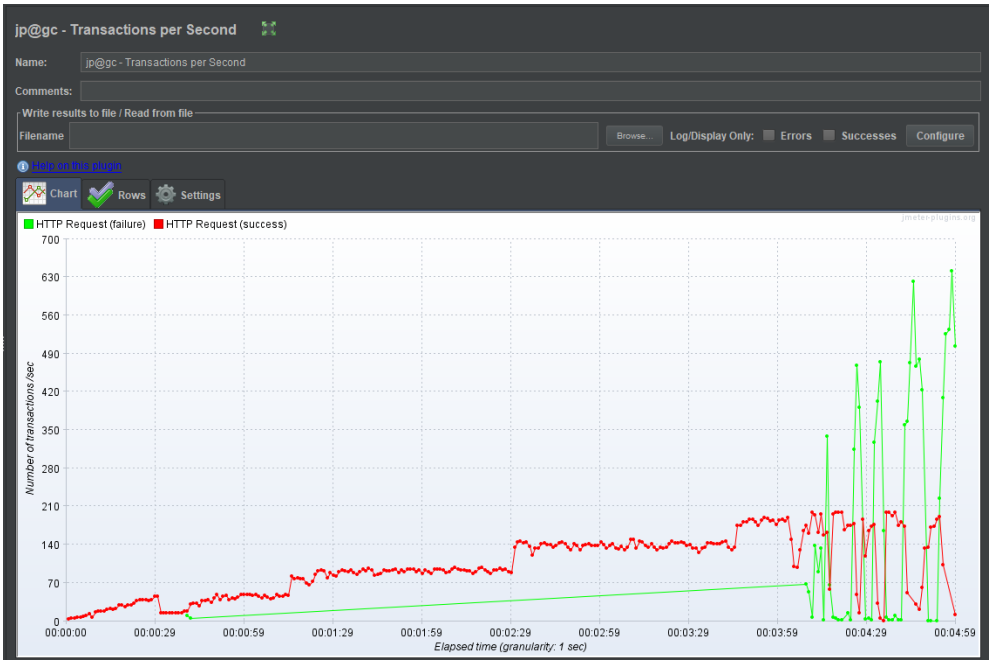

Рисунок **Error! No text of specified style in document.**.49- східчасте навантаження на веб сервіс у хмарі

Рисунок **Error! No text of specified style in document.**.50- графік кількості транзакцій при 180 потоках і 42% помилок

Рисунок **Error! No text of specified style in document.**.51- графік кількості транзакцій при 160 потоках і 25% помилок

Рисунок **Error! No text of specified style in document.**.52- графік кількості транзакцій при 140 потоках і 0% помилок

Як видно – розмежування тестованої системи та системи тестування на різні платформи збільшило загальне навантаження, яке може витримати тестована система. Проте разом з цим можна дійти висновку, що неправильні відповіді від тестованої системи під великим навантаженням були викликані не тестованою системою, а системою тестування. Провівши

третій дослід було визначено, що саме навантаження центрального процесора відповідає за ці помилки.

Змінивши конфігурацію системи тестування використовуючи більш оптимізовані шаблони – було проведено останній дослід. Був змінений шаблон отримання повідомлень на R ThreadPool.

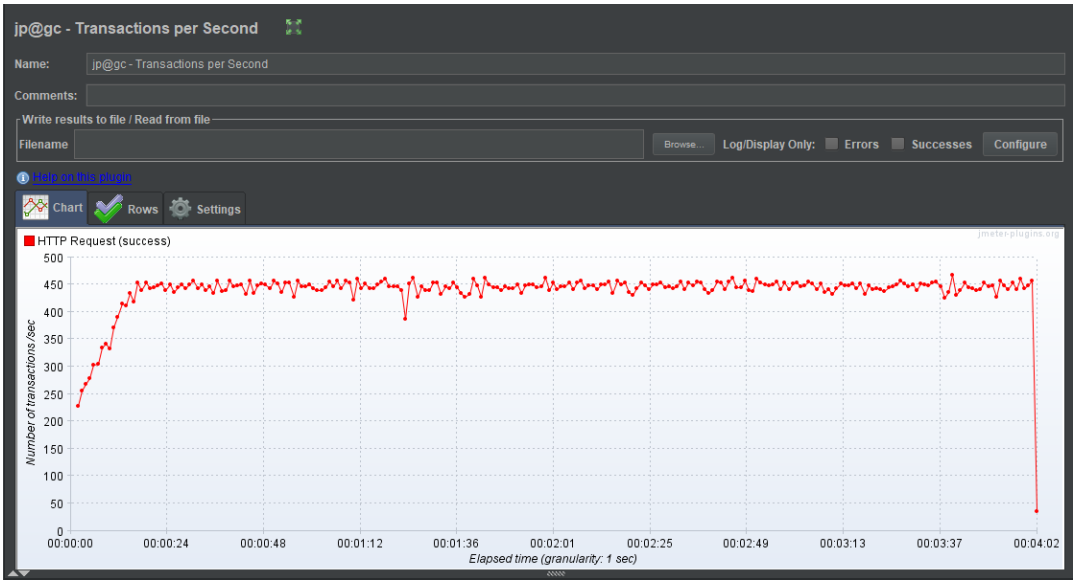

Рисунок 4.13 – тестування системи після зміни шаблону тестування

Система не показала жодної помилки під час тестування системи протягом 240 секунд використовуючи 500 потоків (користувачів). Це значно підвищує ефективність системи тестування.

Висновки до розділу 4

Отже, в цьому розділі було створену елементарний веб-сервіс на один метод, використовуючи стандартні методи побудови веб-сервісів стеком технології Java. Цей стек технологій був обраним через його популярність, безкоштовність та велику кількість користувачів. Системою тестування було обрано програму Jmeter через схожі причини.

Для тестування цього сервісу було використано запропоновану в минулому розділі процедуру та шаблони. Було обрано найпростіші шаблони для реалізації, чий вибір був обґрунтований складністю побудованого веб-сервісу та стандартною реалізацією в побудованій тестовій системі на основі програми Jmeter.

Було виявлено, що при даній конфігурації тестової та тестованої системи оптимальним навантаженням є 130 користувачів, де кожен користувач використовує окремий потік.

Також було виявлено, що при перенесення веб-сервісу на іншу платформу – у хмару сприятливо впливає на навантаження, яке може витримати тестована система, враховуючи результати. Проте разом із цим було виявлено, що неправильні відповіді були отримані системою тестування через неможливість обробити таке навантаження.

Вибрані шаблони для тестування навантаження довели свої мінуси та плюси. Вони є легкими в реалізації, проте на їх реалізацію потрібні великі потужності. Щоб уникнути таких результатів, можливе використання хмари також і для системи тестування, проте цього може бути недостатньо, якщо тестована система достатньо складна, є велика кількість складних сценаріїв і використовується велике навантаження.

5 ЕКОНОМІЧНА ВИГОДА

ІТ сфера в Україні є однією з найуспішніших і найстабільніших галузей, що приносить країні велику кількість робочих місць та суми податків. Ця галузь стрімко розвивається, використовує новітні технології менеджменту, тестування та розробки застосунків.

На ринку існує безліч схожих за своїми цілями сайтів, програм та іншого роду додатків і кожен має за мету отримати більшу аудиторію. Як уже було наведено в цій роботі – працездатність застосунку є однією з головних нефункціональних характеристик системи, що прямо впливає на зацікавленість користувача системою.

Також, як було визначено в даній роботі, навантажувальне тестування має в своїй основі безліч варіантів виконання. Можливо використовувати велику кількість застосунків та мов програмування і кожне рішення має свою ціну і свої цілі. Ця робота надає перелік шаблонів, процедур та методологію для проведення навантажувального тестування над системою.

За допомогою наведеного в роботі дизайну навантажувального тестування створюється покрокова інструкція для того, щоб отримати максимальну кількість корисних метрик для майже будь якого застосунку, адже при створення цього дизайну використовувались абстрактні поняття.

Це в свою чергу дозволяє говорити про економію проведення тестування навантаження системи будь якого розміру.

6 ЗАГАЛЬНІ ВИСНОВКИ

В цій роботі було наведено різні типи тестування, було визначено методологію тестування навантаження веб-сервісів на основі мультисервісних систем, визначено шаблони та загальні процеси тестування навантаження.

Також було створено простий веб-сервіс за допомогою стеку технологій на основі мови програмування Java та використано методологію відносно нього. Було доведено, що запропонована в третьому розділі методологія підходить як для складних мультисервісних систем так і для веб-сервісів, адже використовує загальні поняття, що притаманні обом системам.

Було визначено підходящі для тестування створеного веб-сервісу шаблони та згідно з запропонованим процесом було проведено тестування навантаження. Згідно з цим тестуванням, було визначено навантаження, яке було придатним для адекватної роботи системи згідно з результатами системи тестування як для веб-сервісу, що знаходився з системою тестування на одній платформі – так і у випадку з різними платформами.

Було доведено як плюси, так і мінуси обраних шаблонів тестування навантаження таких як простота в реалізації та надмірне використання ресурсів. Також було наведено приклад можливого усунення даної проблеми шляхом збільшення потужностей системи тестування шляхом перенесення системи тестування в хмару.

Також, як видно з результатів експериментів – основні ресурси системи були використані саме на створення навантаження, а не його на обробку тестованою системою. А це доводить, що необхідно використовувати усі можливі методи оптимізації тестованої системи, як ті, що наведені в методології.

Запропоноване рішення дозволяє підвищити декілька показників телекомунікаційної системи, а саме - підвищується ефективність системи тестування навантаження (збільшується кількість навантаження, що

генерується системою тестування) шляхом надання способів її організації та використання загальних шаблонів проектування модулів цієї системи. Також використання запропонованої методолії для тестування веб-сервісів дозволяє підвищити швидкість виконання та зменшити вартість проведення процесу тестування навантаження обраної системи шляхом використання зазначеної методології.

Ефективність складних системи тестування підвищується до близько 40% тільки шляхом зміни одного шаблону отримання повідомлень із R SepThreadPerSession на R ThreadPool і може бути підвищена до 300% при зміні шаблони інкапсуляції повідомлення [8]. В цій роботі лише зміною одного шаблону отримання повідомлень кількість потоків, що виробляється системою тестування збільшилась більше ніж у два рази (з 150 до 500 потоків).

Схожі показники отримуються шляхом перегляду та зміни також і інших шаблонів при використанні їх для побудови системи тестування веб-сервісів.

Посилання

1. Машнин Т. Web-сервисы Java [Електронний ресурс] / Т. Машнін. – 2012. – Режим доступу до ресурсу: http://scanlibs.com/web-servisyi-java/
2. EAPL/ ElysiumAcademy Private Limited What is Spike Testing in Software Testing? [Електронний ресурс] – 2018. – Режим доступу до ресурсу: https://www.linkedin.com/pulse/what-spike-testing-software-elysium-academy-private-limited/
3. Margaret Rouse Performance testing [Електронний ресурс] – 2007. – Режим доступу до ресурсу: https://searchsoftwarequality.techtarget.com/definition/performance-testing
4. Michael Punsky 7 Types of Web Performance Tests and How They Fit Into Your Testing Cycle [Електронний ресурс] – 2012. – Режим доступу до ресурсу: https://blog.smartbear.com/software-quality/7-types-of-web-performance-tests-and-how-they-fit-into-your-testing-cycle/
5. OctoPerf Performance, load and stress testing explained [Електронний ресурс]. – Режим доступу до ресурсу: https://octoperf.com/blog/2018/04/09/performance-load-stress-testing-explained/
6. Stability Testing [Електронний ресурс]. – Режим доступу до ресурсу: https://strongqa.com/qa-portal/knowledge-base/testing-types/performance-testing/stability-testing
7. Аналіз методів тестування навантаження інформаційних ресурсів веб-сервісів Лобода І.І.
8. George Din A Performance Test Design Method and its Implementation Patterns for Multi-Services Systems / George Din. – 2009.