Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Центр післядипломної освіти_____
(повна назва)

Кафедра_____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження методів автоматизованого тестування для виявлення найбільш ефективного
(тема)

Виконав: студент 2 курсу, групи ІПЗмзд-18-1___
_____Маковоз О.М._____
(прізвище, ініціали)

спеціальності 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

_____Освітньо-наукова програма_____
(тип програми)

__Інженерія програмного забезпечення_____
(повна назва освітньої програми)

Керівник ___доцент каф. програмної інженерії Чуприна А. С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет_Центр післядипломної освіти_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти – другий (магістерський)

Спеціальність _121 – Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми освітньо- наукова програма

Освітня програма_Інженерія програмного забезпечення_____
ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Маковоз Олександрі Миколаївні_____
(прізвище, ім'я, по батькові)

1. Тема роботи _ Дослідження методів автоматизованого тестування для виявлення найбільш ефективного_____
затверджена наказом університету від "___" _____20 ___ р. № _____
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії 16 травня 2020 р.
3. Вихідні дані до роботи *методи автоматизованого тестування навантаження*_____
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, огляд методів автоматизованого тестування*_____
_____

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. каф. ПІ Чуприна А.С		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка*
1.	Аналіз предметної галузі	9.02.2020	
2.	Огляд існуючих методів	02.03.2020	
3.	Методи автоматизованого тестування	16.03.2020	
4.	Підготовка пояснювальної записки	3.04.2020	
5.	Спецчастина	20.04.2020	
6.	Підготовка презентації та доповіді	28.04.2020	
7.	Попередній захист	9.05.2020	
8.	Нормоконтроль, рецензування	9.05.2020	
9.	Занесення диплома в електронний архів	9.05.2020	
10.	Допуск до захисту у зав. кафедри	19.05.2020	
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання _____ _____ 2019 р.

Студент _____
(підпис)

Керівник роботи _____ __ доц. каф. ПІ Чуприна А.С._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 56 с., 23 рис., 4 таблиці, 27 джерел.

ТЕСТУВАННЯ, JAVA, ПЕРФОРМАНС, HTTP, ПРОДУКТІВНИСТЬ, REST, SCALA, GATLING, APACHE JMETER.

Метою роботи є дослідження методів автоматизованого тестування продуктивності для виявлення найбільш ефективного.

В дослідженні використовуються інструменті тестування Apache Jmeter та Gatling з використанням мови розробки Scala, протоколу передачі даних HTTP, засобу автоматизованої роботи з програмними проектами Apache Maven.

В результаті роботи розглянуто методи моніторингу продуктивності, алгоритми аналізу на базі веб сервісу з HTTP протоколом та REST архітектурою.

TESTING, JAVA, PERFORMANCE, HTTP, PERFORMANCE, REST, SCALA, GATLING, APACHE JMETER.

The aim of the research is to determine the methods of automated performance testing to identify the most effective.

The study uses testing tools Apache Jmeter and Gatling using the development language Scala, HTTP data transfer protocol, a tool for automated work with software projects Apache Maven.

As a result, the methods of performance monitoring, analysis algorithms based on a web service with HTTP protocol and REST architecture are considered.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

QA – Quality Assurance (забезпечення якості)

IMS – IP Multimedia Subsystem (мультимедійна підсистема на основі протоколу IP)

SUT – System under Test (система, що тестується)

CI/CD – Continuous Integration / Continuous Delivery or Continuous Deployment (безперервна інтеграція / постійної доставки, або постійне розгортання)

ВСТУП

У сучасному світі розробки програмного забезпечення важливе місце займає саме етап тестування. Без належних заходів забезпечення якості, наразі процес розробки неможливий. Але чим складніша система, тім більше часу доводиться витрачати на тестування. Для скорочення часу та підвищення надійності якісних результатів, команда розробників має імплементувати систему автоматизованого тестування.

Підходи до тестування продуктивності програмного забезпечення — є надзвичайно важливою проблемою для багатьох великих промислових проектів. Найчастіше основними проблемами, про які повідомляють проекти після виходу на поле, є не збої системи або неправильні реакції системи, а швидше погіршення продуктивності системи або проблеми з необхідною пропускною спроможністю системи. Не рідкість дізнатися, що, хоча система програмного забезпечення пройшла обширну перевірку функціональності, вона ніколи не була перевірена, щоб оцінити очікувані показники роботи її продуктивності. Питання, які мають бути вирішені під час тестування продуктивності, різними способами відрізняються від питань, які необхідно вирішити під час типового тестування функціональності. У цій роботі було розглянуто проблеми тестування продуктивності та описані підходи високого рівня до класів програмних систем.

1 АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ

1.1 Огляд задачі тестування

Тестування програмного забезпечення – це технічне завдання, так, але воно також включає деякі важливі міркування економіки та психології людини.

В ідеальному світі ми хотіли б перевірити всі можливі перестановки програми. Однак у більшості випадків це просто неможливо. Навіть на перший погляд проста програма може мати сотні чи тисячі можливих комбінацій вводу та виводу. Створювати тестові приклади для всіх цих можливостей недоцільно. Повне тестування складної програми зайняло б занадто багато часу і вимагало б занадто багато людських ресурсів, щоб бути економічно можливим.

Крім того, тестувальник програмного забезпечення потребує належного ставлення (можливо, "бачення" - це краще слово), щоб успішно протестувати програмне забезпечення. У деяких випадках ставлення тестера може бути важливішим, ніж сам процес. Тому ми розпочнемо наше обговорення тестування програмного забезпечення з цих питань, перш ніж розібратися з більш технічним характером теми[5].

1.2 Забезпечення якості та тестування

Хоча люди часто використовують вираз забезпечення якості для позначення тестування, забезпечення якості та тестування не є однаковим, але вони пов'язані між собою. Більш широка концепція управління якістю пов'язує їх разом.

Управління якістю включає всі види діяльності, які спрямовують та контролюють організацію щодо якості. Серед інших видів діяльності управління якістю включає як забезпечення якості, так і контроль якості. Забезпечення якості, як правило, орієнтоване на дотримання належних процесів, щоб забезпечити впевненість у

тому, що будуть досягнуті відповідні рівні якості. Якщо процеси здійснюються належним чином, продукти роботи, створені цими процесами, як правило, мають більш високу якість, що сприяє запобіганню дефектів. Крім того, використання ефективного аналізу причин для виявлення та усунення причин дефектів, а також правильне застосування висновків ретроспективних нарад для покращення процесів є важливими для ефективного забезпечення якості.

Контроль якості включає різні заходи, включаючи тестові заходи, які підтримують досягнення відповідних рівнів якості. Тестова діяльність є частиною загального процесу розробки або обслуговування програмного забезпечення. Оскільки забезпечення якості стосується належного виконання всього процесу, забезпечення якості підтримує належне тестування. Тестування сприяє досягненню якості різними способами[6].

1.3 Тестові рівні

Тестові рівні - це групи тестових заходів, які організовуються та керуються разом. Кожен рівень тесту є примірником тестового процесу, що складається з заходів, що виконуються стосовно програмного забезпечення на заданому рівні розвитку, від окремих підрозділів або компонентів до повних систем або, де застосовано, системи сістем. Рівні тестування пов'язані з іншими діями в межах життєвого циклу розробки програмного забезпечення. Найрозповсюджені тестові рівні, є:

– тестування компонентів;

– інтеграційне тестування;

– тестування системи;

– тест на прийняття;

– тестові рівні характеризуються такими ознаками;

– конкретні цілі;

– тестова основа, посилається на отримання тестових випадків;

– об'єкт тестування (тобто те, що тестується);

– типові дефекти та збої;

– конкретні підходи та обов'язки.

Для кожного рівня тесту необхідне відповідне тестове середовище. Наприклад, для тестування на прийняття, наприклад, виробниче тестове середовище є ідеальним, тоді як при тестуванні компонентів розробники зазвичай використовують власне середовище розробки. На рисунку 1.1 наведені види тестів які зазвичай автоматизуються.

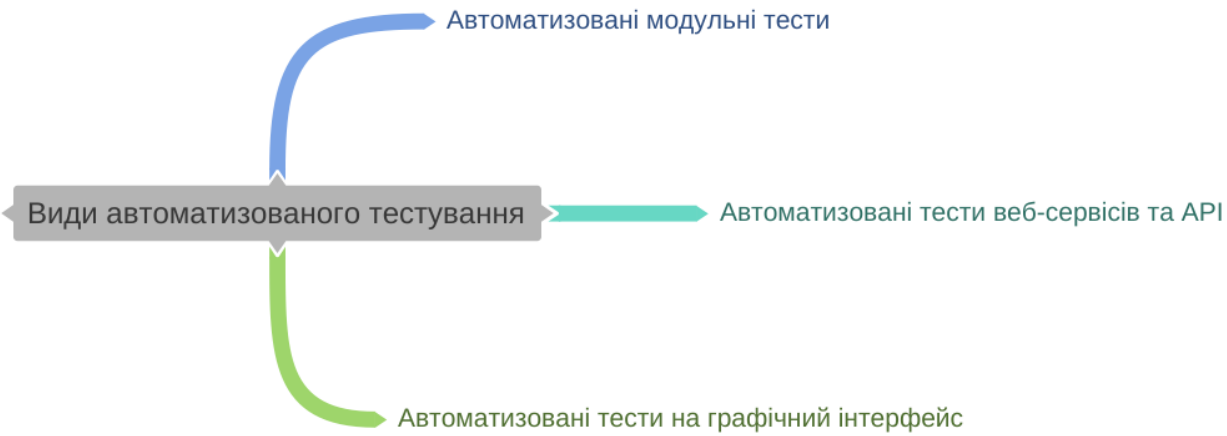

Рисунок 1.1 – Види автоматизованого тестування

1.4 Види тестування продуктивності

Можна визначити різні типи тестування працездатності. Кожен із них може бути застосований до даного проекту, залежно від цілей тесту.

Тестування продуктивності

Тестування продуктивності – це парасольковий термін, що включає будь-який вид тестування, орієнтований на продуктивність (чуйність) системи або компонента при різних обсягах навантаження.

Тестування навантаження

Тестування навантаження фокусується на здатності системи обробляти все більші рівні очікуваних реалістичних навантажень, що виникають в результаті запитів на транзакції, що генеруються контрольованою кількістю одночасно працюючих користувачів або процесів.

Тест на стрес

Стрес-тестування фокусується на здатності системи або компонента керувати піковими навантаженнями, які знаходяться на або за межами його передбачуваного або заданого навантаження. Стресове тестування також використовується для оцінки здатності системи обробляти зменшені ресурси, такі як доступна обчислювальна потужність, наявна пропускна здатність та пам'ять.

Тестування на масштабованість

Тестування на масштабованість фокусується на здатності системи відповідати майбутнім вимогам ефективності, які можуть перевищувати необхідні в даний час. Мета цих тестів - визначити здатність системи до зростання (наприклад, з більшою кількістю користувачів, більшим обсягом даних, що зберігаються), не порушуючи визначених в даний час вимог щодо продуктивності або невдалий. Після того, як будуть відомі межі масштабованості, порогові значення можна встановити та контролювати у виробництві, щоб забезпечити попередження про проблеми, які можуть виникнути. Крім того, виробниче середовище може бути відрегульовано відповідною кількістю обладнання.

Тест на спайк

Тест на спайк фокусується на здатності системи правильно реагувати на раптові сплески пікових навантажень і повертатися після цього до стійкого стану.

Тестування на витривалість

Тестування на витривалість фокусується на стабільності системи протягом певного періоду часу, специфічного для операційного контексту системи. Цей тип

тестування підтверджує відсутність проблем з ємністю ресурсів (наприклад, витоку пам'яті, підключення до бази даних, пулів потоків), які можуть бути
врешті-решт погіршує продуктивність та / або спричинить збої в точках зламу.

Тестування на сумісність

Тестування паралельності фокусується на впливі ситуацій, коли конкретні дії відбуваються одночасно (наприклад, коли велика кількість користувачів входить одночасно). Проблеми з одночасною валютою, як відомо, важко знайти та відтворити, особливо коли проблема виникає в умовах, коли тестування має мало або взагалі не контролює, наприклад, виробництво.

Тестування ємності

Тестування потенціалу визначає, скільки користувачів та / або транзакцій дана система буде підтримувати та як і раніше відповідати заявленим цілям ефективності. Ці цілі можуть бути зазначені також щодо обсягів даних, що виникають в результаті транзакцій[7].

1.5 Тестування продуктивності

Коли ми говоримо про тестування продуктивності програмного забезпечення, мається на увазі всі дії, що беруть участь в оцінці того, як можна очікувати виконання системи на місцях. Це оцінюється з точки зору користувача і, як правило, оцінюється з точки зору пропускної здатності, часу реакції на стимул або деякої комбінації обох. Крім того, тестування працездатності може використовуватися для оцінки рівня доступності системи. Наприклад, для багатьох телекомунікаційних та медичних застосунків важливо мати завжди доступну систему. У цьому рамках існує ряд різних цілей, які можна було б мати при розгляді тестування ефективності. До них належать:

– розробка алгоритмів вибору тестових випадків або генерації, спеціально призначених для перевірки на критерії продуктивності, а не на критерії функціональної коректності;

– визначення метрик для оцінки всебічності алгоритму вибору тестових випадків ефективності для даної програми;

– визначення показників для порівняння ефективності різних стратегій тестування ефективності для даної програми;

– визначення відносин для порівняння відносної ефективності різних стратегій тестування ефективності в цілому. Це вимагає, щоб ми могли якось конкретно сказати, що означає, щоб ця стратегія тестування ефективності була кращою, ніж ця;

– порівняння різних апаратних платформ або архітектури для даного додатка.

Існує також ряд речей, які необхідно виміряти, коли оцінюється продуктивність програмної системи. Серед них є: використання ресурсів, пропускна здатність, час реакції на стимулювання та тривалість черги, що деталізують середню або максимальну кількість завдань, які очікують на обслуговування вибраних ресурсів. Типові ресурси, які потрібно враховувати, включають вимоги пропускної здатності мережі, цикли процесора, дисковий простір, операції доступу до диска та використання пам'яті [1]. Інші ресурси, які можуть бути важливими для конкретних проектів, включають використання комутаторів (для телекомунікаційних проектів) або швидкість доступу до бази даних. Зауважте, що іноді неможливо задовольнити всі запити на ресурси одночасно. Якщо загальні «вимоги» до місця на диску на різних процесах перевищують доступний простір, цю проблему слід визначити якомога раніше на етапі вимог.

У деяких випадках систему доведеться реструктурувати. Це може бути дуже дорогим процесом з важкими наслідками. В інших випадках процесам просто доведеться скоротити свої потреби в ресурсах, можливо, розробивши кращі алгоритми або зменшивши функціональність.

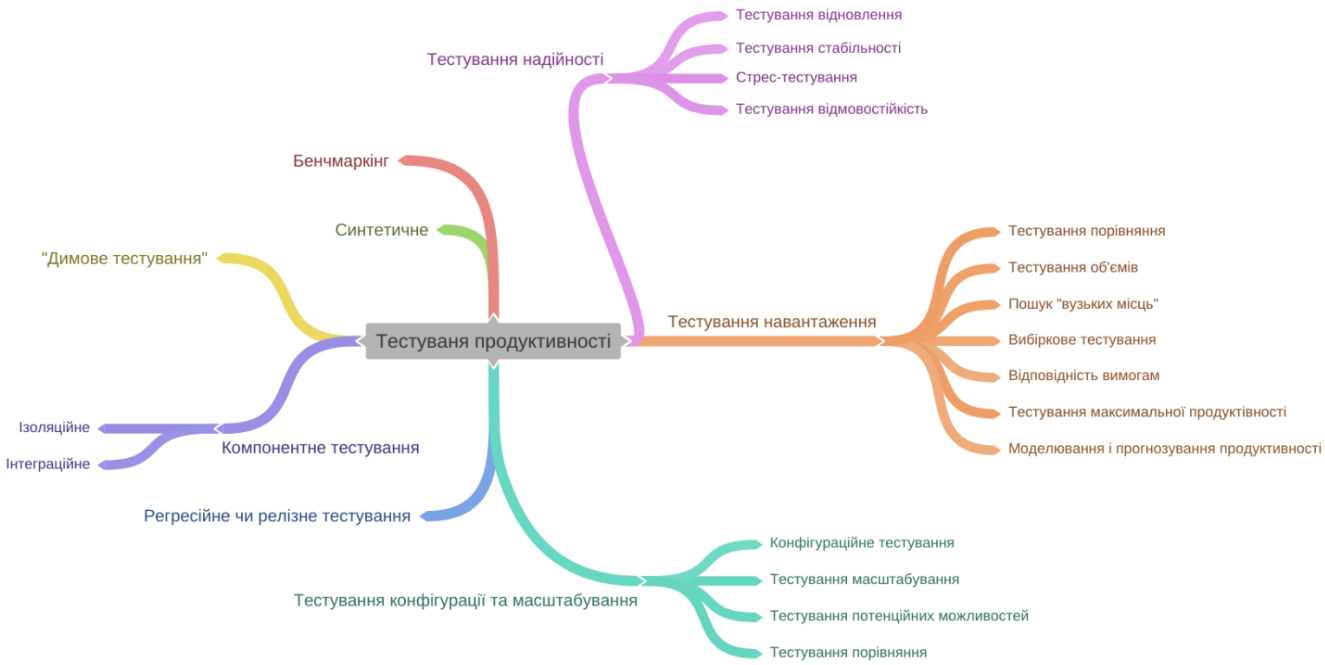

Рисунок 1.1 – Види тестування продуктивності

Іншою альтернативою може бути замовлення додаткового обладнання для задоволення потреб. У будь-якому випадку, чим раніше буде усвідомлення потенційних проблем з роботою, тим більше шансів на те, що можна розробити прийнятне рішення, і тим економніше зробити будь-яку необхідну роботу з реконструкції. Хоча тестування працездатності може бути дорогим та трудомістким, ми вважаємо, що все-таки це може бути рентабельним з причин, зазначених вище.

1.6 Інструменти тестування

Тестування програмного забезпечення є важливим для визначення якості програмного забезпечення. Основна мета тестування – перевірка, виявлення помилок та перевірка з метою пошуку проблем та їх виправлення для підвищення якості програмних продуктів. Інструменти тестування автоматизують процес

тестування та орієнтовані на конкретний тестовий домен [1]. Інструменти тестування використовуються для полегшення тестування та моделювання тестового середовища для системи, що тестується. Вони автоматизують процес тестування та спрощують його. Ручне тестування коштує дорожче, вимагає занадто великих зусиль і вимагає великих витрат часу. Хоча автоматичне тестування за допомогою інструменту зменшує необхідні витрати, час та зусилля. Існують різні типи інструментів тестування, такі як функціональні інструменти тестування, інструменти тестування в чорних ящиках та інструменти тестування білої коробки, засоби відстеження помилок, інструменти тестування продуктивності та багато інших. Інструменти для перевірки працездатності використовуються для різних типів тестування працездатності, таких як навантажувальний тест, стрес-тест, об'ємний тест і міцність. Інструмент тестування дає можливість тестувальникам створювати, керувати та виконувати тести у певному середовищі, що підтримуються для конкретного тестування для певної програми [2].

1.7 Постановка задачі

Метою роботи є дослідження основних методів тестування веб-сервісів. Результатом дослідження повинна стати розробка тестового метода на основі проведених досліджень.

Дослідження різних методів та підходів автоматизованого тестування виявляє, що у різних видах тестування тестові сценарії дублюються. Такі методи призводять до негнучкого та складного у підтримці тестового фреймворку.

У цій роботі пропонується об’єднати функціональні та сценарії навантаження з використанням сучасних підходів до розробки та розгортання веб-сервісів.

Основні вимоги, яким повинна відповідати гнучка та адаптивна методологія:

–тестовий фреймворк повинен бути незалежним від операційної системи (Linux, Windows, MacOS);

–система повинна давати чіткі і зрозумілі звіти;

–швидка інтеграція з CI/CD процесом;

–реалізація складних та довгих тестових сценаріїв з перевірками як функціональними так і навантаження.

У цій дипломній роботі представлена методологія, яка визначає процеси для ефективного проектування та впровадження тестів на ефективність для систем багатокористування. У ній представлений набір методів створення робочих навантажень, показників продуктивності та процедур тестування ефективності. Крім того, в ній обговорюється, як ці технічні випробування можуть бути технічно реалізовані. Побудова ефективних тестів на ефективність з точки зору споживання ресурсів вимагає розробки ефективних методів та алгоритмів на різних рівнях при розробці структури тесту на ефективність. Ефективна конструкція тесту на ефективність починається з вибору найбільш підходящої концепції навантаження. Існує багато можливостей для архітектури робочого навантаження, але проблема полягає в пошуку тієї, яка краще підходить для певної конфігурації апаратних та операційних систем. Крім того, модель виконання робочого навантаження є фундаментальною завдяки двом основним аспектам. По-перше, абстрактні елементи робочого навантаження, наприклад, стан машини, паралельні процеси та дані користувачів, відображаються в основні елементи операційної системи, такі як потоки, пам'ять, одиниці мережевого протоколу. По-друге, деталі робочого навантаження повинні бути виконані таким чином, щоб можливі розділення та розподіл по декількох тестових вузлах. Обидва аспекти повинні враховуватися при розробці тесту на ефективність.

2 ОПИС ДОСЛІДЖЕНЬ ТА ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

2.1 Інструменти

Проблему, яку ця роботи вирішує, можна викласти так:

Як можна ефективно реалізувати тести працездатності для багатокористувацьких систем?

Перш ніж глибше зрозуміти елементи методології тестування ефективності, слід уважніше вивчити наступні питання:

– які експлуатаційні характеристики тестуваної системи? Метою тестування продуктивності є оцінка продуктивності системи. Це досягається шляхом визначення важливих характеристик системи, пов'язаних з продуктивністю, та шляхом їх оцінки. Деякі приклади загальних характеристик продуктивності: об'єм, пропускна здатність та затримка. Характеристики ефективності оцінюються шляхом збору даних у вигляді показників ефективності, таких як: максимальна кількість користувачів, максимальна інтенсивність навантаження або затримка транзакцій;

– які параметри контролюють тест на працездатність? Для управління тестом на працездатність слід визначити ряд параметрів тесту на працездатність. Вони, як правило, стосуються аспектів завантаженості, таких як кількість користувачів, кількість дзвінків в секунду, тривалість тестування, а також математичні властивості трафіку, такі як ймовірність того, що сценарій відбудеться під час тестового пробігу;

– як поведінка користувача стосується навантаження? Робоче навантаження створюється як композиція поведінки величезної кількості користувачів. Користувач - це примірник поведінкової моделі користувача, який визначає кількість і швидкість, з якою окремий користувач робить спроби сценарію. Кожен користувач може бути обраний під час виконання тесту та призначений для певного сценарію.

– як можна ефективно виконувати навантаження? Навантаження є описом взаємодії між тестовою системою (TS) та системою, що перевіряється (SUT). Ці описи часто неформальні і зображують потоки повідомлень та процедуру тестування продуктивності, наприклад, як збільшити навантаження, скільки користувачів створити. Робочі навантаження реалізовані мовою програмування, яка пов'язує абстрактні поняття, такі як користувач, транзакція тощо, на конкретні елементи платформи: потоки, буфери даних або мережеві канали. У цьому відношенні можна виділити кілька моделей реалізації [8] для відображення абстрактних елементів навантаження на конкретні елементи платформи.

З метою реалізації запропонованої методології тестування продуктивності дисертації розглядаються наступні аспекти:

Інформаційна модель тесту на ефективність. Тест ефективності будується відповідно до інформаційної моделі тесту на ефективність. Інформаційна модель визначає обмежену кількість понять, які по суті характеризують перевірку ефективності. Прикладами таких понять є: тестовий сценарій, показник продуктивності, параметр тестування продуктивності або процедура тестування продуктивності. Ці елементи є примірниками для кожного тесту на працездатність. Інформаційна модель базується на концепції, що будь-яке навантаження, представлене SUT, починається з поведінки окремого користувача. Коли користувач взаємодіє з SUT, він / вона робить це з певною метою, наприклад, здійснити дзвінок. SUT може запропонувати різні способи досягнення цієї мети. Однак високочастотні дії відносно обмежені за кількістю і можуть бути захоплені керованим набором сценаріїв. Окремі користувачі можуть відрізнятися відносною швидкістю, з якою виконуються дії, але цю поведінку можна описати імовірнісною моделлю.

Виконання тесту на продуктивність. Сьогоднішні технології забезпечують потужні сценарії, які дозволяють писати тести на мовах високого рівня. Однак більший накладний та ресурсний попит на тестових серверах приходить разом із «простотою у використанні». Модель виконання тесту розглядає всі аспекти відображення абстрактних елементів, таких як тестовий випадок, повідомлення,

канал зв’язку з елементами операційної системи (ОС), такими як процес, пам'ять, мережа. Ідентифіковано та обговорено декілька моделей проектування для паралелізації тестування для підвищення ефективності тесту.

Використовуючи Gatling. Тести на ефективність експериментально реалізуються з використанням мови Scala як специфікації тесту та мови реалізації. Gatling підходить для визначення тестів на працездатність, а його мовні елементи спрощують проектування складних навантажень. Однак конкретне виконання та розподіл виконуваних тестів не враховується у специфікації Gatling. Тому необхідний додатковий елемент специфікації тесту для опису конфігурації тесту в цільовій мережі тестових вузлів, що потенційно складається лише з одного тестового вузла.

2.1.1 Apache Jmeter

Apache Jmeter [3] розроблений Apache Software Foundation (ASF). Проект, який може бути використаний як інструмент тестування навантаження для аналізу та вимірювання ефективності різноманітних послуг з акцентом на веб-додатках. Jmeter може бути використаний як інструмент тестування одиниць для підключення баз даних JDBC, FTP, LDAP, веб-служб, JMS, HTTP, загальних TCP-з'єднань та ОС Native. Він може бути використаний і для деяких функціональних випробувань.

Він може бути використаний для імітації великого навантаження на сервер, генеруючи одночасно кілька потоків користувачів, щоб перевірити його силу або проаналізувати загальну продуктивність при різних типах навантаження. Він також підтримує перекодування сеансу браузера через проксі-сервер і відтворює його для надання різних параметрів продуктивності, таких як час відповіді, пропускна здатність, затримка, байти відповіді та час завантаження. Він також дає різні уявлення про результати як у вигляді дерева, так і таблиці або графіків. Ці види

також одночасно доступні для використання. Плани тестів можуть бути збережені у форматі XML і можуть бути повторно використані. Він також може бути використаний для деяких функціональних тестувань. Архітектура Jmeter заснована на плагіні. Інші його функції реалізовані за допомогою плагінів. Розробники за межами сайту можуть легко розширити Jmeter за допомогою спеціального плагіну[1].

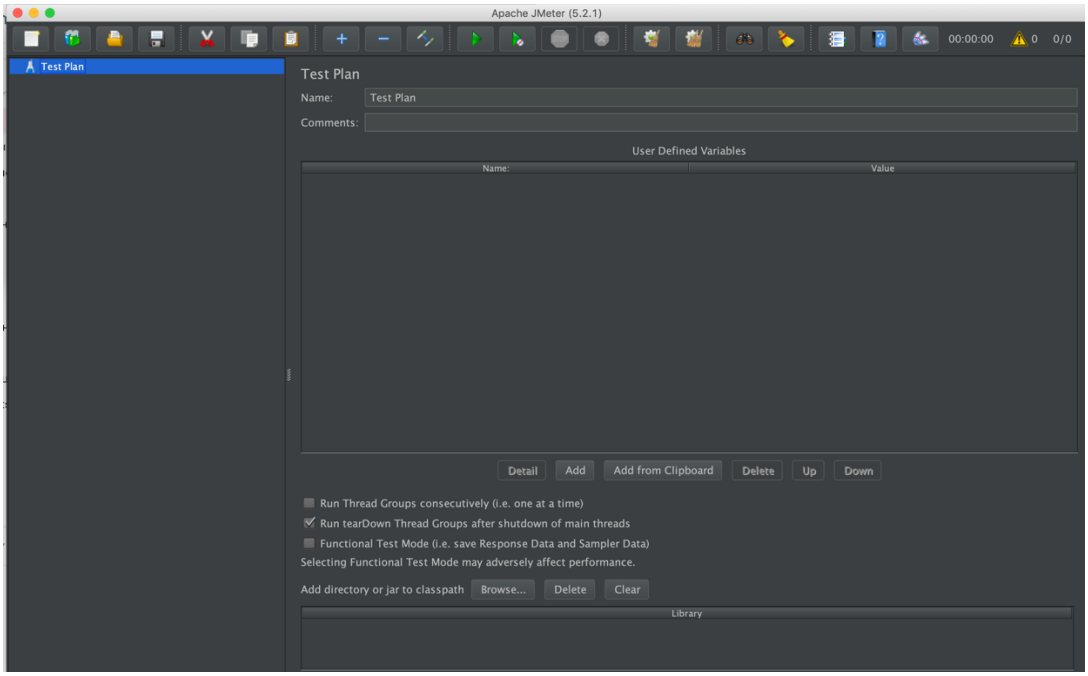

Рисунок 2.1 – Графічний інтерфейс JMeter

Основні характеристики:

Наявність інтерфейса користувача: JMeter постачається з інтерактивним графічним інтерфейсом

Незалежна платформа: JMeter написаний та розроблений за допомогою Java, тому він може працювати на будь-якому середовищі чи робочій станції, яка приймає віртуальну машину Java, наприклад - Windows, Linux, Mac тощо.

Підтримуються різні типи серверів: Web (HTTP, HTTPS, SOAP), база даних (JDBC, LDAP, JMS) та пошта (POP3).

Підтримує декілька протоколів, таких як HTTP, JDBC, LDAP, SOAP, JMS та FTP.

Моделює декількох користувачів, використовуючи віртуальних користувачів або унікальних користувачів, щоб генерувати велике навантаження на тестування веб-додатків.

Багатопотокова рамка дозволяє одночасно і одночасно вибирати різні функції багатьма або окремими групами ниток.

Віддалене розподілене тестування: JMeter використовує концепцію Master-Slave для розподіленого тестування, де майстер розподіляє тести між усіма рабами, а раби виконуватимуть сценарії проти вашого сервера.

Результати тестування можна переглянути в різних форматах, таких як графік, таблиця, дерево та звіт тощо.

2.1.2 Gatling

Gatling[4] — це інструмент тестування з відкритим кодом навантаження та працездатності на основі Scala, Akka та Netty. Програмне забезпечення розроблене для використання в якості інструменту тестування навантаження для аналізу та вимірювання продуктивності різноманітних послуг з акцентом на веб-додатках. Розроблений для полегшення безперервного тестування, він інтегрується з вашим інструментом збирання та пропонує веб-рекордер та барвисті звіти. Gatling також постачається у платній, корпоративній версії, Gatling FrontLine, яка забезпечує розширену аналітику та інтеграцію.

Основні характеристики:

– автономний протокол HTTP Proxy Recorder;

– сценарії на основі Scala;

– виразний роз'яснювач DSL для розробки тесту;

– асинхронний неблокуючий двигун для максимальної продуктивності;

– відмінна підтримка протоколів HTTP (S), а також може використовуватися для тестування навантаження JDBC та JMS;

– валідації та перевірки;

– вичерпний HTML-звіт.

Рисунок 2.2 – Графічний інтерфейс рекордера Gatling

У таблиці 2.1 наведено найвище порівняння між JMeter і Gatling:

Таблиця 2.1 – Порівняння JMeter та Gatling

	JMeter	Gatling
Визначення	Це інструмент тестування навантаження та працездатності, що надається Apache, який є відкритим кодом та безкоштовний у використанні	Інструмент тестування навантаження та працездатності, який і має безліч функцій

Продовження таблиці 2.1

	JMeter	Gatling
Ліцензія	Ліцензувалася під ліцензією Apache License 2.0	Ліцензувався під ліцензією Apache License 2.0
Інтеграція	Інтеграція з ним має різні плагіни для інтеграції з кількома сторонніми програмами	Він має різні функції для підтримки функцій інтеграції за допомогою інструменту панелі інструментів
Розроблено	Це було розроблено за допомогою мови програмування Java	Він був розроблений за допомогою мови програмування Scala, яка має функціональні аспекти програмування. Також є можливість використовувати Java
Загальність	Він не має гарних інструментальних панелей, але має кілька плагінів	У ньому є приладова панель з моніторингом у режимі реального часу
Технічне обслуговування	Його підтримує Apache Software Foundation	Його підтримує компанія Gatling Enterprise
Вбудовані засоби	У ньому є графічний інтерфейс, а не лише командний рядок	Він заснований на бігунці командного рядка
Простота використання	Трохи складно використовувати. Має надлишковий застарілий інтерфейс	Він простіший у використанні і може легко інтегруватися з будь-яким додатком
Гнучкість	Він підтримує лише протокол HTTP. Він підтримує протокол	HTTP, а також кілька інших протоколів

Кінець таблиці 2.1

JMeter і Gatling обидва можуть бути використані для тестування у випадку навантаження та параметрів продуктивності. Щодо надання результатів тестування, два інструменти відрізняються. У випадку з JMeter, використання процесора більше, тоді як Gatling використовує менше процесора, більше використання мережі та менше місця на диску. Зрештою, JMeter використовує більше ресурсів системи порівняно з Gatling.

У порівнянні з JMeter, Gatling має у своєму інструменті різні інформаційні панелі з Runner Command Line для відображення результатів продуктивності в одному екземплярі. JMeter має функцію графічного інтерфейсу, але вона не користується великою популярністю. У Gatling є кілька можливостей інтеграції з інструментами безперервної інтеграції, такими як Jenkins, Hudson тощо. Gatling має інструменти моніторингу та засоби інтеграції в реальному часі для моніторингу веб-додатків або служб. Нарешті, підсумовуючи, Gatling має більшу кількість функцій та простіший у використанні порівняно з JMeter.

Керуючись тим, що вимоги до системи стоять такі як: реалізація багатокрокових тестових сценаріїв та проста інтеграція з CI/CD процесом, то було обрано в якості основного інструменту – Gatling. 2.1.1 Apache Jmeter популярний інструмент, але він підходить тільки для тестових випадків спрямованих на тестування продуктивності і реалізація складних багатокрокових сценаріїв складна чи неможліва.

В рамках цієї роботи було обрано Gatling, тому що є підтримка захищеного протоколу HTTP (тобто HTTPs). Та є можливість імпортувати та виконувати тести у контейнерах.

3 ПРОЕКТУВАННЯ ТЕСТОВОЇСИСТЕМИ СИСТЕМИ

3.1 Визначення еталону

Мета проекту, з якого походить дослідження, - створити орієнтир для IMS, який передбачає реалізацію репрезентативного робочого навантаження та тестової процедури, здатної дати порівнянні результати. Тому деякі визначення, що стосуються бенчмаркінгу, подаються спочатку.

Еталон [10] – це програма або програма, яка використовується для оцінки продуктивності цільової системи (апаратного чи програмного забезпечення). Це реалізується шляхом імітації певного типу робочого навантаження на компонент або систему. У тестових сценаріях тестові сценарії тестують послуги SUT, продуктивність утиліти, можна експериментувати з різними методами завантаження даних, характеристиками швидкості транзакцій, оскільки додається більше користувачів, і навіть тестувати регресію при модернізації деяких частин системи.

Еталон – тип тесту на продуктивність, розроблений таким чином, що його можна використовувати як метод порівняння продуктивності різних підсистем у різних архітектурах. З точки зору бізнесу, орієнтир є ідеальним інструментом порівняння систем та прийняття адекватних рішень щодо придбання. Тим не менш, орієнтир виступає також як інструмент інженерії продуктивності для виявлення проблем продуктивності серед версій системи.

Тест непростий для реалізації, і часто інструменти мають справу з високими можливостями продуктивності. Перелік вимог включає:

– контрольна процедура – метод визначення навантаження та вимірювання продуктивності повинен базуватися на кількості віртуальних користувачів. Це може бути виміряно або кількістю абонентів, яку може підтримувати конкретна конфігурація, або мінімальною системною вартістю конфігурації, яка може підтримувати певну кількість користувачів.

– обмеження апаратних ресурсів – орієнтир повинен вимірювати потужність системи. Для цього потрібна тестова система для імітації поведінки великої кількості користувачів, які взаємодіють з SUT. Хоча тест визначатиме навантаження на трафік, що відповідає ти сячам модельованих користувачів, метод генерації трафіку повинен бути визначений таким чином, щоб його можна було економічно реалізувати тестовою системою.

– реалістичне навантаження – орієнтир повинен керуватися набором сценаріїв руху та профілів трафіку, тобто тест виконується для різних рівнів навантаження. Тестова система повинна генерувати реалістичний трафік і охоплювати більшість випадків використання, що цікавлять.

3.2 Експериментальні дослідження

Експерименти, представлені в цьому розділі, служать двом головним цілям. Перша мета - показати конкретне виконання еталону з повним аналізом працездатності тестованого веб-сервісу. Крім того, тест застосовується до різних конфігурацій апаратних засобів, щоб проілюструвати, як тест може бути використаний для порівняння продуктивності. Друга мета - більш детально обговорити деякі параметри тесту, які впливають на результати. Важливо зрозуміти їх і навчитися вибирати дійсні значення для них, щоб результати були також достовірними та значущими для аналізу ефективності.

3.2.1 Випробувальне середовище

Середовище яке використовується для запуску тестової системи це хмарний CI/CD инструмент, що надає контейнерні ресурси для сбірки, тестування та

розгортання програмного забезпечення. Поточна конфігурація складається з Docker контейнеру, на якому встановлена операційна система Kernel Version: 18.04.1-Ubuntu. 2 ядерний виртуальный центральний процесор, та оперативна памьять – 4096Мб.

3.2.2 Програмне забезпечення SUT

SUT складається з програмного та апаратного забезпечення. Однак оскільки обладнання, на якому працює SUT, не залежить від експерименту, воно не буде представлено окремо для кожного експерименту.

Реалізація веб-сервісу, якій використовується як SUT в рамках тематичного дослідження, є API з HTTPS протоколом. Аналіз продуктивності будується як сервер, використовуючи ту саму пам’ять, блокування, процедури реєстрації та основні структури. Його швидкість в основному обмежена лише базою даних, яка використовується.

4 ОПИС ПРОВЕДЕННЯ ТЕОРЕТИЧНІХ І ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Тест 1. Повний приклад виконання еталону

У цьому розділі пояснюється спосіб виконання тесту щодо конкретного веб-сервісу. Він також обговорює, як інтерпретувати різні статистичні дані, і, що найважливіше, як визначити максимальне навантаження, яке підтримує SUT.

Набір трафіку

Згідно з процедурою випробувань, повне тестове тестування складається з декількох етапів випробувань з метою визначення максимального витримуваного навантаження SUT. Поріг припинення тестової кампанії досягається, коли відсоток помилок перевищує поріг. Значення, вибрані для порогу відмов від створеного навантаження, становлять 0,1%. Вибрані сценарії для набору трафіку представлені в таблиці 2.2.

Таблиця 2.2 – Склад набору трафіку

Тип сценарію	Мітка	Коефіцієнт
початкова реєстрація	s1.1	1%
перереєстрація	s1.2	1%
зняття з реєстрації	s1.3	3 1%
успішний голосовий дзвінок без резервування ресурсів	s2.1	12%
успішний голосовий дзвінок із резервацією ресурсів на стороні, що надходить	s2.2	12%
успішний голосовий дзвінок із резервацією ресурсів на стороні, що припиняє	s2.3	12%
успішний голосовий дзвінок з резервацією ресурсів з обох сторін	s2.4	12%
кинутий голосовий дзвінок без резервування ресурсів	s2.5	3%

Продовження таблиці 2.2

Тип сценарію	Мітка	Коефіцієнт
кинутий голосовий дзвінок із резервацією ресурсу на стороні, що надходить	s2.6	3%
кинутий голосовий дзвінок із резервацією ресурсів на стороні, що припиняє	s2.7	3%
кинутий голосовий дзвінок із резервацією ресурсів з обох сторін	s2.8	3%
відхилений голосовий дзвінок без резервування ресурсів	s2.9	3%
відхилений голосовий дзвінок із резервацією ресурсу на стороні, що надходить	s2.10	3%
відхилений голосовий дзвінок із резервацією ресурсу на стороні, що припиняє	s2.11	3%
відхилений голосовий дзвінок із резервацією ресурсів з обох сторін	s2.12	3%
невдача голосового дзвінка без резервування ресурсів	s2.13	1%
успішне повідомлення в режимі сторінки	s3.1	19%
невдале повідомлення в режимі сторінки	s3.2	5%

Кінець таблиці 2.2

Профіль трафіку

Профіль трафіку, параметризуючи виконання тесту, наведений у таблиці 2.3. Тест починається з фази перемішування SUT протягом 720 секунд з подальшими трьома кроками навантаження при 60, 70 та 80 SAPS. Кожен крок виконується протягом 600 сек. Пробіг використовує 100000 абонентів, і з них 40% реєструються на початку і використовуються на початку тестів. Під час тестування також будуть зареєстровані інші користувачі від решти до 100000, але кількість активних користувачів залишиться майже однаковою, оскільки реєстрація та зняття з обліку мають однакову частоту в наборі трафіку. Ця кількість може дещо відрізнятися

через сценарії реєстрації / зняття з обліку, які роблять користувачів активними або неактивними.

Таблиця 2.3 – Профіль трафіку

Параметра профілю трафіку	Значення
PX_StirTime	720 сек
PX_SimultaneousScenarios	2
PX_TotalProvisionedSubscribers	100000
PX_PercentRegisteredSubscribers	40%
PX_StepNumber	2
PX_StepTransientTime	120 сек
PX_StepTime	600 сек
PX_SApSIncreaseAmount	10 SAPS
PX_SystemLoad	60 SAPS
PX_PreambleLoad	42 SAPS

Графік SAPS

На рисунку 2.1 представлений основний графік тесту на тест. На графіку зображено дві фігури. Перший, вище, представляє навантаження, прикладене до СТВ. Він перетинається декількома вертикальними лініями, які розмежовують окремі кроки пробігу. Тест починається з кроку преамбули, коли користувачі реєструються. Це легко розпізнати, оскільки навантаження застосовується з постійною швидкістю. Перша вертикальна лінія позначає початок часу перемішування, коли SUT "прогрівається" для першого кроку завантаження. Час перемішування вибирається досить тривалим, щоб дозволяти SUT прилаштовуватися до навантаження, таким чином тестова система дозволяє уникнути наслідків, викликаних безпосередньо великим навантаженням. Час навантаження перемішування насправді розбивається на три покрокові етапи, щоб останній крок мав навантаження, близьку до навантаження першого кроку. Кінець часового навантаження перемішування позначається другою вертикальною лінією.

Після закінчення часу перемішування випробувальна система переходить до передбачуваного випробування навантаження.

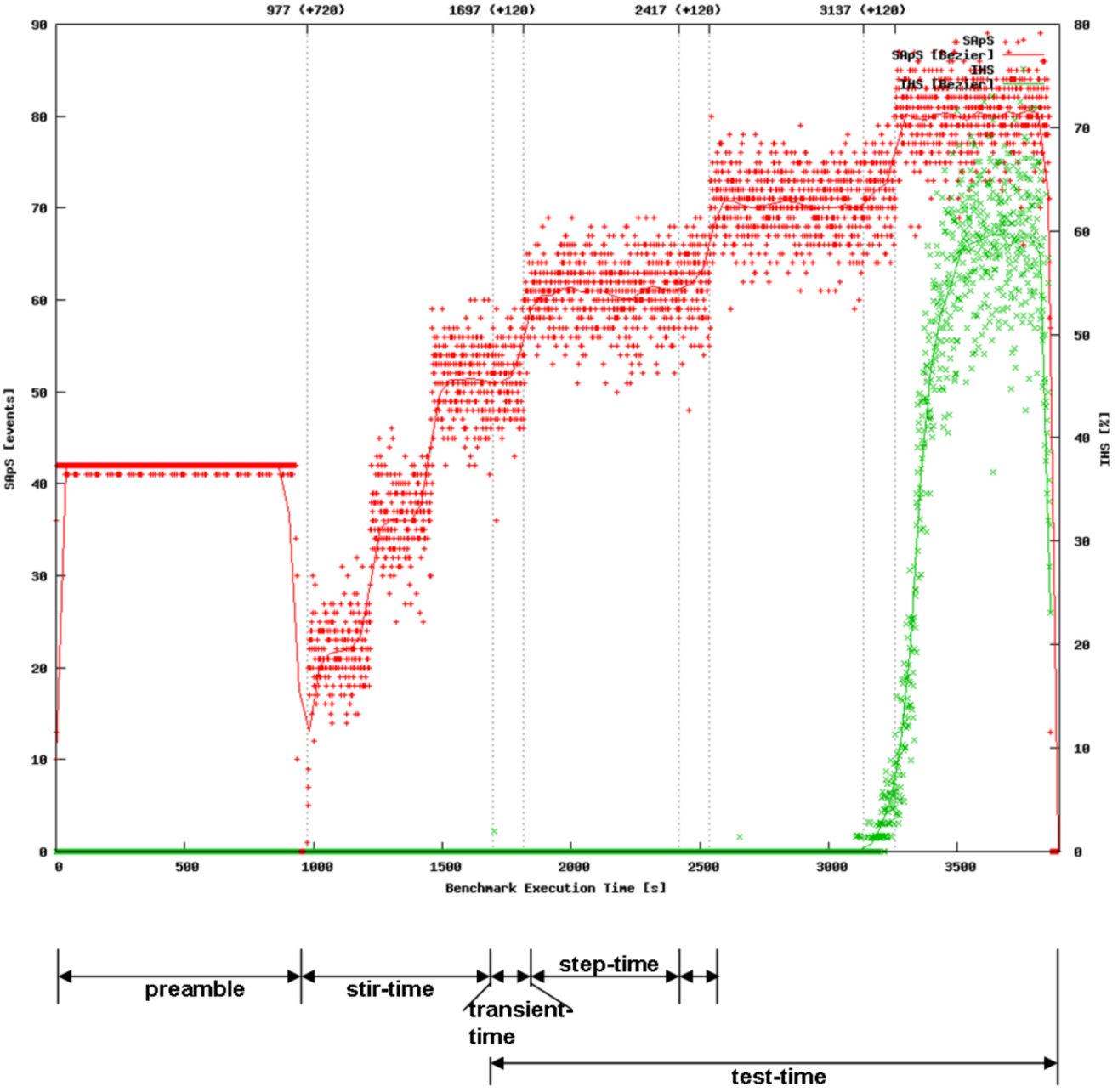

Рисунок 2.1 – Кроки для визначення навантаження, для якої SUT починає виходити з ладу

На початку кожного кроку навантаження застосовується на короткий період часу розміщення, який називається перехідним часом. Під час перехідного часу тестова система збільшує навантаження з одного кроку на інший, але не обчислює статистику. Причина додавання перехідного часу полягає в тому, щоб дозволити

тест-системі чекати, поки виклики, створені на попередньому кроці, закриються і більше не заважатимуть навантаженню, яке буде створено на наступному кроці. Після закінчення часу перемішування тест виконує три етапи (на 60, 70 і 80), а на початку кожного кроку перехідний час позначається вертикальною лінією.

Друга форма відображає коефіцієнт помилок, який обчислюється як відсоток відмов від загального створеного навантаження. Форма помилки залишається нульовою протягом тривалого періоду часу, але починає збільшуватися протягом останнього кроку. У середині останнього кроку він досягає 50-60%.

Очевидно, тестова система також виявляє, що SUT досяг межі стійкого навантаження, порівнюючи на кожному етапі середнє значення IHS з порогом. Максимальні та середні значення IHS% представлені в таблиці 2.4.

Таблиця 2.4 – Статистика неадекватної обробки сценаріїв

Step	Avg	Max
step1 (60SAPS)	0.00	0
step2 (70SAPS)	0.01	1.56
step3 (80SAPS)	49.28	73.08

IHS% перевищує поріг на останньому кроці, де середній показник становить 49,28%. Це означає, що майже половина створених дзвінків виходять з ладу. Це підводить нас до висновку про перевищення ДОК СУТ. Виходячи з процедури випробування, останнє навантаження, де середнє значення IHS нижче порогового значення, - фактичне значення DOC SUT. Це відбувається в прикладі на етапі 2, де середній показник IHS становить лише 0,01.

Графік IHS

Більш детальний вигляд типів помилок наведено на рисунку 2.2, де відображаються форми IHS% за кожний випадок використання. Цей графік показує, що найбільш часто трапляються помилки - це випадок використання сеансу налаштування / руйнування сеансу, в той час як менше помилок має тип повідомлень.

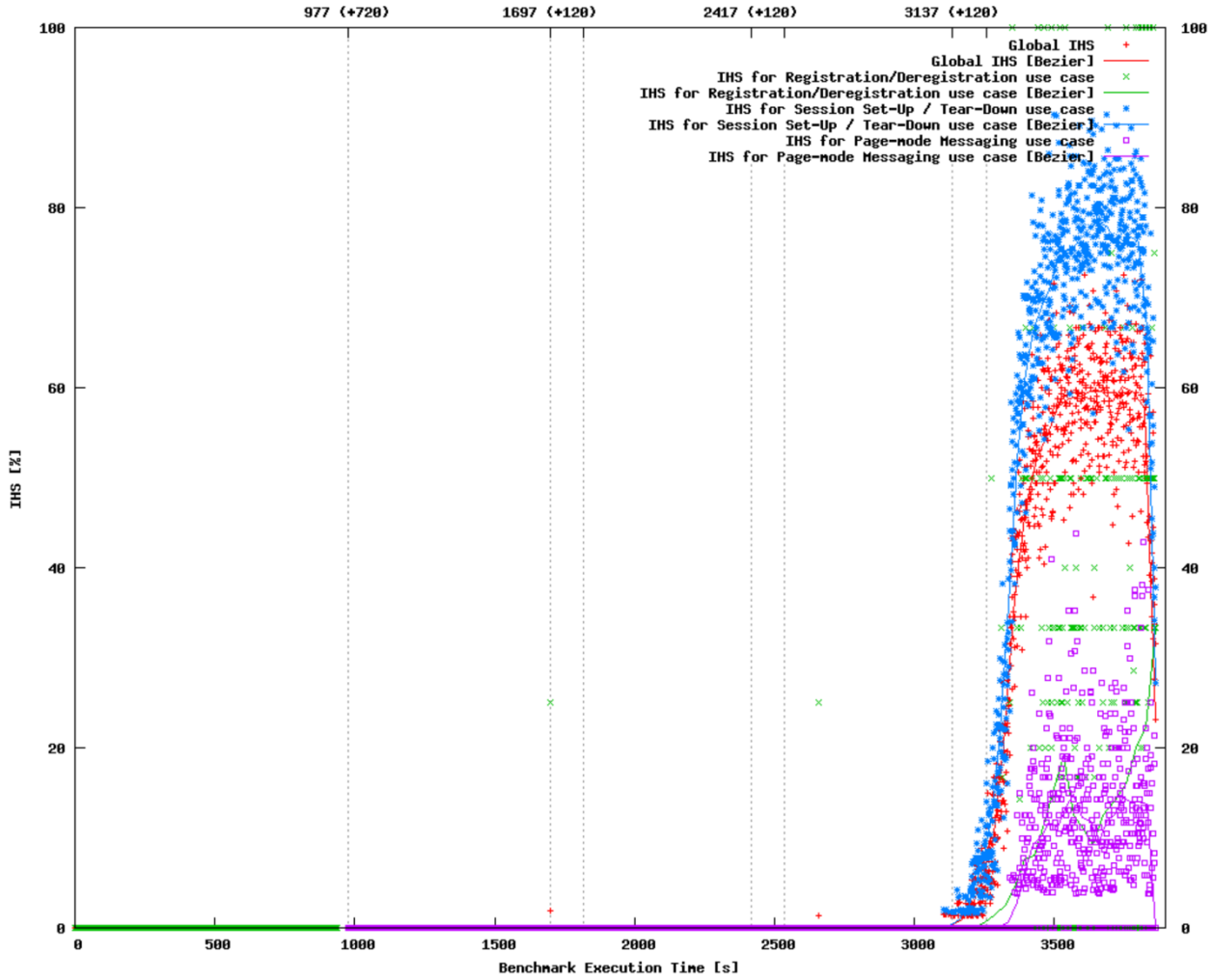

Рисунок 2.2 – Візуалізація збоїв у кожному випадку

Причини помилок у сценаріях

Причини помилок можна дослідити окремо для кожного сценарію. На рисунку 2.3 показана статистика помилок для успішного виклику зі сценарієм резервування ресурсів. Протягом перших двох кроків помилок немає. Але на останньому кроці всі показники помилок збільшуються. Найбільше невдач відбувається при надсиланні першого запиту (ЗАПИТАННЯ) до сторони, що закінчується, а це означає, що SUT не в змозі обробляти нові дзвінки після досягнення потужності перевантаження.

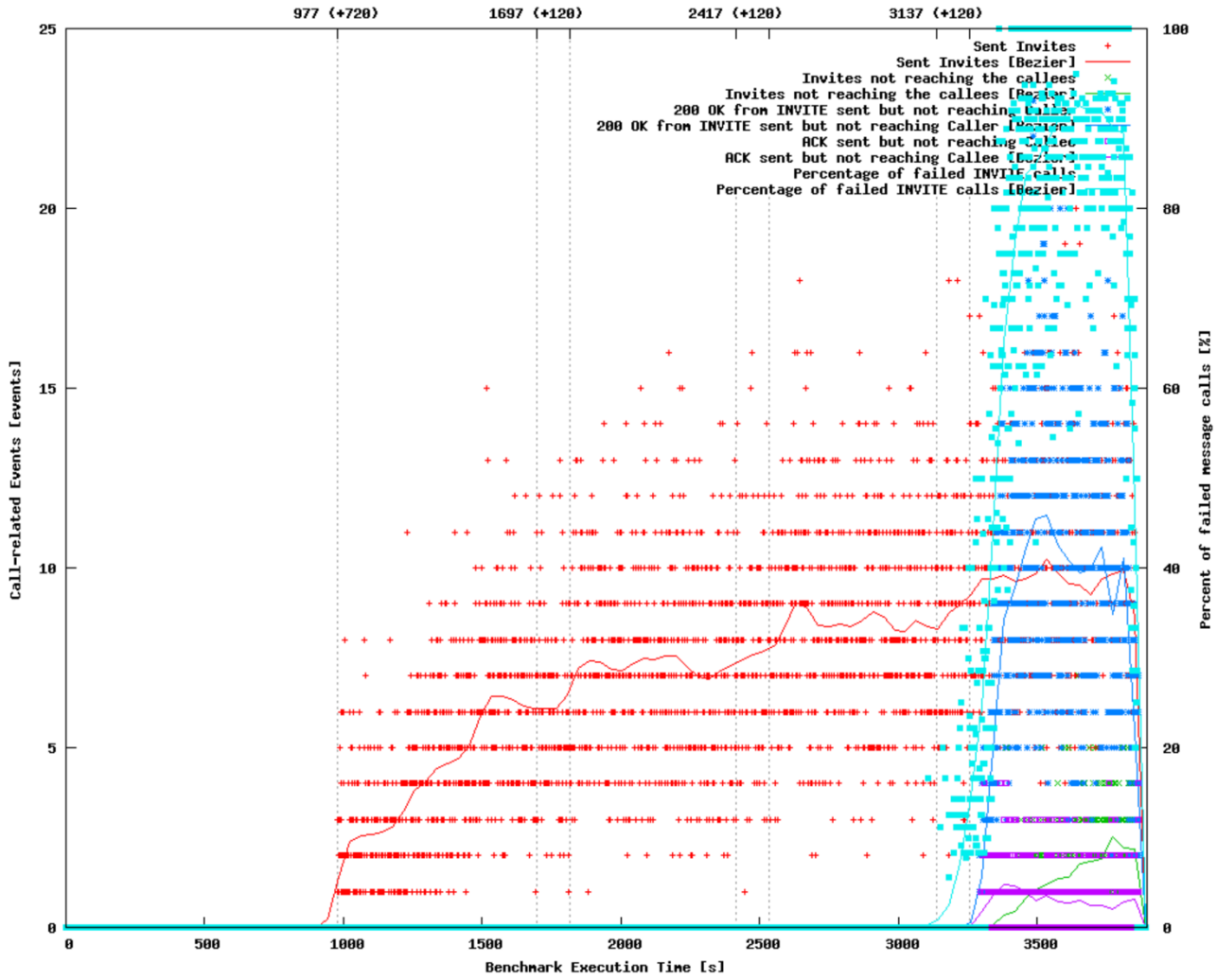

Рисунок 2.3 – Статистика помилок для успішного виклику у сценаріях з резервуванням ресурсів

Графік SIMS

На рисунку 2.4 показано SIMS (одночасні сценарії) вздовж тестового виконання. Залежно від сценарію, час дзвінка залишається відкритим, коливається від декількох секунд, наприклад, реєстрація, обмін повідомленнями, до декількох хвилин, наприклад, успішний дзвінок. Тому очікується, що значення метрики SIMS вказуватиме на велику кількість відкритих дзвінків у порівнянні з кількістю SAPS.

У нормальній поведінці цей показник повинен стабілізуватися навколо середнього значення, якщо SAPS підтримується постійним. На цій фігурі, після нормальної форми на етапі 1 і 2, значення SIMS різко зростають на останньому кроці, що фактично означає, що SUT отримує занадто багато викликів і він не може

закінчити існуючі. Форма SIMS, здається, не стабілізується, а навпаки, постійно піднімається до кінця тесту. Це ще одна чітка підказка про те, що SUT досягає свого DOC на останньому кроці.

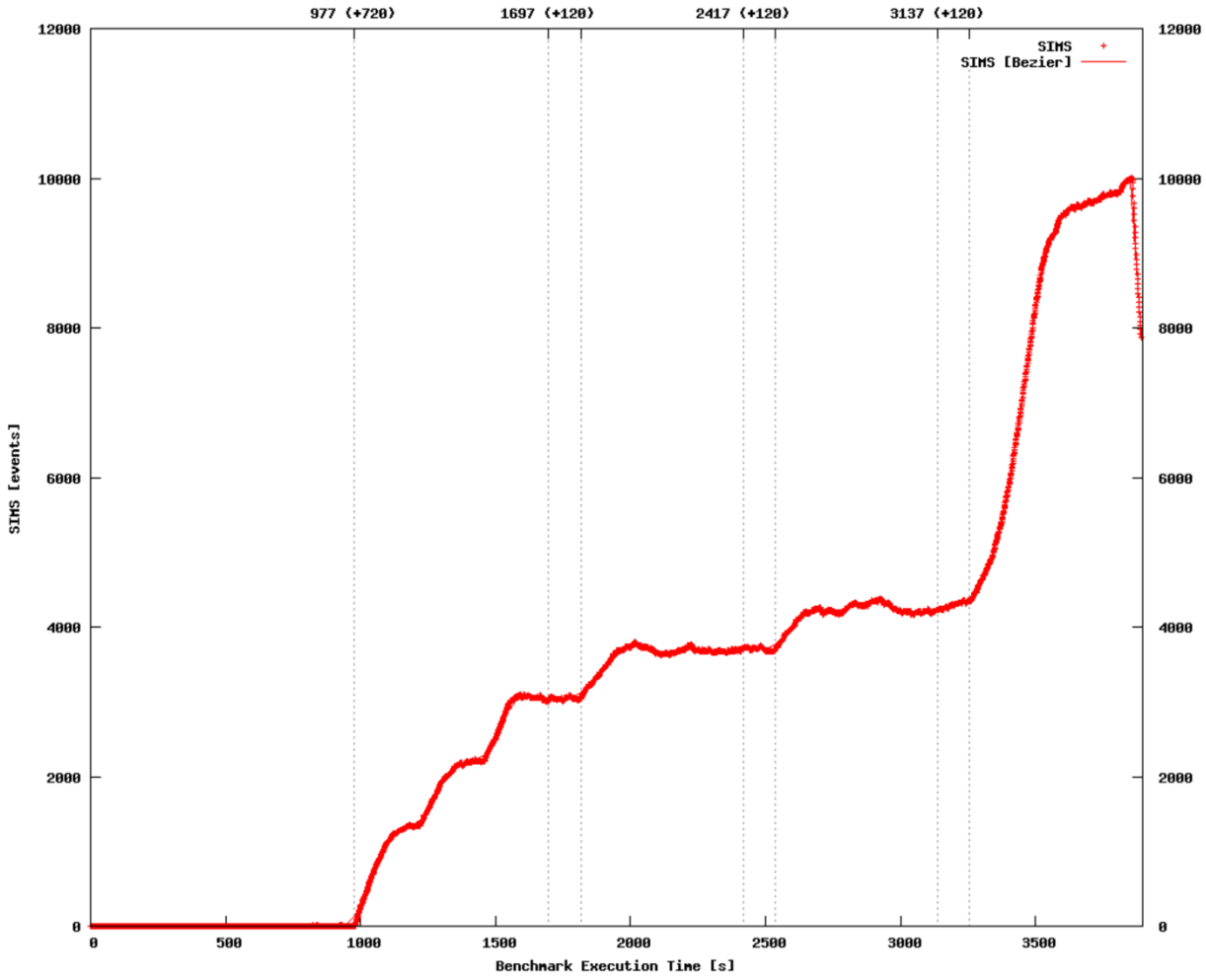

Рисунок 2.4 – Візуалізація одночасних сценаріїв

RETR графік

На наступному графіку (див. рис. 2.5) метрика RETR (повторна передача) візуалізується для кожного випадку використання. Метрика RETR показана паралельно з метрикою SAPS, щоб інтенсивність навантаження могла бути співвіднесена зі значеннями RETR. У звичайному SUT поведінка, кількість повторних передач повинна бути дуже низькою (в ідеалі не має повторних передач). Під час кроків 1 і кроку 2 RETR близький до нуля. На останньому кроці

тестова система повинна повторно передати багато повідомлень, оскільки SUT не відповідає вчасно.

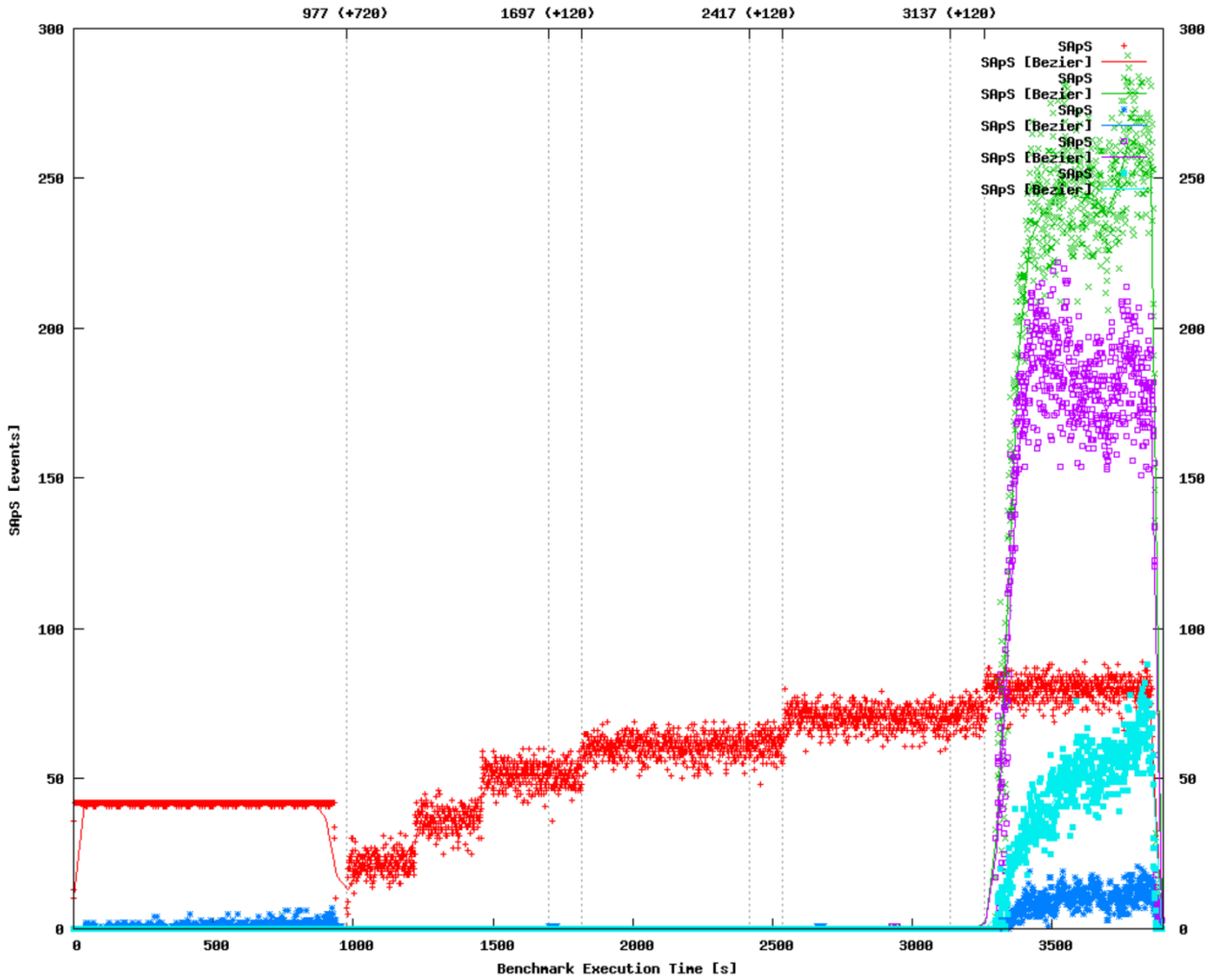

Рисунок 2.5 – Візуалізація передачі повідомлень

Графік процесора

Без сумніву, найкращий вигляд того, що відбувається навколо DOC, дають графіки моніторингу процесора. На рисунку 2.6 показано час роботи в режимі очікування, системного та користувальницького процесора. З цього графіку очевидно, що DOC досягається тоді, коли у SUT майже не вистачає ресурсів. Наприклад, форма холостого ходу зменшується з 50% під час першого кроку до 10% на останньому кроці. Дзеркально, час користування збільшився приблизно з 40% до 80%, що вказує на високий попит процесора на стороні SUT.

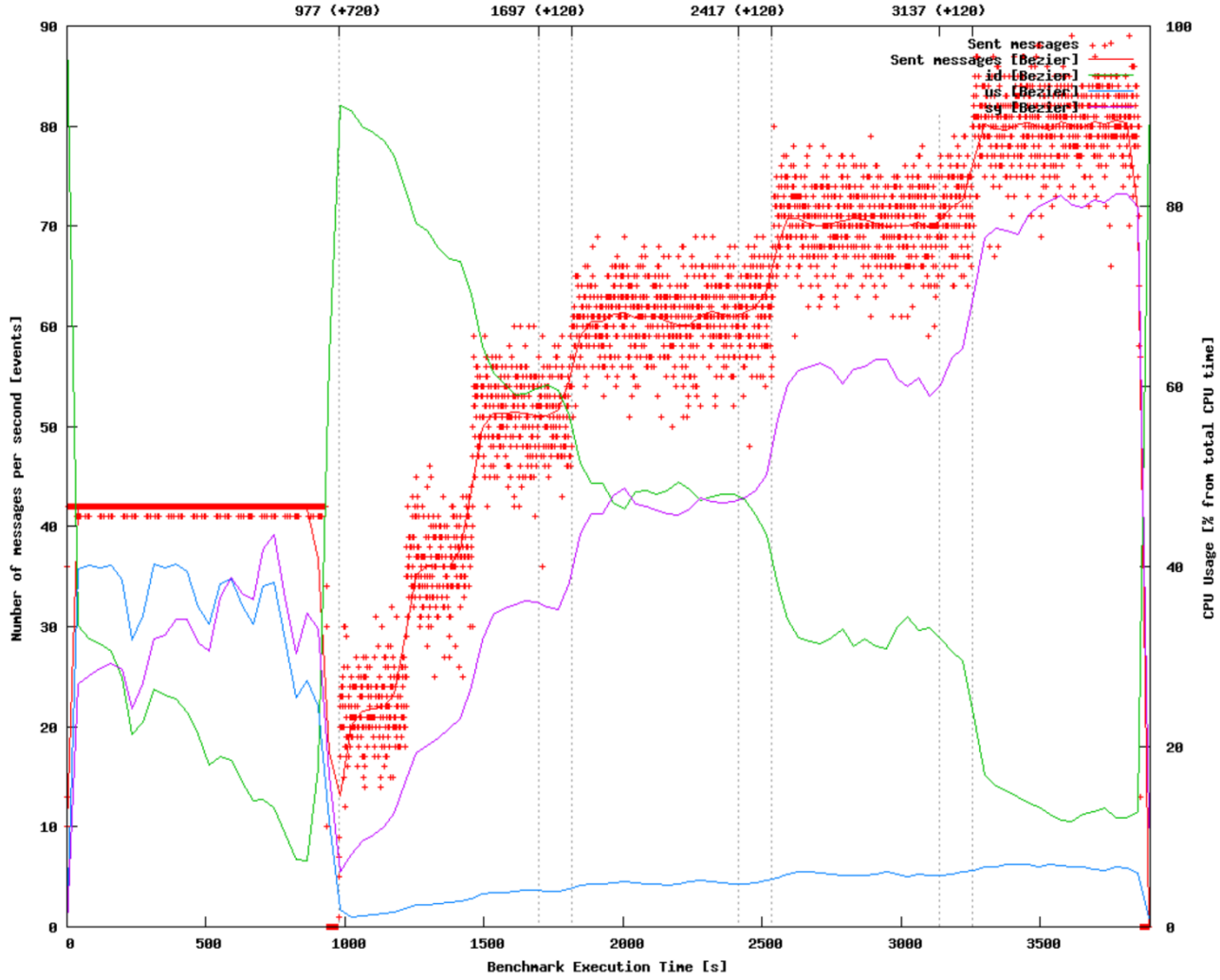

Рисунок 2.6 – Споживання процесора

Графік використання пам’яті

Ненормальну поведінку SUT навколо DOC також помічено на графіку споживання пам'яті, представленому на рисунку 2.6. Хоча під час перших двох кроків вільна пам'ять зменшується дуже мало, на останньому кроці спостерігається величезне падіння, що вказує на те, що SUT потрібно набагато більше пам’яті.

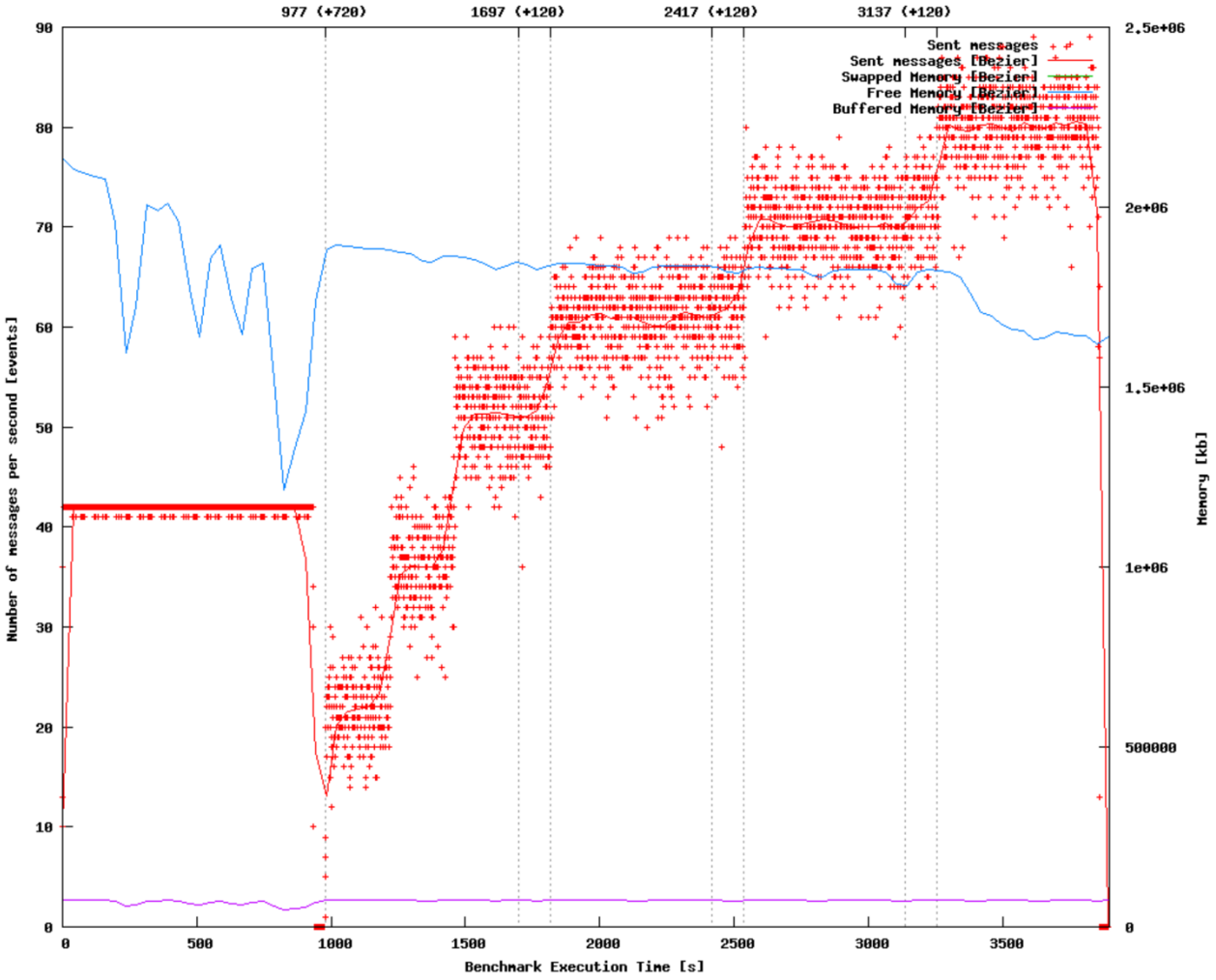

Рисунок 2.6 – Споживання пам'яті

Тест моніторингу центрального процесору

Для перевірки результатів необхідно довести, що тестова система не була перевантажена під час виконання тесту. Це можна перевірити, переглянувши споживання процесора кожного сервера тестової системи. Один з цих графіків представлений на рісунку 2.7. В ході тесту система тесту ледь використовувала ресурс процесора, майже 90% не працює, за винятком останнього кроку, коли тестовій системі довелося мати справу з повторною передачею (що вимагає більше процесора).

Рисунок 2.6 – Витрата процесора на системному сервері

Довгі тести, щоб довести, що SUT працює стабільно при завантаженні DOC.

Після того, як DOC був знайдений, важливо перевірити, чи SUT веде себе стабільно і на більш тривалих ходах. Тривалий пробіг протягом принаймні півгодини виявляє ситуації, коли SUT може утримувати навантаження DOC лише для коротких, але не для більш тривалих. В ідеалі всі етапи процедури тестування повинні бути тривалими, але щоб уникнути дуже тривалих тестових виконань, для знаходження DOC використовуються більш короткі прогони, і наприкінці декілька циклів підтвердження доводять значення DOC. На рисунку 2.7 видно, що SUT залишається стабільним також протягом 30 хвилин.

Рисунок 2.7 – Довгий тест для перевірки стійкості DOC

4.2 Тест 2. Успішні сценарії проти поєднання успішних, кинутих, відхилених, невдалих сценаріїв

Цей експеримент показує вплив змішування успішних сценаріїв із покинутими, відхиленими чи невдалими сценаріями. На рисунку 2.8 показано споживання процесора для набору трафіку, який складається лише з успішних сценаріїв. На рисунку 2.9 показано споживання процесора для набору трафіку, який складається із сценаріїв з усіх категорій. У другому запуску SUT використовує менше процесора, оскільки помилки легше обробляти, ніж успішні дзвінки. Це

відбувається тому, що успішні дзвінки включають також період часу розмови, який вимагає від SUT зберігати стан виклику в пам’яті протягом цього періоду часу. Це контрастує із покинутими або відхиленими сценаріями, які не повинні залишатися активними, поки успішні. Цей результат може допомогти постачальникам послуг правильно оцінити потужність своїх систем. Однак ці оцінки покладаються на статистику, яка вказує на частку відмовлених або відхилених дзвінків із загальної кількості дзвінків.

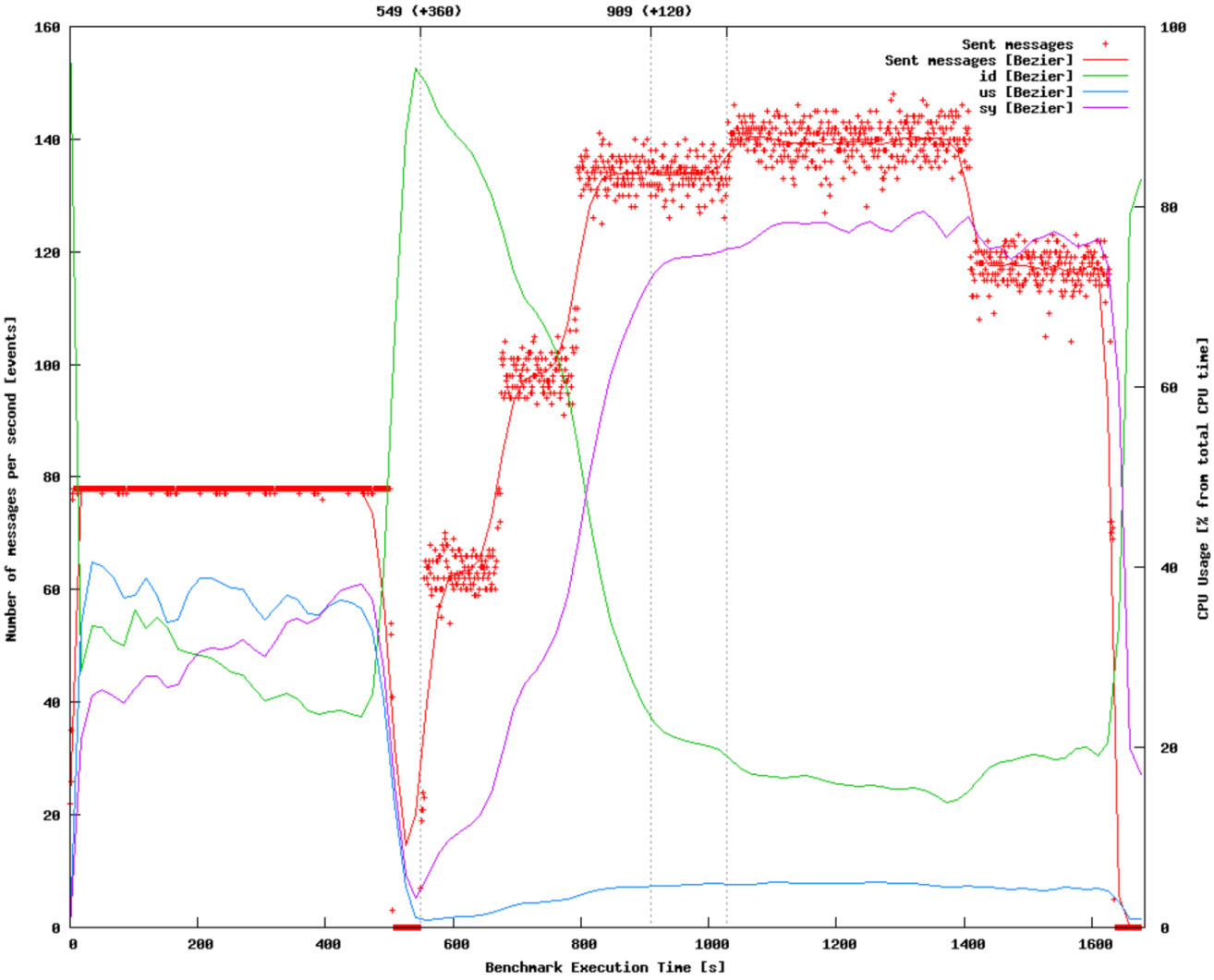

Рисунок 2.8 – Споживання процесора, використовуючи лише успішні сценарії

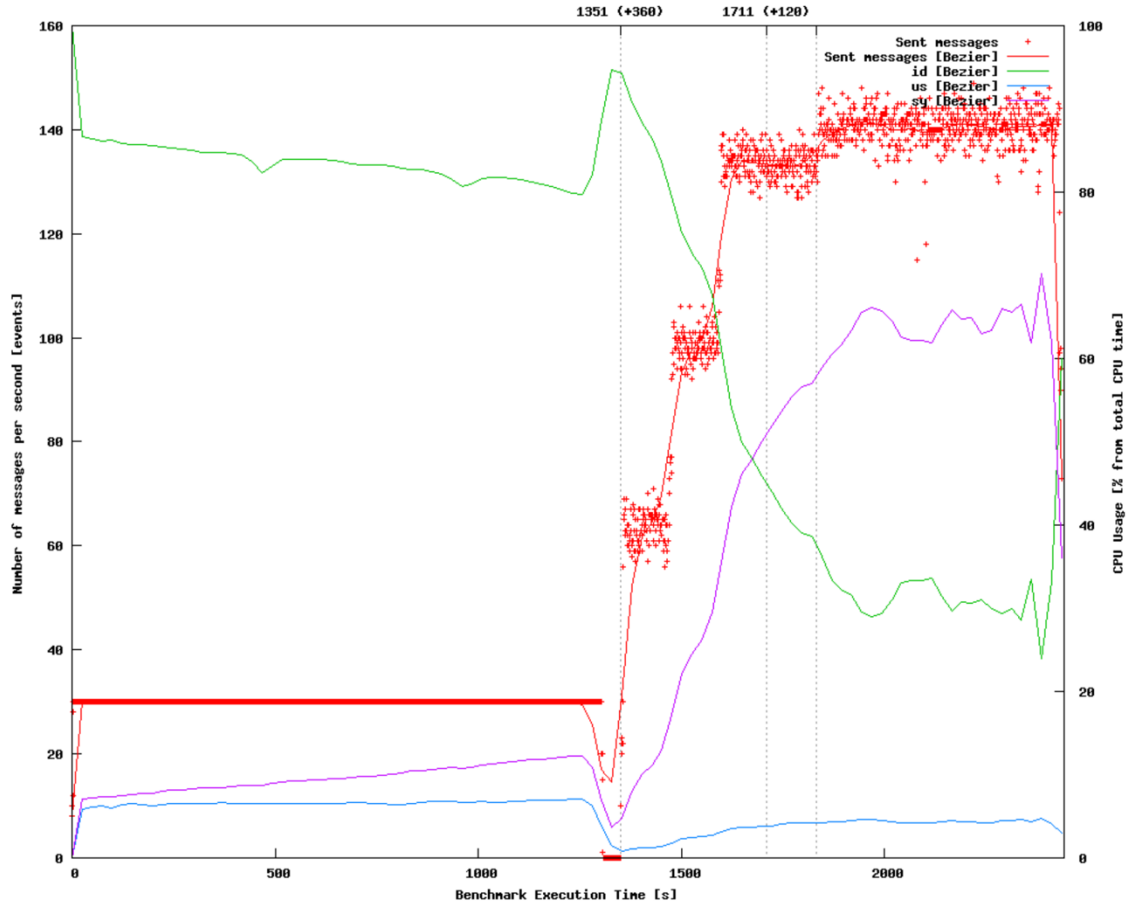

Рисунок 2.9 – Споживання процесора за допомогою поєднання успішних, покинутих, відхилених та невдалих сценаріїв

4.3 Тест 3. Комбінація трафіку

Щоб отримати цінні результати, набір трафіку необхідно ретельно вибирати. Очевидно, що для різних наборів трафіку SUT має різну потужність. Цей результат можна спостерігати також із наступного експерименту. Вибрано два різних набори трафіку. Перший - це набір трафіку, який використовується також в Тест1 (див. підрозділ 2.4.3). Другий набір трафіку містить той самий вибір сценаріїв, але з дещо різними пропорціями (переміщений 2% від успішних сценаріїв до покинутих сценаріїв).

Рисунок 2.10 – Перший набір трафіку, що включає лише успішні сценарії

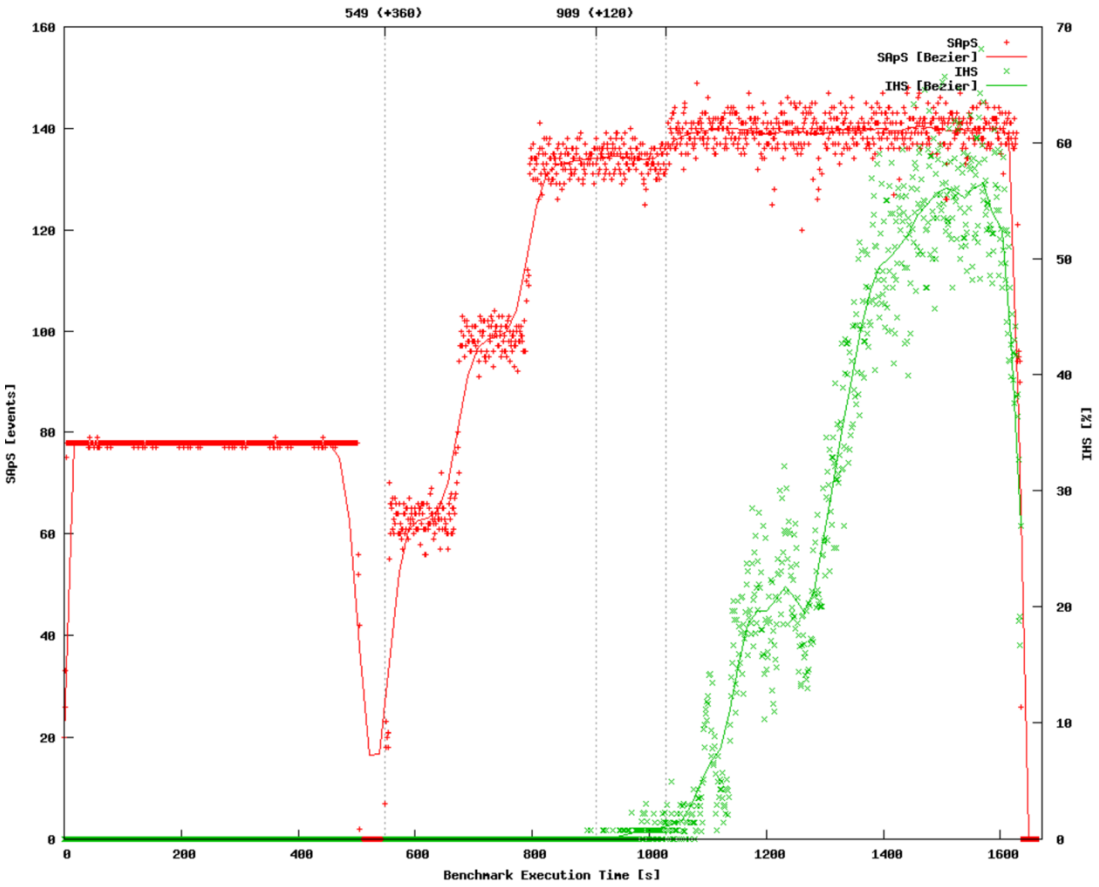

Рисунок 2.11 – Другий набір трафіку, що включає занедбані, відхилені та пошкоджені сценарії

Параметри тестування, які впливають на продуктивність SUT

На результати роботи впливають не тільки комбінація набору трафіку, але й параметри профілю трафіку. Наступні експерименти показують, як модифікація цих параметрів діє на споживання ресурсів на стороні SUT.

4.4 Тест 4. Кількість користувачів

Кількість активних користувачів впливає як на пам'ять, так і на споживання процесора SUT. Пам'ять виділяється для зберігання інформації про стан кожного користувача. Тому чим більше користувачів зареєстровано, тим більше пам’яті виділяється. Процесор необхідний для обробки транзакцій та підписок.

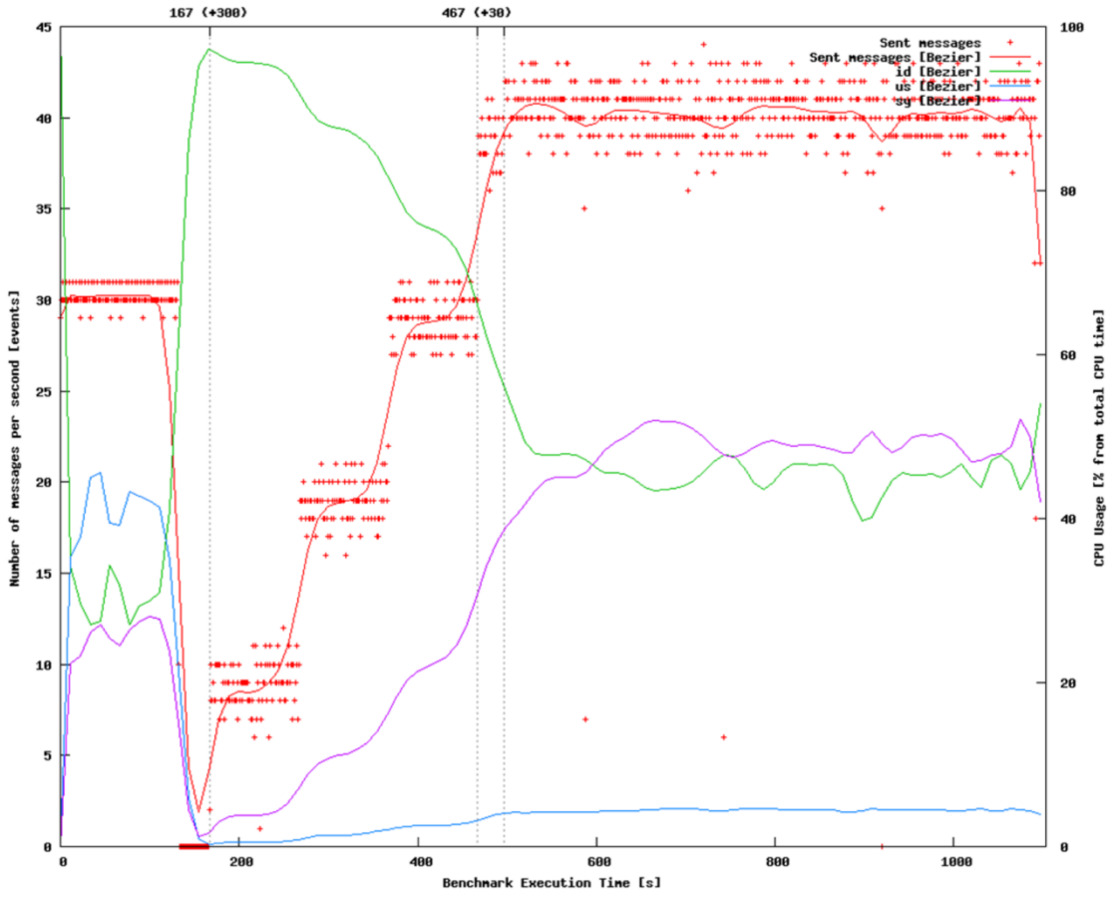

Рисунок 2.12 – Споживання процесора для 5000 користувачів

На рисунках 2.12 та 2.13 показано споживання процесора на два тести з однаковим навантаженням, але з використанням різної кількості користувачів. Якщо в першому тесті вільний процесор становить близько 40%, у другому тесті вільний процесор досягає 0%. Це може здатися дивним, що другий експеримент використовує більше процесора для тієї ж кількості транзакцій, але різниця з'являється через внутрішнє управління станом користувачів (SUT залишає більше пам'яті у другому експерименті).

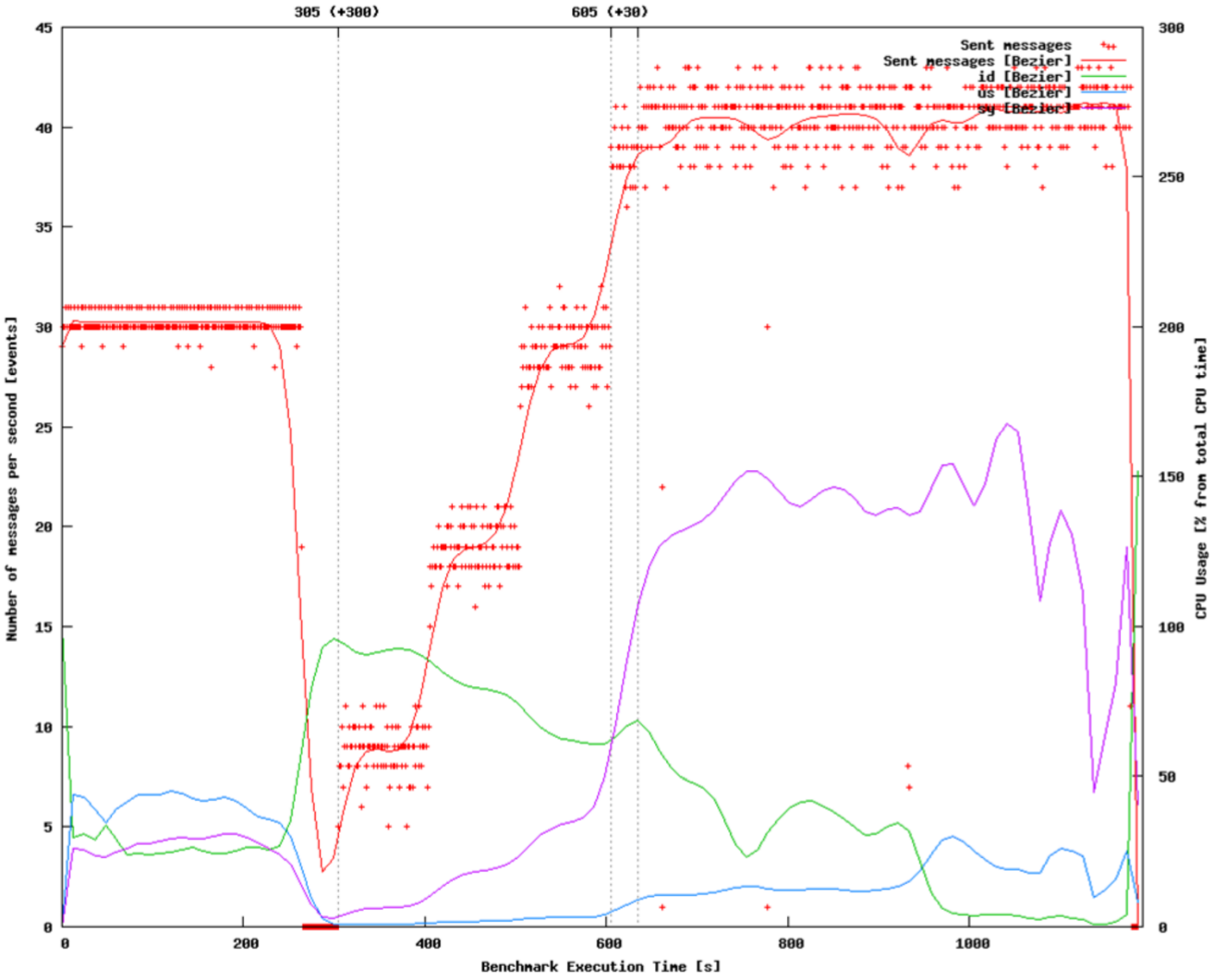

Рисунок 2.13 – Споживання процесора для 10000 користувачів

4.5 Тест 5. Час виконання

Для точних результатів час виконання повинен бути досить довгим, щоб було виконано велику кількість сценаріїв. На рисунку 2.14 показаний тест з двома кроками при 320 і 330 SAPS, виконаних по 10 хвилин кожен. Під час першого кроку SUT тримає навантаження з дуже невеликими відмовами. На другому кроці SUT стає перевантаженим і закінчується невдачею всіх викликів. Згідно з цим тестом, здається, що DOC становить 320 SAPS. Запустивши черговий тест для того ж набору трафіку, але протягом 30 хвилин, виявляється, що SUT досягає межі перевантаження через 15 хвилин. Це означає, що DOC нижче 320 SAPS.

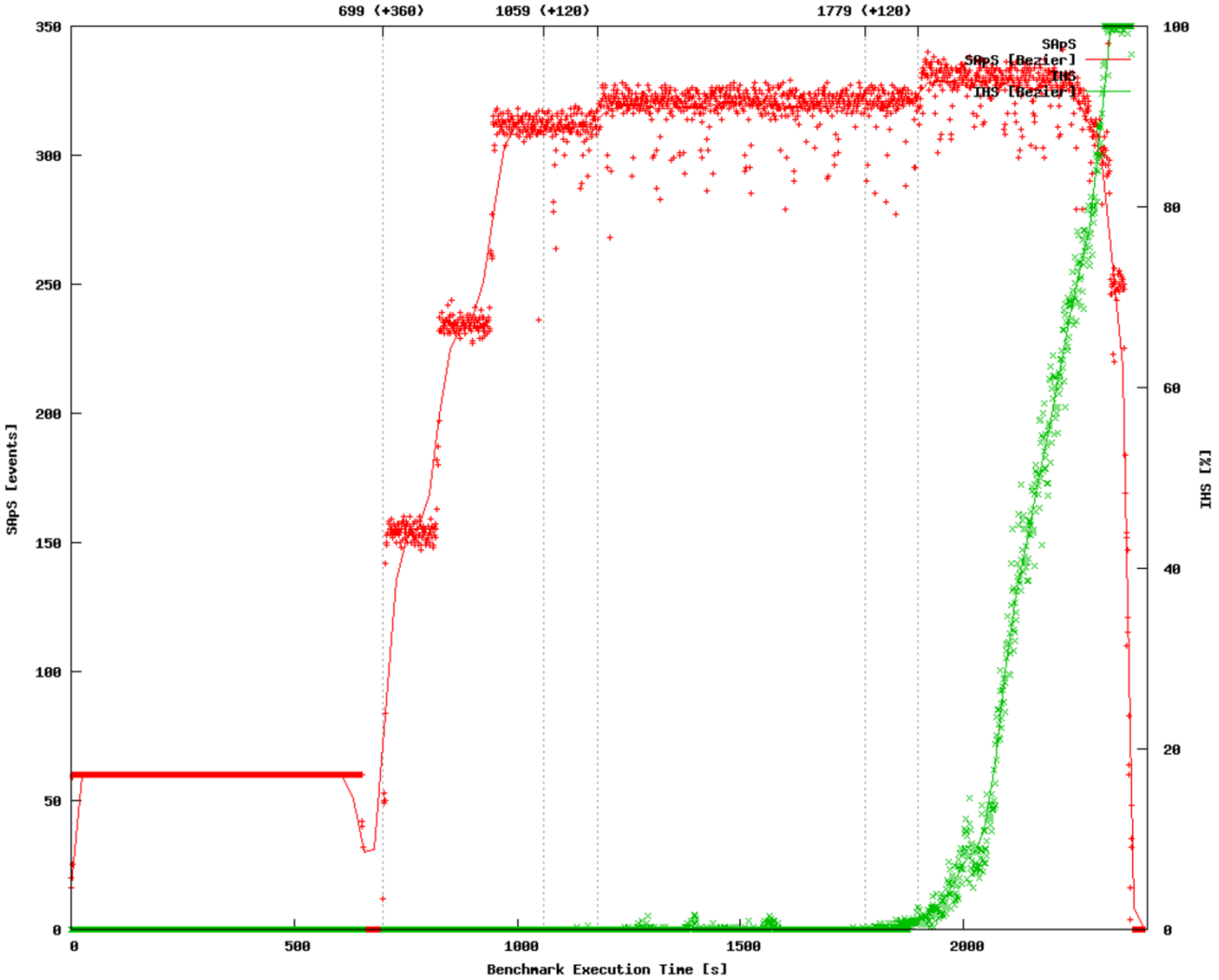

Рисунок 2.14 – Тестовий запуск з 320 SAPS протягом 10 хвилин

Таблиця 2.5 – Тривалість часу виконання відповідно до 1.000.000 SAP

SAPS / сек	Тривалість
100	2,7 години
200	1,3 години
300	55 хв
400	41 хв

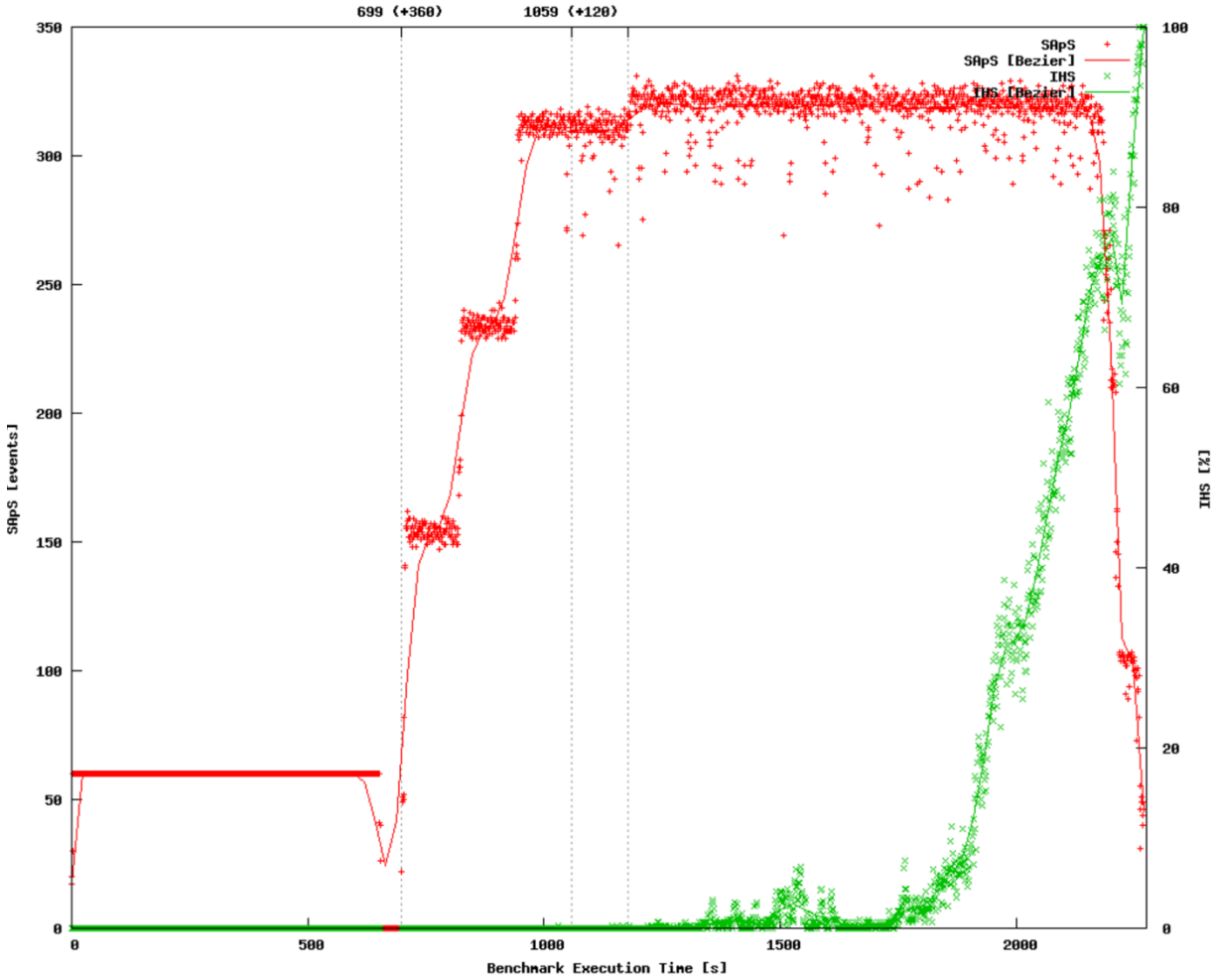

Рисунок 2.15 – Тестовий запуск з 320 SAPS протягом 30 хвилин

Цей експеримент приводить до висновку, що SUT може витримати більші навантаження, але лише на короткий проміжок часу. На тривалість фактично впливає кількість загальних дзвінків, створених у ході тесту. Для різних навантажень, але однакової тривалості, TS створюють різну кількість дзвінків. Адекватне значення кількості дзвінків становить мільйон дзвінків. Ця кількість

дозволяє проводити обчислення статистики з дуже невеликими коефіцієнтами помилок, але також повинна бути достатньо величезною, щоб не дозволяти SUT "виживати", як це робиться для коротких пробігів. У таблиці 2.5 наведені оптимальні тривалості для одного мільйона дзвінків, обчислених для різних навантажень. Для 100 SAPS тест повинен виконуватися протягом 2,7 годин, тоді як для 400 SAPS достатньо лише 41 хвилини.

4.6 Тест 6: Час перемішування

Час перемішування – це параметр, що представляє собою проміжок часу в преамбулі контрольного тесту, в якому завантажується система для того, щоб початкові перехідні умови послаблювалися до незначного рівня. На рисунку 2.16 початкові дзвінки відразу після початку тесту, що впливає на статистику продуктивності.

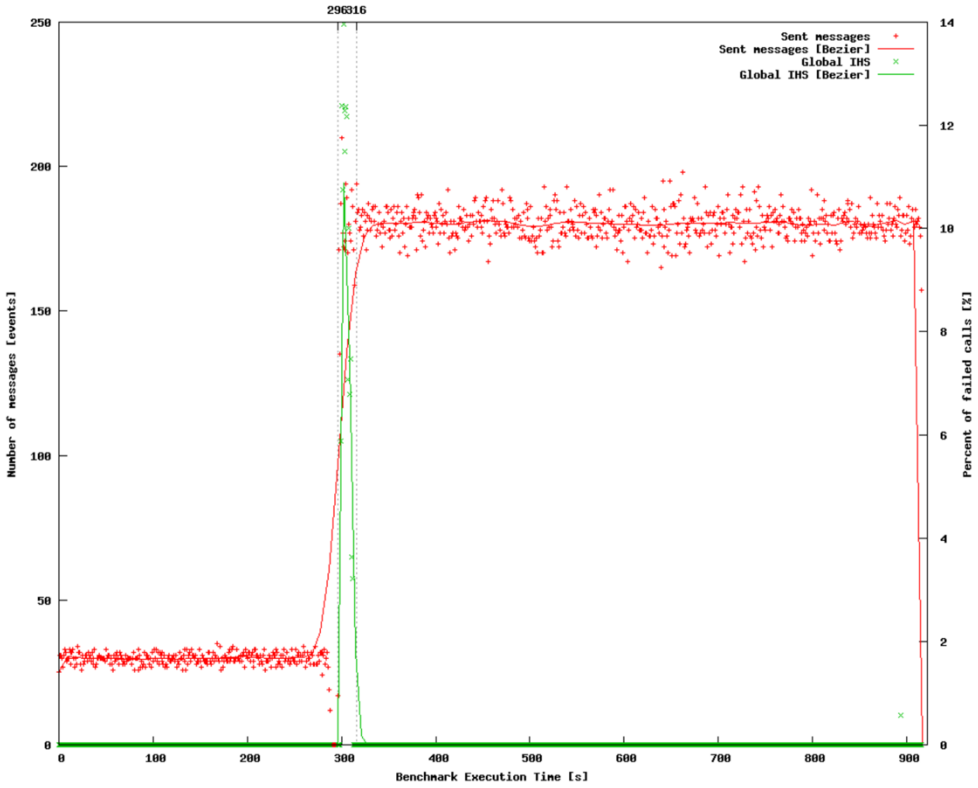

Рисунок 2.16 – Виконання тесту без часу перемішування

Досить довгий час перемішування, наприклад, 10 хвилин, повинен уникати цього ефекту, як це спостерігається в тесті 1 (див. підрозділ 2.3.4), коли цього ефекту не відбувається.

4.7 Тест 7. Перехідний час

Збільшуючи навантаження на новий крок, настає період часу, коли SUT піддається новим навантаженням, але коли він також повинен обробляти існуючі виклики. Оскільки існуючі дзвінки доведеться припинити протягом наступного кроку, є ймовірність, що SUT буде перевантажений новим завантаженням, а попередні припинення виклику не вдасться. У зв'язку з цим явищем попередній крок може перевищити поріг відмови лише тому, що припинення останніх створених викликів відбувається під час другого кроку.

На рисунку 2.17 наведено приклад того, як останні створені дзвінки виходять з ладу лише тому, що SUT досягає ємності перевантаження на останньому кроці. Щоб уникнути цього ефекту, було введено перехідний час. Безпечне значення, як правило, є достатньо довгим, щоб забезпечити припинення всіх дзвінків з попереднього кроку.

У цій главі представлено тестовий випадок веб-додатку. Він спрямований на розробку показника ефективності порівняння конфігурацій розгортання веб-сервісу. Було введено архітектуру підсистеми веб-додатку та виділено елементи тестування продуктивності, які використовуються для проектування еталону. Було проведено ряд експериментів для визначення потужностей та порівняння продуктивності розгортання IMS для різних апаратних конфігурацій.

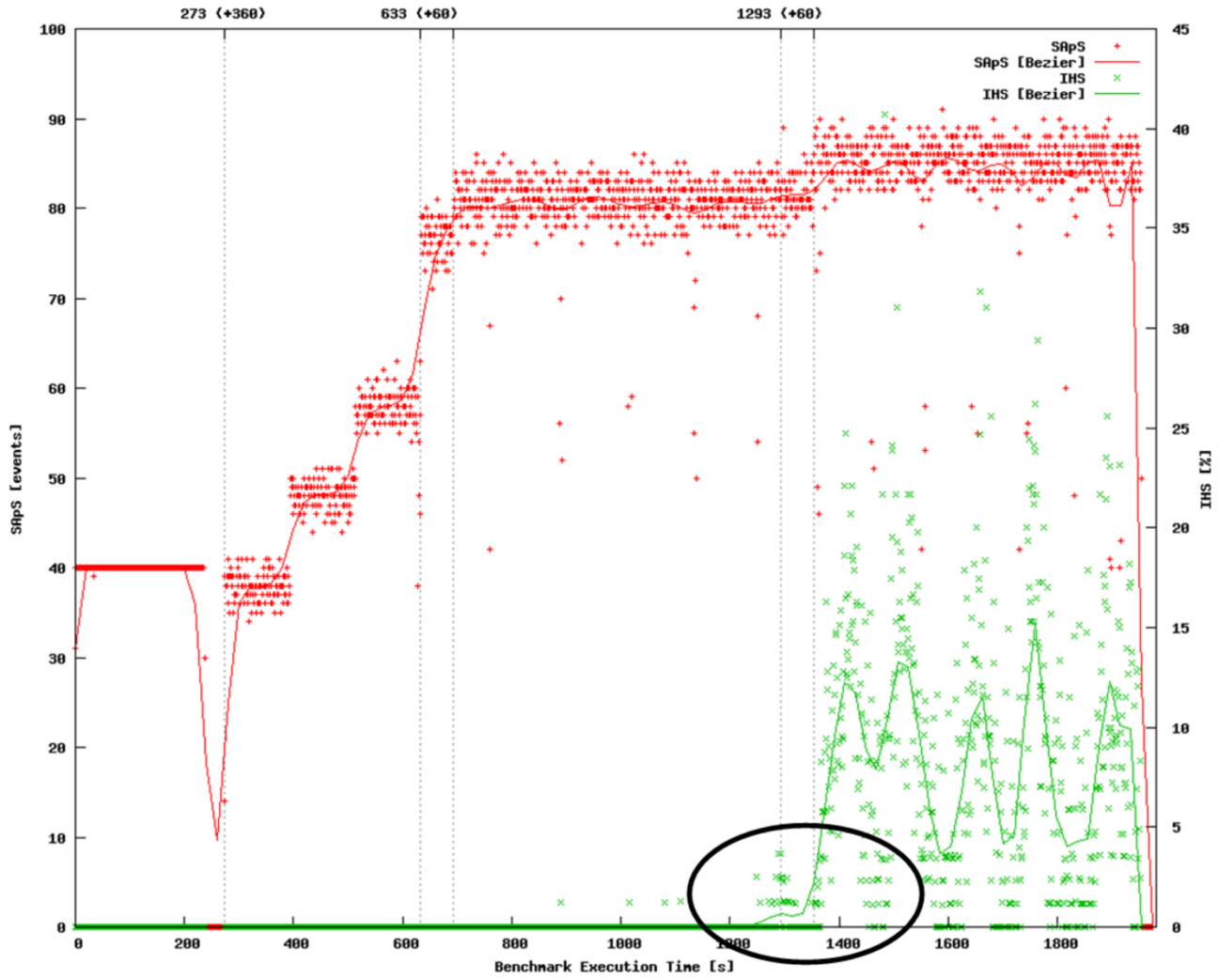

Рисунок 2.17 – Перехідний час

Значна частина розділу присвячена дослідженню параметрів, які впливають на результати роботи. Виявлено, що багато з цих параметрів мають великий вплив на продуктивність SUT. У зв'язку з потребою в реалістичних експериментальних умовах проаналізуються значення цих параметрів.

ВИСНОВКИ

Тестування ефективності послуг, що постійно розвиваються, стає справжньою проблемою через оцінене збільшення кількості абонентів та попиту на послуги. Потрібні більш ефективні та потужні рішення для тестування. Ця здатність сильно залежить від розробки робочого навантаження та ефективного використання апаратних ресурсів для виконання тесту.

У цій дипломній роботі представлена методологія тестування працездатності багатосервісних систем. Тема охоплює виклики сучасних телекомунікаційних технологій, які характеризуються величезною різноманітністю послуг та взаємодій. Основним результатом є метод створення робочих навантажень для тестування працездатності таких систем. Підхід враховує багато факторів: тестові сценарії, тестові процедури, метрики та звітність про тестові показники. Метод проектування гарантує, що робочі навантаження реалістичні та імітують умови, які очікуються у реальному використанні.

Основним показником ефективності є DOS, який повідомляє про найбільше навантаження, яке може витримати SUT, і він служить критерієм порівняння між різними реалізаціями SUT, конфігураціями апаратних засобів SUT тощо.

Робота підходить також до теми тестування в контексті тестування ефективності. Він встановлює вимоги, що стосуються випробувального джгута, до характеристик, характерних для випробувань на продуктивність (включаючи розподіл випробувань). Інструмент Gatling був обран для більш конкретної реалізації представлених концепцій. Наводяться також аргументи для вибору цієї технології. Потім мова використовується, щоб показати, як можуть бути розроблені різні концепції тестування продуктивності, введені в методологію тестування, такі як обробка подій, набір трафіку, сховища даних тощо.

У рамках тематичного дослідження було розроблено набір тестів на ефективність, здатний оцінювати ефективність веб-сервісів за реалістичними навантаженнями. Наявна реалізація забезпечує сценарії з різними наборами

трафіку та збільшенням завантаженості трафіку, і він використовується для диференціювання продуктивності різних конфігурацій.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Раві Кумар Порівняльне дослідження та аналіз засобів тестування веб-сервісів. Міжнародний журнал комп'ютерних наук та мобільних обчислень, т. 4, випуск.1, 2015.

2. Шагун Б.Е. Інструменти тестування продуктивності: порівняльний аналіз, Міжнародний журнал інженерних технологій, Управління та прикладні науки, т. 3 випуск 4, 2015.

3. Офіційний веб-ресурс з документацією Apache Jmeter. URL: https://jmeter.apache.org/ (дата звернення 05.03.2020)

4. Офіційний веб-ресурс з документацією Gatling. URL: https://gatling.io/docs/current/ (дата звернення 05.03.2020)

5. Glenford J. Myers, Tom Badgett, Corey Sandler, The Art of Software Testing, 3rd Edition – Hoboken, New Jersey, John Wiley & Sons, Inc.,.2012

6. Klaus Olsen, Meile Posthuma and Stephanie Ulrich, International Software Testing Qualifications Board - Version V3.1, 2018

7. Graham Bath, Rex Black,Alexander Podelko, Foundation Level Specialist Syllabus Performance -International Software Testing Qualifications Board Testing, 2018

8. G. Din, S. Tolea, and I. Schieferdecker. Distributed Load Tests with TTCN-3. In TestCom, volume 3964 of Lecture Notes in Computer Science, pages 177–196. Springer, 2006.

9. Spirent Communications. Spirent Protocol Tester. http://www.spirent.com, 2020. (дата звернення 10.03.2020)

10. L. K. John and L. Eeckhout. Performance Evaluation and Benchmarking. CRC Press, September 2005.

11. Kirill Smelyakov, Pribylnov Dmitry, Martovytskyi Vitalii, Chupryna Anastasiya. Investigation of network infrastructure control parameters for effective intellectual analysis, 2018

12. Чуприна А.С. Бабій А.С. Дудар З.В., Каук В.І., Ревенчук І.А., Турута О.П., Науковий довідник "Програмна інженерія в Україні", 2019

13. Закон України «Про охорону праці» від 14.10.1992 No 2694 – XII

14. В.І. Голінько Основи охорони праці: підручник / В.І. Голінько – Д.: НГУ, 2014. – 271 с

15. ДСаНПіН «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості ті напруженості трудового процесу», МОЗУ Наказ від 08.04.2014 No248.

16. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» від 01.12.1999.

17. ДБН В.2.5-28-2006. Державні будівельні норми. Природне і штучне освітлення. К.: Мінбуд України,-Київ: 2006.

18. В.В. Березуцький Основи професійної безпеки та здоров'я людини: підручник / В.В. Березуцький – Харків: НТУ «ХПІ», 2018. – 553 с.

19. СанПіН 2.2.2/2.4.1340-03 «Гігієнічні вимоги до персональних електронно- обчислювальних машин та організації роботи» зі змінами від 21.07.2016.

20. ДСТУ Б В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» від 01.01.2017.

21. В.Ц. Жидецький Основи охорони праці / В.Ц. Жидецький – Львів: Афіша, 2004. – 250 с.

22. Закон України «Про охорону навколишнього природного середовища» No 1264-XII від 25 червня 1991 року.

23. Отходы ПК и компьютерной техники: как правильно утилизировать [Електронний ресурс]. – Режим доступу: https://ecologia.life/othody/tehnika/kompyuternaya-tehnika.html (дата звернення 04.05.2019). – Назва з екрану

24. ДСаНПіН 3.3.6-069-2002 Державні санітарні норми і правила при роботі з джерелами електромагнітних полів // Затвердженно наказом МОЗУ від 18.12.2002 No476.

25. ДСТУ ISO 9241-5;2004 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози. – Чинний від 01.01.2006.

26. ДСТУ Б. В.1.1-36:2016 Національний стандарт України. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. – К.: Мінрегіонбуд України, 2016.

27. Правила улаштування електроустановок. – Чинний з 20.11.2014. Затв. Наказом Міністра енергетики та вугільної промисловості України від 20 червня 2014 р. No469.