КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра прикладної статистики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 124 Системний аналіз

на тему:

МЕТОДИ АДАПТИВНОГО КЕРУВАННЯ СТОХАСТИЧНИМИ СИСТЕМАМИ

Виконав студент 4-го курсу
Снітко Нікіта Дмитрович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Пономарьов В.Д.

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри прикладної статистики
«_05_» ___червня_____ 2023 р.,
протокол № 11
Завідувач кафедри
І. В. Розора _____
(підпис)

Київ – 2023

ЗМІСТ

ВСТУП

В сучасному світі, де швидкість змін і необхідність ефективного управління системами стають ключовими факторами успіху, адаптивне керування стає надзвичайно важливим інструментом. Стохастичні системи, що характеризуються випадковими величинами та невизначеністю, зустрічаються у різних галузях, включаючи економіку, промисловість, технології та соціальні системи. Адаптивне керування дозволяє забезпечити ефективну роботу таких систем в умовах невизначеності та змінних умов. З цієї причини, актуальність дослідження полягає у розробці адаптивного керування стохастичними системами, що дозволяє системам, наприклад, малого та середнього бізнесу, бути більшефективними в управлінні своїми процесами, зменшенні часу очікування клієнтів та покращенні якості обслуговування.

Метою даної роботи є аналіз системи і розроблення адаптивного керування стохастичними системами в ресторанній галузі.

Об'єктом дослідження є процесс побудування керування системи масового обслуговування.

Методи дослідження включають аналіз літературних джерел, експериментальні дослідження та математичне моделювання системи з використанням адаптивного керування стохастичними системами.

Керування системою може бути використане у інших галузях, де присутні схожі процеси управління, наприклад, у готельній індустрії або в сфері обслуговування, у фінансовому управлінні, автоматичному керуванні, робототехніці, управлінні виробничими процесами та багатьох інших.

Реалізації мети роботи передував аналіз відповідної літератури, де вивчалися методи адаптивного керування, а також теорія масового обслуговування, яка грає головну роль у аналізі роботи системи.

1 АНАЛІЗ ПІДХОДІВ КЕРУВАННЯ І КЕРУВАННЯ ДИНАМІЧНИМИ СИСТЕМАМИ

1.1 Системи масового обслуговування

1.1.1 Історія масового обслуговування

Масове обслуговування - це процес, при якому велика кількість клієнтів або заявок отримують послуги або продукти від одного або декількох постачальників. Масове обслуговування виникло як наслідок розвитку промисловості, транспорту, зв'язку, торгівлі та інших сфер діяльності людини.

Першими задачами масового обслуговування були задачі, пов'язані з телефонним зв'язком. У 1908-1922 роках співробітник Копенгагенської телефонної компанії Агнер Ерланг розглянув проблеми розподілу телефонних ліній між абонентами, оцінки часу очікування дзвінка та імовірності втрати дзвінка. Він запропонував математичні моделі та методи для розв'язання цих задач, які стали основою теорії масового обслуговування.

Теорія масового обслуговування - це розділ теорії ймовірностей, який досліджує статистичні закономірності в системах масового обслуговування. Система масового обслуговування складається з двох основних елементів: потоку заявок (або клієнтів) та каналу (або постачальника) обслуговування. Поток заявок - це випадковий процес, який характеризується інтенсивністю надходження заявок та їх розподілом за часом. Канал обслуговування - це пристрій або особа, яка виконує певну операцію над заявкою за певний час. Час обслуговування також є випадковою величиною.

Основними питаннями теорії масового обслуговування є: існування та єдиність розв'язку системи масового обслуговування; стабільність розв'язку

системи масового обслуговування; оптимальне керування системою масового обслуговування.

1.1.2 Актуальність масового обслуговування

Масове обслуговування є актуальним, оскільки воно застосовується в багатьох галузях науки і практики. Масове обслуговування дозволяє аналізувати та оптимізувати роботу різних систем, таких як: транспортні системи, логістичні системи, комп'ютерні мережі, банківські системи, медичні системи, промислові системи, торговельні системи та інші. Масове обслуговування допомагає покращити якість обслуговування клієнтів, зменшити час очікування та втрати заявок, збільшити продуктивність та прибутковість систем.

1.1.3 Проблематика масового обслуговування

Масове обслуговування має ряд проблем, які потребують подальшого дослідження та розв'язання. До них належать:

- складність моделювання реальних систем масового обслуговування, які можуть мати нестандартну структуру, неоднорідний потік заявок, залежність часу обслуговування від стану системи та інші фактори;

- невизначеність параметрів систем масового обслуговування, які можуть змінюватися в часі або бути невідомими;

- необхідність урахування психологічних факторів при формуванні черг та впливі на задоволеність клієнтів;

- необхідність врахування економічних факторів при оцінці ефективності систем масового обслуговування та визначенні оптимальної стратегії керування.

Особливо важлива складова у цій роботі це саме невизначеність параметрів. Необхідно буде проводити із системою певні дії, що допоможуть з оволодіванням стабільного функціонування.

1.1.4 Класифікація Кендала

Класифікація Кендалла - це спосіб позначення типів систем масового обслуговування за допомогою трьох або шести символів, які відображають їхні основні характеристики. Класифікація Кендалла має такий вигляд:

$$A/B/C/D/E/F,$$

• *A* - тип потоку заявок, який визначає закон розподілу імовірностей надходження заявок в систему. Найпоширенішими типами потоків є *M* (markovian – експоненціальний), *D* (deterministic - детермінований), *E* (erlang - ерланговий), *G* (general - загальний).

• *B* - тип часу обслуговування, який визначає закон розподілу імовірностей тривалості обслуговування заявки в каналі. Найпоширенішими типами часу обслуговування є M (markovian – експоненціальний), D (deterministic - детермінований), E (erlang - ерланговий), G (general - загальний).

• *C* - кількість каналів обслуговування, яка визначає склад системи. Якщо $C = 1$, то система одноканальна, якщо $C > 1$, то система багатоканальна.

• *D* - ємність накопичувача, яка визначає максимальну кількість заявок, які можуть очікувати в черзі. Якщо $D = \infty$, то система з необмеженою ємністю, якщо $D < \infty$, то система з обмеженою ємністю.

• *E* - розмір джерела заявок, який визначає кількість потенційних заявок, що можуть надходити в систему. Якщо $E = \infty$, то система з необмеженим джерелом, якщо $E < \infty$, то система з обмеженим джерелом.

• *F* - дисципліна формування черги, яка визначає правило вибору заявок з черги для обслуговування в каналі. Найпоширеніші дисципліни формування черги - це *FCFS* (first come, first served - першим прийшов, першим обслужений), *LCFS* (last come, first served - останнім прийшов, першим обслужений), *RANDOM* (random - випадковий вибір), *PRIOR* (priority - за пріоритетом).

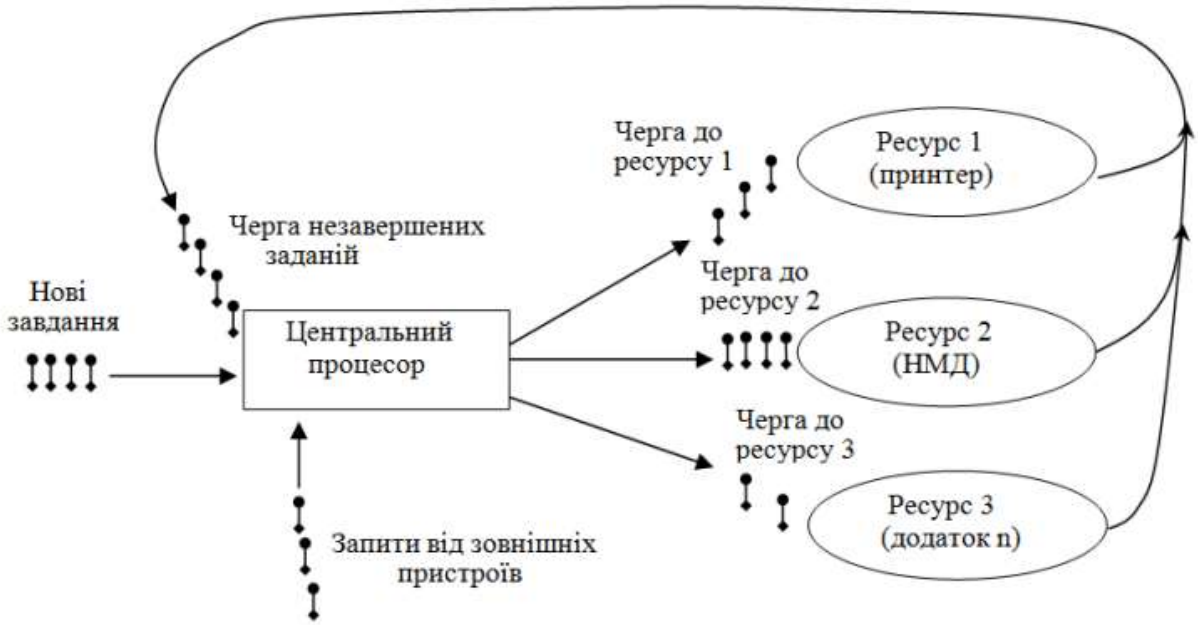

Рисунок 2.1 Черги в мультипрограмній системі (комп’ютер)

Дуже розповсюджено використовується система вузлів з однією чергою. Вона складається з трьох параметрів: *A/B/C*, де *A* вказує на розподіл тривалості між надходженнями до черги, *B* - розподіл часу обслуговування заявок, а *C* - кількість серверів у вузлі.

У цій роботі розглядатиметься система *M/M/C*. Така система є доволі дослідженою системою масового обслуговування.

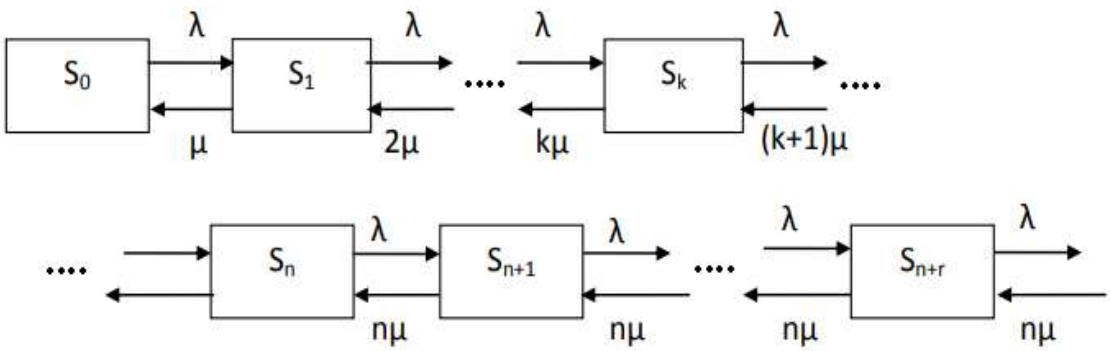

Рисунок 2.2 Граф станів багатоканальної СМО з необмеженою чергою

S_i – стан системи, з i кількістю запитів. S_0 - канали вільні, S_i, $i = \overline{1,n}$ – зайняті канали, S_{n+k} – усі канали зайняті з k замовленнями у черзі.

Обозначимо $p_i(t)$ як ймовірність S_i. Інтенсивність надходження до черги та швидкість обслуговування позначаються як λ та μ.

Для оцінки ефективності системи також використаємо закон Літтла, що каже, що добуток середьного часу перебування у системі і середього темпу прибування буде дорівнювати середній кількості людей у системі.

$L = \lambda W$, де W – середній час перебування у системі.

Модель може бути описана як ланцюг Маркова, де час неперервний з матрицею швидкості перехода:

Для M/M/1:

$$Q = \begin{pmatrix} -\lambda & \lambda & & & & & & \\ \mu & -(\mu+\lambda) & \lambda & & & & & \\ & \mu & -(\mu+\lambda) & \lambda & .... & & & \\ & & & -(\mu+\lambda) & .... & ... & & \\ & & & & .... & ... & & . \\ & & & & & & & \\ & & & & & .... & & \\ & & & & & ... & ... & \\ & & & & & & ... & \end{pmatrix}$$

Ця модель є типом процессу народження-смерті.

Та для M/M/N:

$$Q = \begin{pmatrix} -\lambda & \lambda & & & & & & & \\ \mu & -(\mu+\lambda) & \lambda & & & & & & \\ & 2\mu & -(2\mu+\lambda) & \lambda & .... & & & & \\ & & & -(3\mu+\lambda) & .... & & ... & & \\ & & & & .... & & ... & & . \\ & & & & c*\mu & (-c*\mu+\lambda) & & & \\ & & & & & c*\mu & .... & & \\ & & & & & & ... & ... & \\ & & & & & & & ... & \end{pmatrix}$$

1.2 Керування системами

1.2.1 Задачі керування системами

Керування є важливою складовою багатьох систем, які можуть бути широко застосовані в різних сферах, включаючи технологію, інженерію, економіку, та більш масових, у сенсі загальної маси людей, системах масового обслуговування. Упровадження керування може сприяти покращенню продуктивності, зниженню витрат та оптимізації процесів.

Історія теорії керування сягає свої коріння до давніх часів, коли люди вперше почали задумуватися про контроль та управління різними процесами. Однак, сучасна теорія керування виникла в середині 20-го століття і зазнала значних розвитків.Одним з перших вчених, який вніс вагомий внесок у теорію керування, був Рудольф Калман. У 1960-их роках він розробив фільтр Калмана, що став важливим інструментом для оптимального керування системами з випадковими величинами.Також великий внесок у розвиток теорії керування внесли Норберт Вінер та Джордж Заде. Норберт Вінер є одним з піонерів кібернетики, галузі, яка досліджує системи керування та комунікації. Він вперше сформулював основні поняття та принципи кібернетики, такі як зворотній зв'язок та керована система.

Розвиток теорії керування почався в середині XX століття і був сильно пов'язаний з розвитком електроніки та обчислювальної техніки. На ранніх етапах були розроблені традиційні види керування, такі як пропорційно-інтегрально-диференціальні контролери, які використовувалися для стабілізації систем та підтримки постійного стану. Протягом часу, з появою нових технологій та підходів, було розроблено різноманітні методи керування, такі як оптимальне керування, прогнозуюче керування, регулятори зворотного зв'язку та інші.

Існують різні задачі керування. Наприклад, оптимальне керування полягає в пошуку найкращого керування з точки зору задовільнення деякого критерію якості. Для однієї системи критерії оптимізації можуть бути різними, залежить це

від конкретної системи. Є ситуації, де необхідно оптимізувати систему за декількома параметрами – багатокритеріальна оптимізація, яка буде розглянута пізніше.

Є також керування зворотного зв’язку. Цей метод використовує інформацію про стан системи та її вихід, щоб коригувати керуючий сигнал. Також існує прогнозуюче керування,що використовує моделі системи для прогнозування майбутнього стану та вихідних даних системи. Ці прогнози використовуються для вибору оптимальних керуючих сигналів з метою досягнення бажаного результату.

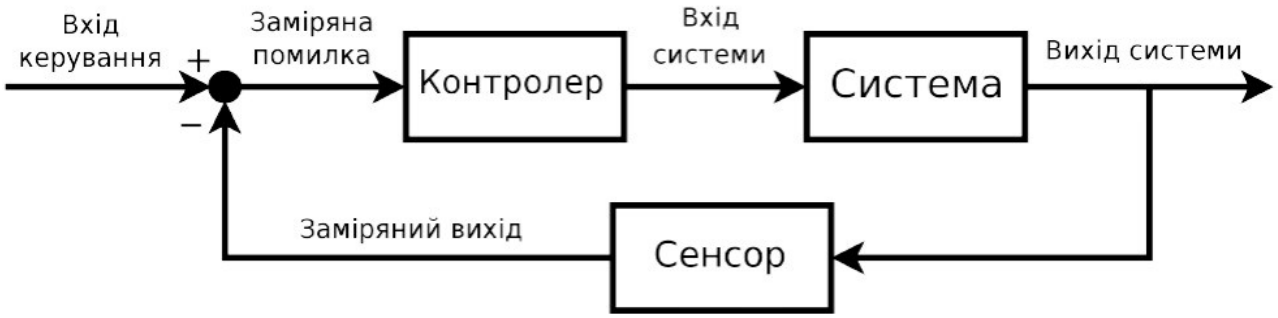

Рисунок 1.1 Схема системи керування із негативним зворотнім зв'язком

Ці принципи можуть бути поєднані між собою у різних комбінаціях.

1.2.2 Адаптивне керування

Адаптивне керування є видом керування, який дозволяє системі адаптуватися до змін у внутрішніх або зовнішніх умовах. Воно базується на використанні алгоритмів адаптації, які коригують параметри керування на основі зібраної інформації про систему та її оточення.

Основна ідея адаптивного керування полягає в тому, що параметри керування не є сталими, а змінюються відповідно до зміни умов. Це дозволяє системі адаптуватися до невідомих або змінних параметрів, що можуть вплинути

на її функціонування. Алгоритми оцінки використовують спостереження зовнішніх вхідних сигналів та вихідних даних для визначення параметрів системи. Після ідентифікації параметрів, керуючий сигнал адаптується відповідно до отриманих значень. Адаптивне керування системою зазвичай відбувається за допомогою математичних моделей або нейромережевих алгоритмів. Система оновлюється залежно від інформації про стан системи, і ця оновлена модель використовується для налаштування параметрів керування.

Таким чином, існує широкий спектр застосувань в різних галузях. Наприклад, у робототехніці для контролю руху роботів, в автомобільній промисловості для керування автомобілями та автопілотами, в електроніці для автоматичного регулювання електронних пристроїв та багато інших сфер. Хоча краще за все використовувати адаптивне керування саме у таких системах, де зараз використовується машинне вивчення інформації, на практиці можна реалізувати й у інших системах, більш простих.

1.2.3 Методи адаптивного керування

Існують різні підходи до адаптивного керування системами. Загалом, слід розрізняти прямі методи, непрямі методи, і гібридні методи.

Прямі методи використовують параметри, що будуть використовуватися у контролері напряму. Непрямі же використовують ці параметри для окремих розрахунків параметрів для контролера. Гібридні використовують обидва підходи.

Одним із методів є MRAC (з англійської Model-reference adaptive control), або модельно-орієнтоване адаптивне керування. Цей метод базується на ідеї порівняння вихідного сигналу системи зі сигналом, що генерується моделлю, до якої прямує система. Основна мета полягає у забезпеченні того, щоб вихідна поведінка системи була близькою до поведінки моделі, незважаючи на зміни у

системі. Для досягнення цієї мети, керуючий сигнал адаптивно коригується на основі розбіжності між вихідним сигналом системи та сигналом моделі посилки.

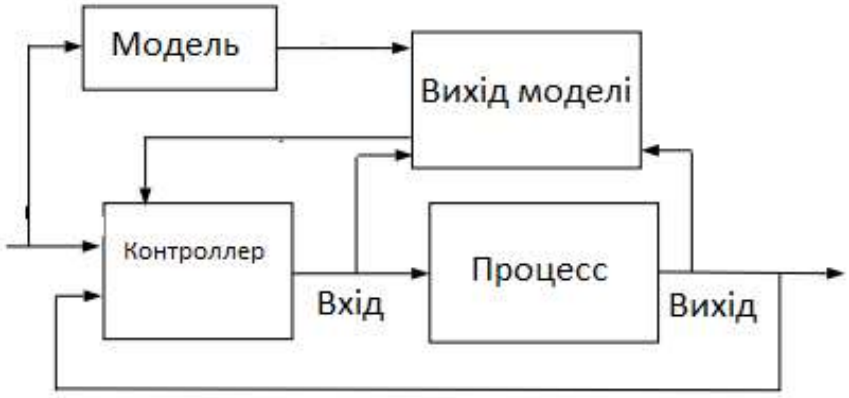

Рисунок 1.3 Схема модельно-орієнтованого адаптивного керування

За структурою модельно-орієнтований метод буде виглядати таким чином:
Об'єкт керування:

$$y(t) = P(z,\theta)u(t) + v(t)$$

Еталонна модель:

$$y_m(t) = M(z)\,r(t)$$

Контролер:

$$u(t) = K(z,\theta)\,[r(t) - y(t)].$$

де $y(t)$ - вихід системи. Це значення представляє фактичний вихідний сигнал системи, який характеризує її поведінку у даний момент часу.

$P(z,\theta)$ - перехідна функція системи. Це математична модель, яка описує взаємозв'язок між вхідним сигналом системи ($u(t)$) і вихідним сигналом системи ($y(t)$). Вона залежить від вектора параметрів θ, який включає налаштовувані параметри системи.

u(t) - вхід керування. Це керуючий сигнал, який подається на систему для керування її поведінкою. Його значення обчислюється за допомогою контролера і залежить від різниці між вхідним сигналом еталонної моделі (r(t)) і вихідним сигналом системи (y(t)).

$v(t)$ – збурення входу. Це додатковий сигнал, який може впливати на систему та викликати непередбачувані зміни у вихідному сигналі системи. Врахування цього збурення у моделі може допомогти покращити точність адаптивного керування. $y_m\ (t)$- вихід еталонної моделі. Це вихідний сигнал еталонної моделі, який використовується цільове значення для системи. Він відображає бажану поведінку системи, яку необхідно досягти.

$M(z)$ - перехідна функція еталонної моделі. Це математична модель, яка описує взаємозв'язок між вхідним сигналом еталонної моделі *(r(t))* і вихідним сигналом еталонної моделі (y_m (t)). Вона визначає бажану $r(t)$ - вхід керування,

$K(z, \theta)$ - Це математична модель, яка визначає взаємозв'язок між різницею між вхідним сигналом еталонної моделі (*r(t)*) і вихідним сигналом системи (*y(t)*), і вхідним сигналом керування (*u(t)*). Ця функція коригує керуючий сигнал для забезпечення адаптації системи до змін у умовах.

θ- вектор оцінюваних параметрів. Це вектор, який містить параметри системи, які можуть змінюватися в процесі роботи системи або через зовнішні впливи. Ці параметри оцінюються адаптивним алгоритмом на основі спостережень за вихідним сигналом системи та вхідним сигналом керування, і використовуються для налаштування контролера.

Іншим методом є метод самостійного самоналаштування, або STC (Self-tuning control). Основна мета полягає у тому, щоб система самостійно налаштовувала свої параметри з метою досягнення оптимальної продуктивності або заданих критеріїв якості. Це досягається шляхом використання адаптивних алгоритмів, які оновлюють параметри керування на основі зворотного зв'язку та аналізу даних з реального часу. Принцип самоналаштовуючого керування полягає

в тому, щоб система керування могла адаптуватись до змін в своєму оточенні або параметрах системи без необхідності вручну змінювати свої параметри. Система має "навчитися" оптимальним параметрам на основі реальних даних зворотного зв'язку.

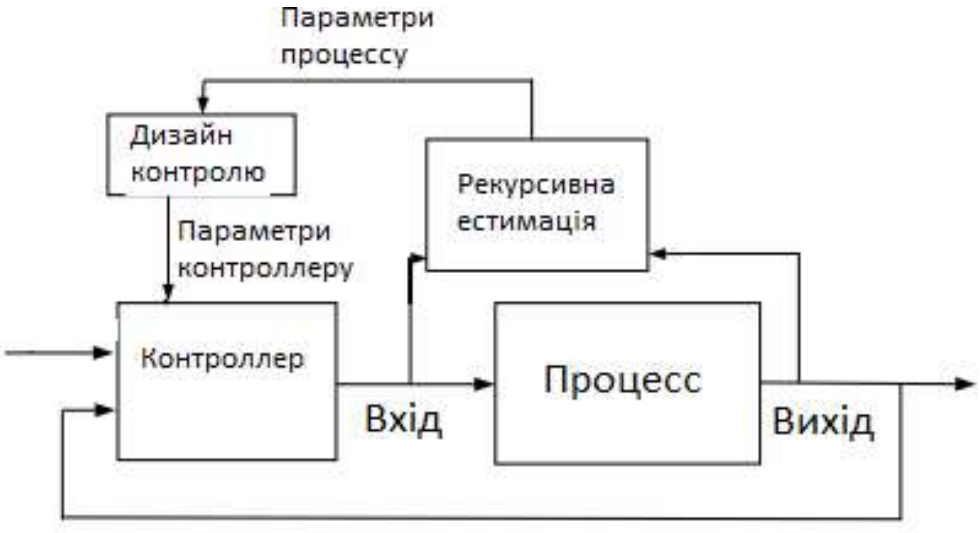

Рисунок 1.4 Схема самоналаштовуючогося адаптивного керування

Основна структура системи може бути представлена наступними чином:

Контролер:

$$u(t) = K(z, \theta(t))\,[r(t) - y(t)].$$

Оцінка параметрів:

$$\theta(t+1) = f(\theta(t), y(t), u(t))$$

де *f(θ(t),y(t),u(t))*- алгоритм онлайн оцінювання параметрів.

Існує також подвійне керування (Dual Control). Основна ідея подвійного керування полягає в тому, що система керування одночасно використовує дві стратегії: оптимальне керування і наближене адаптивне керування. Оптимальне керування базується на заздалегідь відомій моделі системи та математичних методах оптимізації, щоб знайти оптимальні керуючі сигнали. Наближене

адаптивне керування використовує адаптивні алгоритми для наближення оптимальних параметрів системи, використовуючи реальні дані зворотного зв'язку. Застосування подвійного керування може бути ефективним в різних галузях, де важко або неможливо побудувати точну модель системи.

Рисунок 1.5 Схема подвійного керування

Основна структура системи може бути представлена наступними чином:

Контролер:

$$u(t) = K(z, \theta(t))\,[r(t) - y(t)].$$

Оцінка параметрів:

$$\theta(t+1) = f(\theta(t), y(t), u(t))$$

Це базові параметри кожного з керувань. Кожен з цих методів був вивчений, модифікований та покращений.

1.2.4 Багатокритеріальні оптимізаційні задачі

Зазвичай, ціль дослідження систем, а в особливості систем масового обслуговування – покращення ефективності роботи. Але не завжди це дається просто. З усіх параметрів системи треба вибраті такі, які будуть забезпечувати найефективніше функціонування системи. Це називається задачею оптимізації.

Параметр, що оптимізується, називається критерієм оптимізації. Критерій може мати як негативне (тобто має значення витрат), тоді необхідно його мінімізувати, так і позитивне значення (прибутковий критерій), і в цьому випадку він максимізується.

Бувають випадки, коли цих параметрів може бути декілька. У цій ситуації необхідним чином потрібно кожен з цих параметрів мінімізувати та максимізувати, відповідно до їх негативного чи позитивного значення. Але у такому випадку стає очевидним, що при, наприклад, максимізації деякого прибуткового критерію, з'являється проблема з неоптимальним витратним критеріїєм. Тоді необхідно розглянути шукати якийсь компроміс, де буде враховуватися важливість кожного критерію.

Нехай є k функціоналів :

$$f_1, f_2, \dots f_n \dots, f_k,$$

де деякі функціонали будуть мати вигляд

$$f_i \to min,\ i = 1,2,\dots,n$$

інші ж будуть

$$f_i \to maxi = n+1,\dots,k.$$

Тоді критерії утворюватимуть векторний критерій :

$$F(x) = (f_1(x), f_2\,(x), \dots f_n(x)\dots, f_k(x))$$

Таку задачу багатокритеріальної оптимізації дуже зручно розв’язувати метод скалярної згортки цих критеріїв. Тоді глобальний критерій оптимальності виглядатиме так:

$F(x) = \sum_{i=1}^{k} a_i f_i(x)$, де $a_i -$ коефіцієнти важливості кожного з критеріїв.

2 МАТЕМАТИЧНА МОДЕЛЬ ТА ПРОГРАМА РЕАЛІЗАЦІЯ

2.1 Описання системи

2.1.1 Математична модель

Згідно класифікації Кендалла, цю систему можна представити як M/M/3. Це означає, що вхідний потік викликів та швидкість обслуговування розподілені експоненціально, в системі присутня необмежена черга та відсутні відмови.

На практиці, ця система представляє собою автомобільний мийний центр з такими параметрами: вхідний потік, позначений як λ , складає 81 особу за годину. Середній час обслуговування автомобіля на автомийці μ становить 2 хвилини. Зазвичай, в цьому автомийці працюють 3 сервери (комірки для автотранспорту), які обслуговують клієнтів. Особливість даної системи полягяє у тому, що це доволі швидка автоматична автомийка. Нажаль, на даний момент, в Україні йде повтомасштабна війна, і через це у цілої країни дуже великі проблеми з електропостачанням. Ця система – не виключення. Працюючи навіть в умовах повного відключення світла, і вже маючи доволі дорогу систему універсальної очистки автомобіля, кожна комірка даної автомийки коштує великих витрат для підтримки стабільної роботи.

2.1.2 Параметри аналізу системи

Задля того, щоб здійснювати якісь аналітичні дії для цієї системи, необхідно визначити параметри, за якими буде здійснюватися аналіз. Такими параметрами у цій системі є:

- Середнє навантаження на систему : $\rho = \frac{\lambda}{\mu}$
- Коефіцієнт серверів, що зайняті : $k = \frac{\rho}{n}$

Важливо те, що k має бути менше одиниці, задля обезпечення нормального функціонування системи.

- Середня кількість клієнтів у черзі : $L_{\text{ч}} = \frac{\rho^{n+1}}{n*n!(1-\frac{\rho}{n})^2}\ P(0)$
- Середня кількість клієнтів, які перебувають в системі : $L = L_{\text{ч}} + \rho$
- Середній час очікування клієнта у черзі : $W_{\text{ч}} = \frac{L_{\text{ч}}}{\lambda}$
- Середній час, який клієнт проводить у системі : W = $W_{\text{ч}} + \frac{1}{\mu}$
- Ймовірність того, що клієнт очікує обслуговування :
- $P_{\text{оч}} = \frac{\rho^{n+1}}{n!(n-\rho)} P(0)$
- Ймовірність того, що у системі перебуває n клієнтів : $P_n = \frac{\rho^n}{n!}\ P(0)$
- Ймовірність відсутності клієнта в системі:

$$P(0) = \left(1 + \frac{\rho}{1!} + \frac{\rho}{2!} + \cdots + \frac{\rho}{n!} + + \frac{\rho^{n+1}}{n!\,(n-\rho)}\right)^{-1}$$

2.1.3 Керування системою та її оптимізація

Для регулювання кількості серверів у системі автомийки реалізовано метод управління на основі правил. Цей підхід не є специфічним для будь-якого відомого методу управління, а скоріше є базововим методом, який використовується для адаптації системи на основі деякого параметру системи. Важливо зазначити, що цей метод управління може не забезпечувати найточнішу оптимальну продуктивність у порівнянні з більш іншими, більш розширеними методами управління, наприклад, такими як модельно-орієнтоване адаптивне керування. Але якщо брати до уваги ресурси, що використовують інши методи, для системи такої структури, а також для наглядності, цього більш ніж достатньо.

Тож, у нас є система автомийка, якак працює в умовах постійних перепадів електропостачання. Цю проблему можна вирішити різними способами, один із них це знайти потужне та надійне джерело енергії. Задля підтримання стабільності системи, кожна комірка починає приносити більшу кількість втрат. Тож, відтепер,

задля оптимізації системи необхідно обмежити кількість серверів. Тому необхідно створити керування.

По суті, оскільки система стохастична, швидкість прибуваючого потоку та обслуговування випадкові. Це дає змогу стверджувати, що при даних умовах, наявних у задачі, можуть виникати моменти, коли при занизькому рівню серверів починає утворюватися черга, а при зависокому збільшуються витрати, оскільки кожен сервер дорого коштує. При виборі кількості серверів, в першу чергу дивляться на співвідношення вхідних даних між собою, тобто інтинсивності появи викликів та швидкість їх обслуговування. На практиці, як вже було сказано, буде утворюватися черга. Тому саме на неї треба орієнтуватися при побудові керування. Кореляція очевидна, і не потребує перевірки. Порогове значення дорівнює 10.

Оскільки може бути тільки три варіанти, а саме :

1. Кількість серверів дорівнює 3
2. Кількість серверів дорівнює 4
3. Кількість серверів залежить від актуального стану системи, і може бути як 3, так і 4,

то це дає змогу перевірити кожну стратегію окремо. В цьому і буде полягати задача.

Оскільки переключення одного стану систему на другий потребує ресурсів, необхідно встановити лічильник, який зменшить шанс марного переключення. Суть роботи керування проста. Є порог – поточна кількість автомобілів у черзі. Якщо достатньо часу кількість автомобілів перевалює за порог, то стан системи змінюється, лічильник переключень збільшує своє значення, а система продовжує свою роботу. З вище сказаного випливає, що ми маємо діло із задачею багатокритеріальної оптимізації. У цьому випадку ми маємо прибутковий критерій обслугованих клієнтів, який ми маємо максимізувати, та два критерія витрати – кількість активних серверів, та кількість перемикань станів системи.

У цей момент вступає у діло функція якості. Вона відповідає за перевірку оптимальності стратегії. Функція якості для даного випадку виглядає так:

$$L(H) = C_1 L_1(H) - C_2 L_2(H) - C_3 L_3(H) \rightarrow \max$$

Де С1 – коефіцієнт прибутку від кількості обслугованих, $L_1(H)$ – кількість обслугованих, C_2 – коефіцієнт втрат від активних серверів, $L_2(H)$ – кількість активних серверів, C_3 – коефіцієнт втрат від перемикань станів системи, $L_3(H)$ – кількість перемикань системи, H_1, H_2, H_3 – стратегії керування. Коефіцієнти розподілені таким чином : 120, 5, та 2 відповідно.

2.2 Особливості програмної реалізації

Керування системами масового обслуговування має велике значення в сучасному світі. Особливий підхід потрібен для систем, які мають стохастичний характер, оскільки вони включають елементи невизначеності та випадковості. У цьому контексті існують різні типи систем та методи керування ними. Одним з підходів до дослідження та керування системами масового обслуговування є побудова моделей імітації, що дозволяє аналізувати та оптимізувати їх роботу.

Для побудови та аналізу моделі системи масового обслуговування використовувалися різні інструменти та підходи. У середовищі PyCharm за допомогою мови програмування Python було реалізовано програмний код, який забезпечував генерацію вхідного потоку викликів з відповідними характеристиками, реалізацію черги та розподілу автомобілів між доступними серверами. Код також враховував статистику часу очікування клієнтів у черзі, часу обслуговування та інші показники продуктивності системи.

Для того, щоб побудувати систему таку систему, була встановлена спеціальна бібліотека для Python, що називається Simpy. Ця бібліотека надає засоби для моделювання та симуляції подійних систем. Вона дозволяє

розробникам створювати моделі, які відображають реальні події та процеси, і виконувати симуляції для вивчення поведінки системи в різних умовах.

SimPy базується на концепції подійної моделі, де система складається з окремих подій, які відбуваються в певні моменти часу. Кожна подія може мати свою тривалість і впливати на інші події у системі. За допомогою SimPy можна моделювати різні види систем, такі як черги, мережі зв'язку, процеси виробництва та інші. Саме це і необхідно для виконання задачі. У роботі також використовується бібліотека Random, для реалізації експоненційного розподілу.

```
class CarWash:
    def __init__(self, env, num_servers, service_rate):
        self.env = env
        self.server = simpy.Resource(env, num_servers)
        self.service_rate = service_rate

    def wash(self, car):
        yield
self.env.timeout(max(random.expovariate(self.service_rate), 1))

    def change_servers(self, num_servers):
        self.server = simpy.Resource(self.env, num_servers)
```

Це метод, що симулює миття автомобілів системи. Дія цієї системи – метод wash(), що відповідає за експоненційний розподіл часу обслуговування. Оскільки є процедури, які неможливо пропустити при митті автомобіля, було виставлено обмеження : неможливо закінчити обслуговування менше, ніж за 1 хвилину.

Функція car() представляє процес прибуття та обслуговування автомобіля в автомийці. Вона записує дані про дії автомобілів у список car_par, де вони будуть зберігатися для подальшого використання. Також є спеціальні списки que та que_sys, які зберігають дані про кількість знаходження клієнтів у тому чи іншому стані. До них додана que2, яка також зберігає дані про чергу, але у іншому вигляді, який буде використаний у подальшому методі керування.

```
     def car(env, name, num, cw):
    car_par.append([])
    print(f'{name} arrives at the car wash at {env.now}')
```

```
    car_par[num - 1].append(env.now)
    if num == 1:
        que.append(1)
        que_sys.append(1)
        que2[0]=1
    else:
        que.append(que[-1] + 1)
        que_sys.append(que_sys[-1] + 1)
        que2[0]+=1
    arrival_time = env.now
    with cw.server.request() as request:
        yield request
        print(f'{name} enters the car wash at {env.now}')
        que.append(que[-1] - 1)
        que2[0] -= 1
        car_par[num - 1].append(env.now)
        yield env.process(cw.wash(name))
        print(f'{name} leaves the car wash at {env.now}')
        car_par[num - 1].append(env.now)
        print(f'{name} spent {env.now - arrival_time} minutes in the
system')
        car_par[num - 1].append((car_par[num - 1][1] - car_par[num -
1][0]))
        car_par[num - 1].append((car_par[num - 1][2] - car_par[num -
1][0]))
        que_sys.append(que_sys[-1] - 1)
```

Якщо ми приймаємо рішення використати адаптивне керування, необхідно реалізувати це керування. До реалізації системи масового обслуговування додаємо контролер, функцію adaptive_control(). Вона регулює кількість серверів у системі автомийки на основі спостережуваної довжини черги та середнього часу очікування, та виконує адаптивне керування кількістю серверів в системі на основі поточного стану черги, викликаючись в основному циклі симуляції кожну одиницю часу.

На основі показників, що знаходяться та були знайдені раніше, вона приймає рішення щодо зміни кількості серверів у системі. Керування відбувається за допомогою змінної num_servers. Якщо довжина черги перевищує порогове значення, і це триває протягом трьох послідовних часових кроків, кількість серверів збільшується на одиницю. Якщо довжина черги залишається нижче порогового значення і триває протягом трьох послідовних часових кроків, кількість серверів зменшується на одиницю.

Додана функція виглядяє так :

```
def adaptive_control(env, carwash, num_servers, threshold,
is_adapt):
    if is_adapt == 1:
        ccount = 0
        ccount2 = 0
        while True:
            yield env.timeout(1)


            queue_length = que2[0]

            if queue_length >= threshold and num_servers != 4:
                ccount += 1
                ccount2 = 0
                if ccount == 3:
                    ccount = 0
                    num_servers = 4
                    switches[0] += 1
                    numserv.append(4)

                    carwash.change_servers(num_servers)
            elif queue_length < threshold:
                ccount = 0
            if queue_length < threshold and num_servers == 4:
                ccount = 0
                ccount2 += 1
                if ccount2 == 3:
                    ccount2 = 0
                    num_servers = 3

                    switches[0] += 1
                    numserv.append(3)
                    carwash.change_servers(num_servers)
            elif queue_length > threshold:
                ccount2 = 0
```

```
            else:
                numserv.append(NUM_SERVERS)
```

Дана функція займається переключенням станів системи з одного у інший. Усього їх два : кількість серверів – 3, та кількість серверів – 4.

Після цього, створюємо метод setup(), який запускає процеси вхідних викликів та їх обслуговування.

```
def setup(env, car_wash, num_servers, service_rate, arrival_rate):
    i = 0

    while True:
        yield env.timeout(random.expovariate(arrival_rate))
        i += 1
        env.process(car(env, f'Car {i}', i, car_wash))
```

Основна частина програми: В цій частині коду встановлюються параметри моделювання, створюється середовище env, створюється об'єкт автомийки carwash з відповідними параметрами, і запускаються процеси генерації автомобілів та адаптивного керування. Симуляція запускається, і процес починається.

Після закінчення симуляції виводяться розраховані показники, такі як середня кількість у черзі, середній час очікування клієнта, ймовірність відсутності машини у системі та ймовірність присутності машини у черзі.

Тепер, врешті решт, запускаємо модель, та даємо їй працювати 600 “хвилин”.

```
env = simpy.Environment()

car_wash = CarWash(env, NUM_SERVERS, SERVICE_RATE)
env.process(setup(env, car_wash, NUM_SERVERS, SERVICE_RATE,
ARRIVAL_RATE))
env.process(adaptive_control(env, car_wash, NUM_SERVERS,
threshold=10, is_adapt=adapt))
env.run(until=SIM_TIME)
```

Так виглядає процес:

….

Car 253 arrives at the car wash at 131.02346595397427

Car 227 leaves the car wash at 131.06549952790485

Car 227 spent 14.326338434129212 minutes in the system

Car 228 enters the car wash at 131.06549952790485

Car 254 arrives at the car wash at 131.17650887793076

Car 255 arrives at the car wash at 131.25114952513565

Car 225 leaves the car wash at 131.51342727520247

Car 225 spent 15.248547222526554 minutes in the system

Car 229 enters the car wash at 131.51342727520247

Car 256 arrives at the car wash at 131.66908437896092

Car 257 arrives at the car wash at 131.69436916679302

….

2.3 Апробація та аналіз результатів

Тепер, маючи імітацію роботи системи, яка триває 10 годин (що має значення, оскільки автомийка цілодобова), можна створити імітацію системи та знайти значення параметрів, необхідних для аналізу.

ρ	2.7
K	0.9
$L_{\text{ч}}$	22.354078549848943
L	25.36319612590799
$W_{\text{ч}}$	14.505484216867156
W	16.766374156114853
$P_{\text{оч}}$	0.9716012084592145
$P(0)$	0.0006053268765133172

Таблиця 2.2 Параметри системи

Середня кількість у черзі дорівнює 22.354, а середня кількість в системі (у черзі та обслуговуванні) дорівнює 25.363.

Середній час очікування клієнта у черзі є рівним 14.50, середній час присутності клієнта в системі рівний 16.76.

Хоча ці значення є не ідеальними, для конкретної системи з високою інтенсивністю викликів вони є в межах норми.

Ймовірність відсутності машини у системі рівна 0.006. Це означає, що в середньому лише 0.6% часу система є вільною (без автомобілів). Результат нормальний.

Ймовірність присутності машини у черзі - 0.971 що означає, що в середньому, близько 97.1% часу система має автомобілі у черзі.

Аналізуючи ці значення, можна зробити наступні спостереження. Система має доволі велику кількість автомобілів у черзі та в системі загалом. Це може означати, що кількості серверів і їх продуктивності недостатньо для виконання потоку автомобілів.

Середній час очікування клієнта у системі в цілому також є доволі великим.

Ймовірність відсутності машини у системі є невеликою, що показує ефективне використання системи.

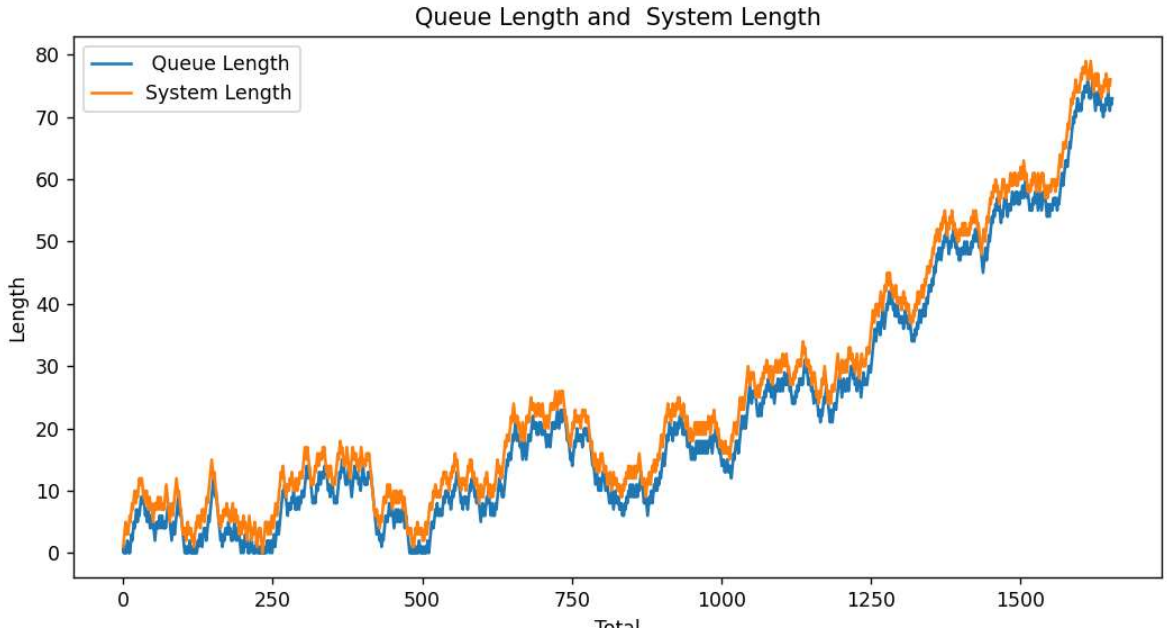

Рисунок 2.1 Прогресія черги та загальної кількості автомобілей у системі

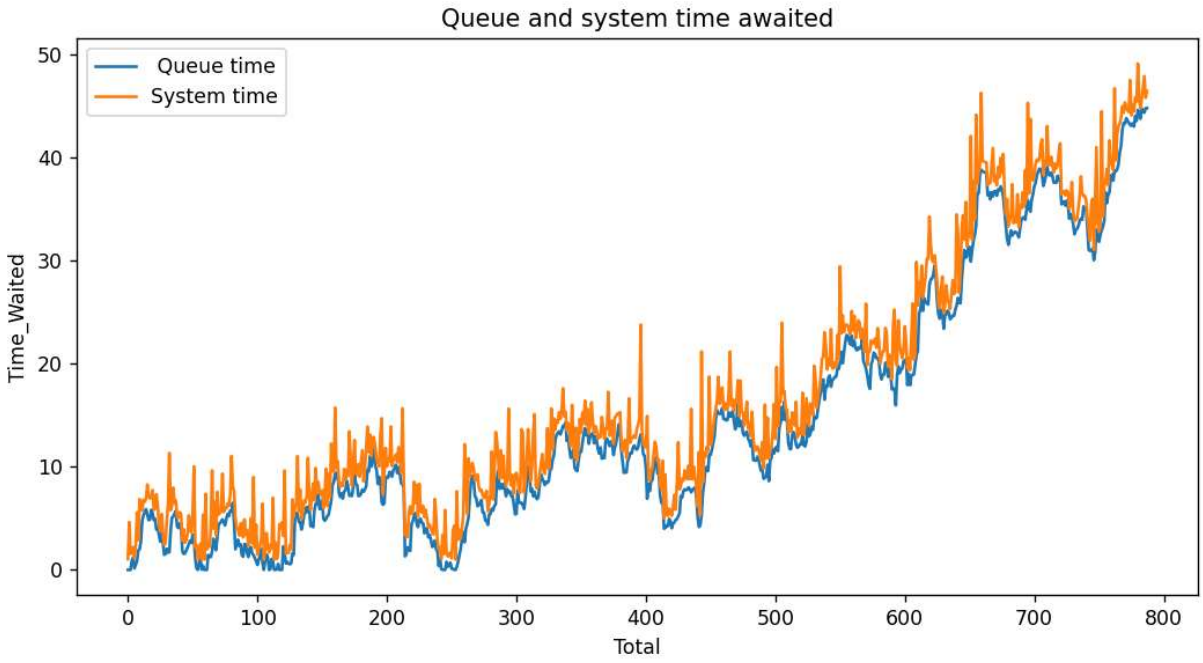

Рисунок 2.2 Прогресія часу очікування у черзі та часу проведеного у системі

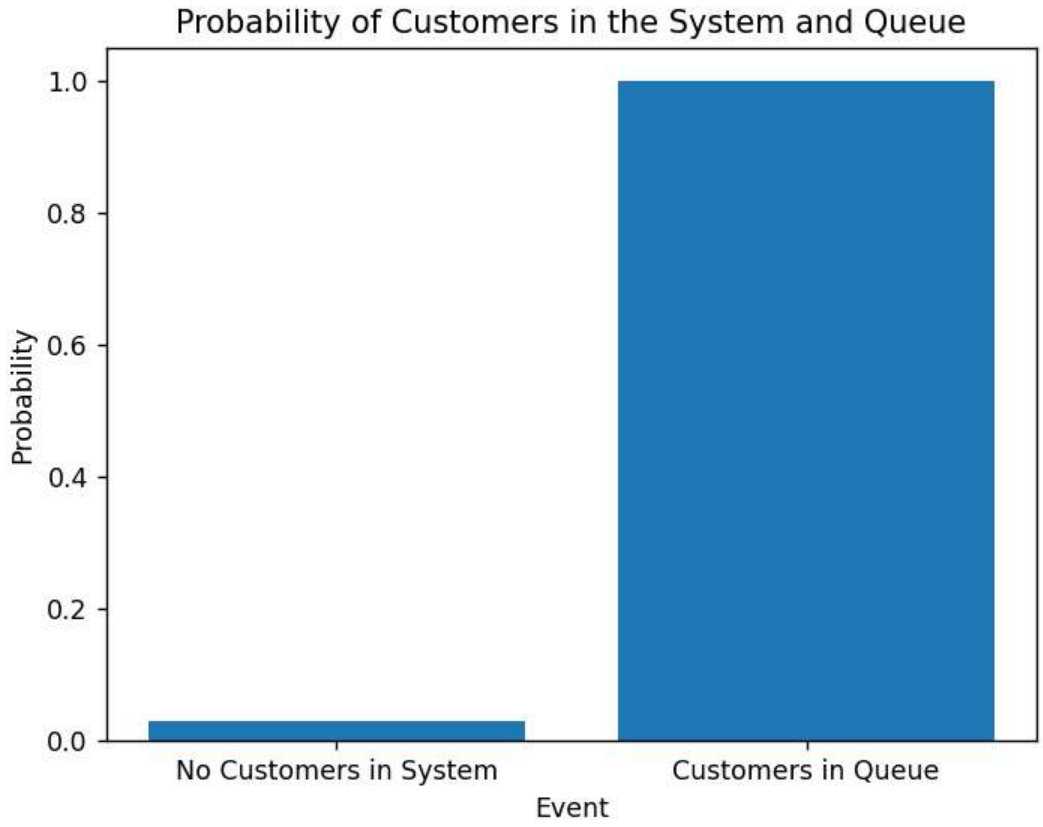

Рисунок 2.3 Ймовірності системи

Також, варто сказати про варіацію вхідних параметрів для системи. Тобто, для того, щоб оцінити роботу системи, можна провести тестування з різними параметрами. Але при цьому важливо пам’ятати, що має зберігатися умова $k < 1$.

При невиконанні умови, з часом, буде створюватися проблема зростаючої черги, тобто система не буде виконувати свою задачу ефективно. Найпростіший варіант покращити результат – це збільшити кількість працюючих серверів. Тобто, у нашому випадку, з 3 до 4. Тож, змінюємо параметр, а саме – кількість серверів.

ρ	2.7
k	0.675
$L_{\text{ч}}$	1.5267965895249695
L	4.475579975579976
$W_{\text{ч}}$	0.7689599786567457
W	2.931989081258022
$P_{\text{оч}}$	0.6766138855054811
$P(0)$	0.018925518925518924

Таблиця 2.2 Параметри системи з чотирма серверами

Порівнюємо результати з системою, де кількість серверів дорівнює трьом, і відразу бачимо, що параметри оптимізувалися. Середня кількість у черзі та системі 1.5267965895249695 та 4.475579975579976 відповідно, середній час очікування та

присутності 0.7689599786567457 та 2.931989081258022 відповідно означають однозначний ріст ефективності системи. Також ймовірність очікування у системи знизилася приблизно на 30%. Водночас ймовірність того, що у системі немає жодного автомобіля зросла приблизно у 4 рази, що буде згадано пізніше.

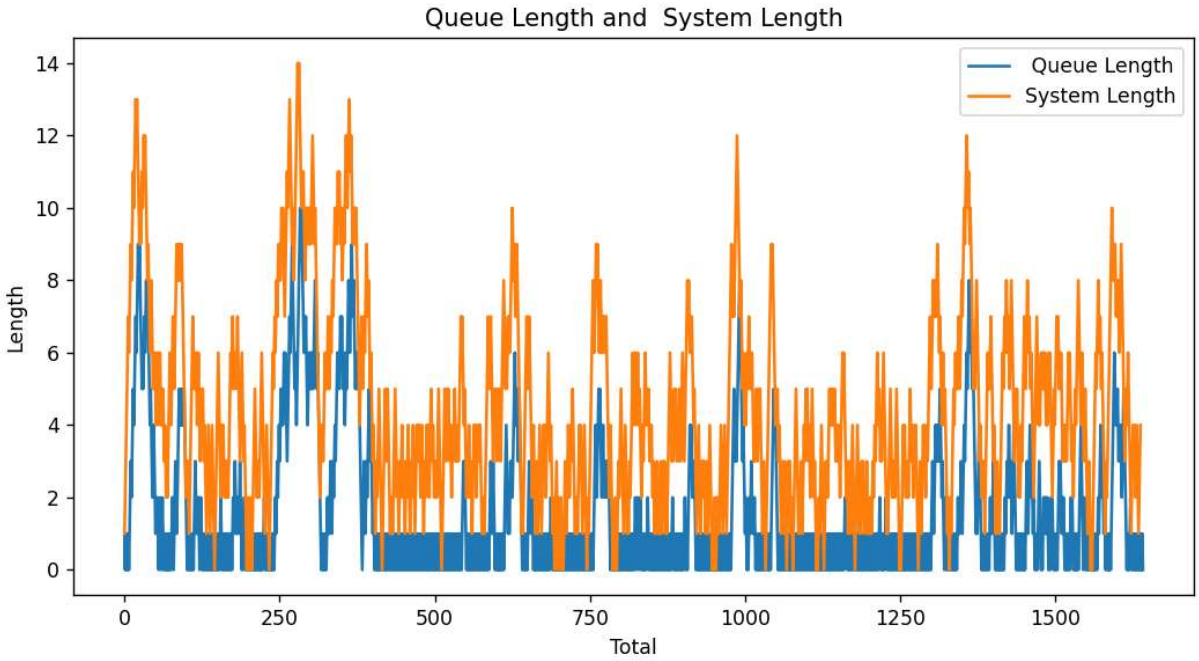

Рисунок 2.4 Прогресія черги та загальної кількості автомобілей у системі

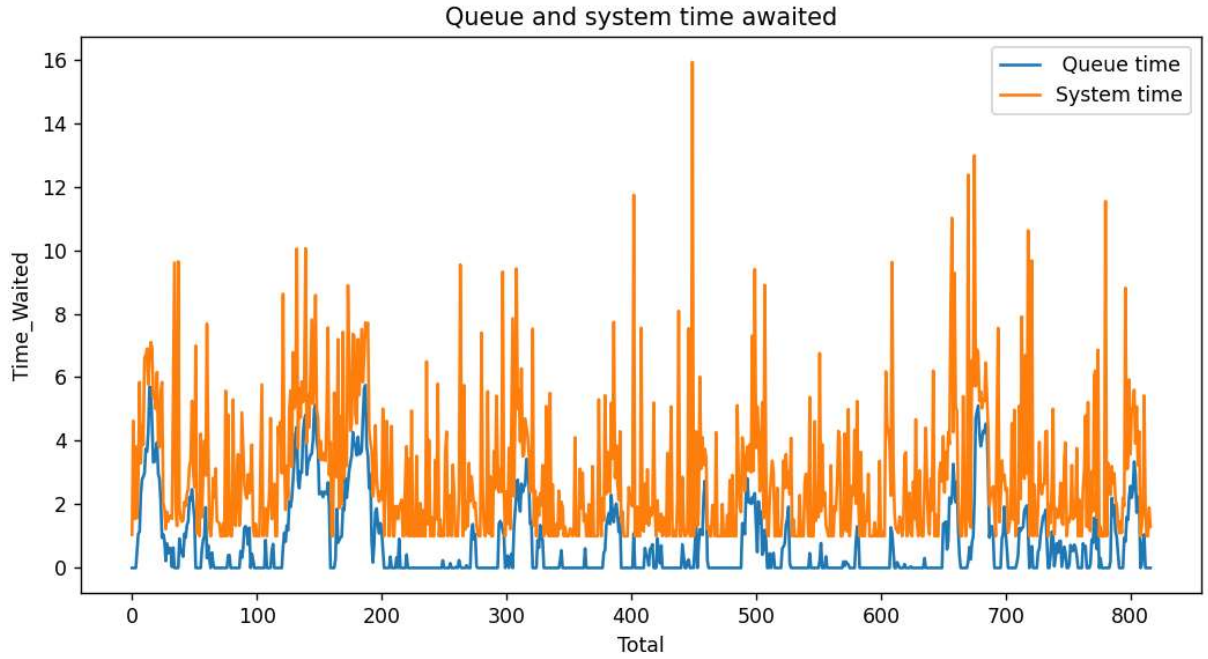

Рисунок 2.5 Прогресія часу очікування у черзі та часу проведеного у системі

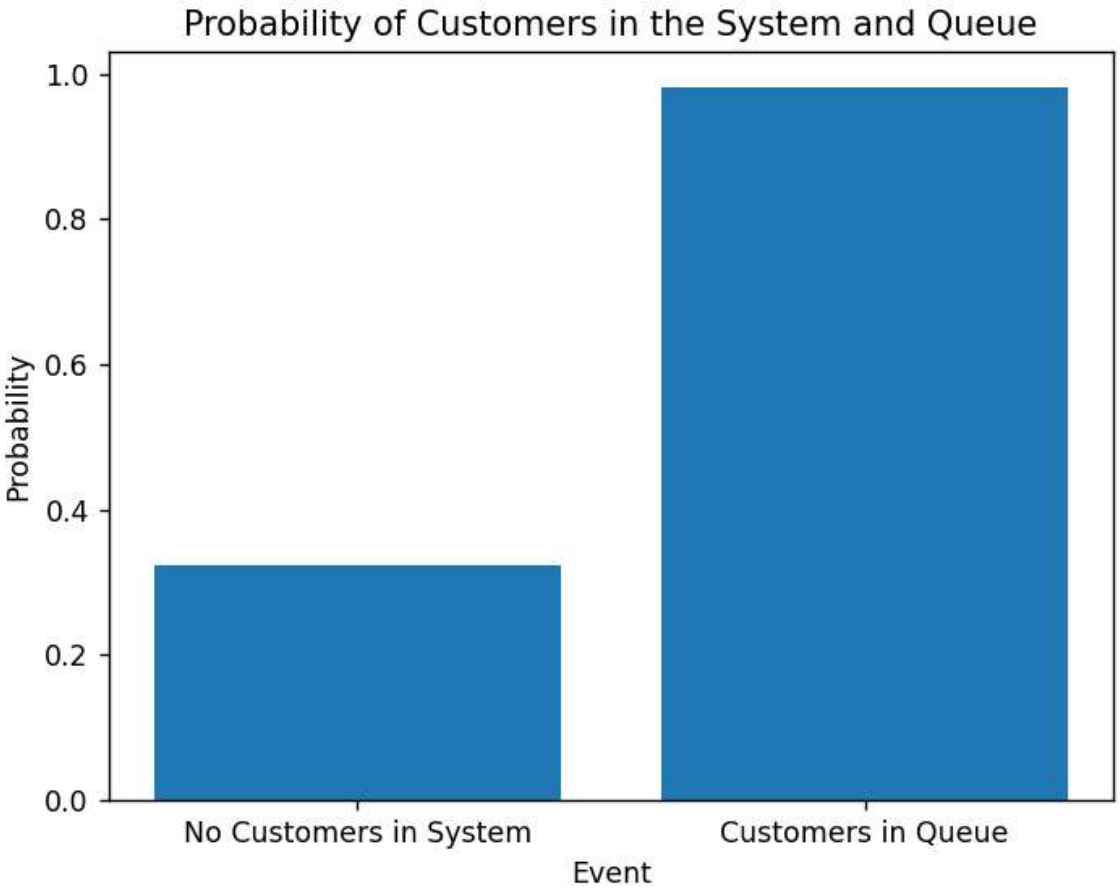

Рисунок 2.6 Ймовірності системи

Очевидно, що варіант із чотирма серверами буде завжди краще обслуговувати клієнтів.

2.4 Керування системою

Тепер розглянемо керування системою. Як вже було сказано, керується система за допомогою перевірки кількості клієнтів у черзі.

ρ	2.7
k	0.7714285714285715

$L_{\text{ч}}$	4.0212765957446805
L	6.968864468864469
$W_{\text{ч}}$	2.562979030135745
W	4.726377017869693
$P_{\text{оч}}$	0.8668693009118541
$P(0)$	0.010378510378510378

Таблиця 2.3 Параметри системи з адаптованими серверами

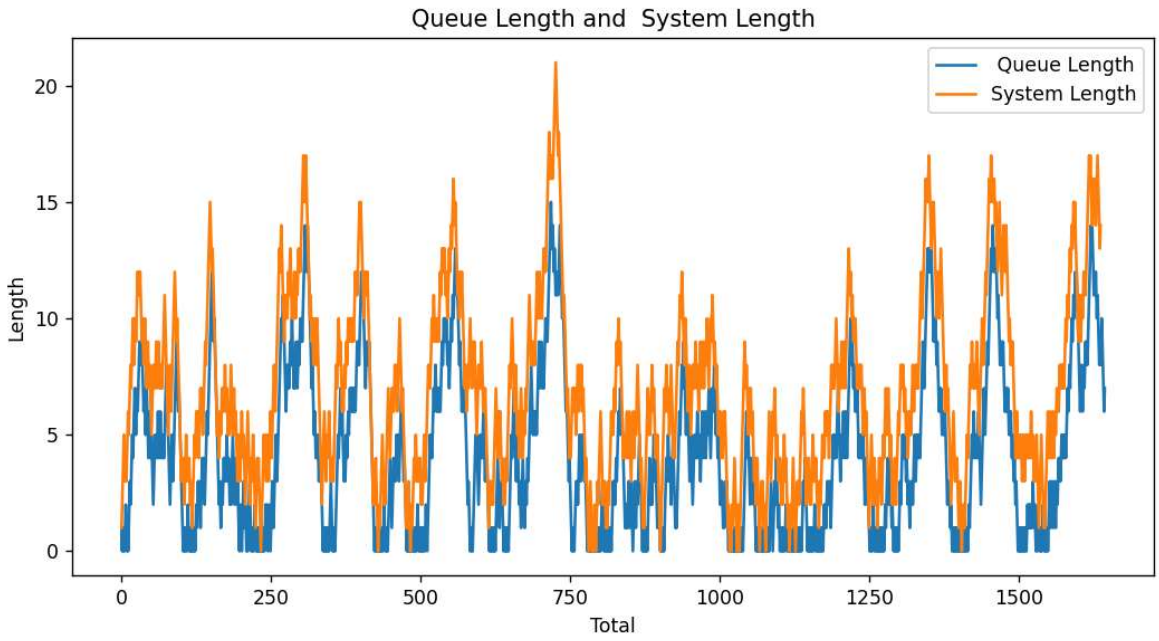

Рисунок 2.7 Прогресія черги та загальної кількості автомобілей у системі

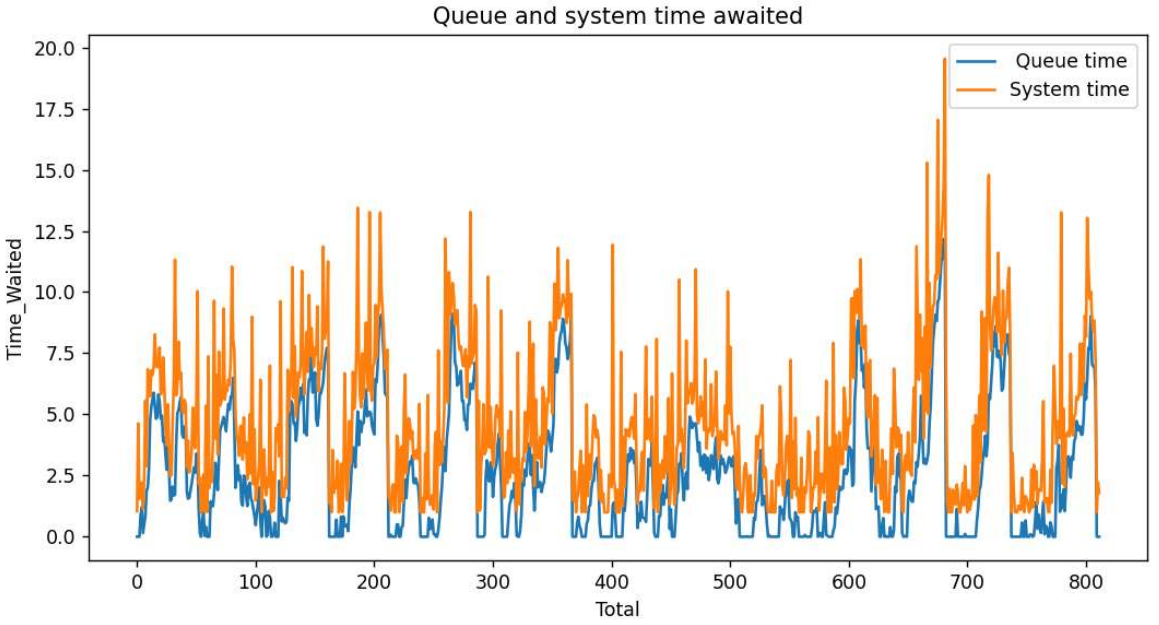

Рисунок 2.8 Прогресія часу очікування у черзі та часу проведеного у системі

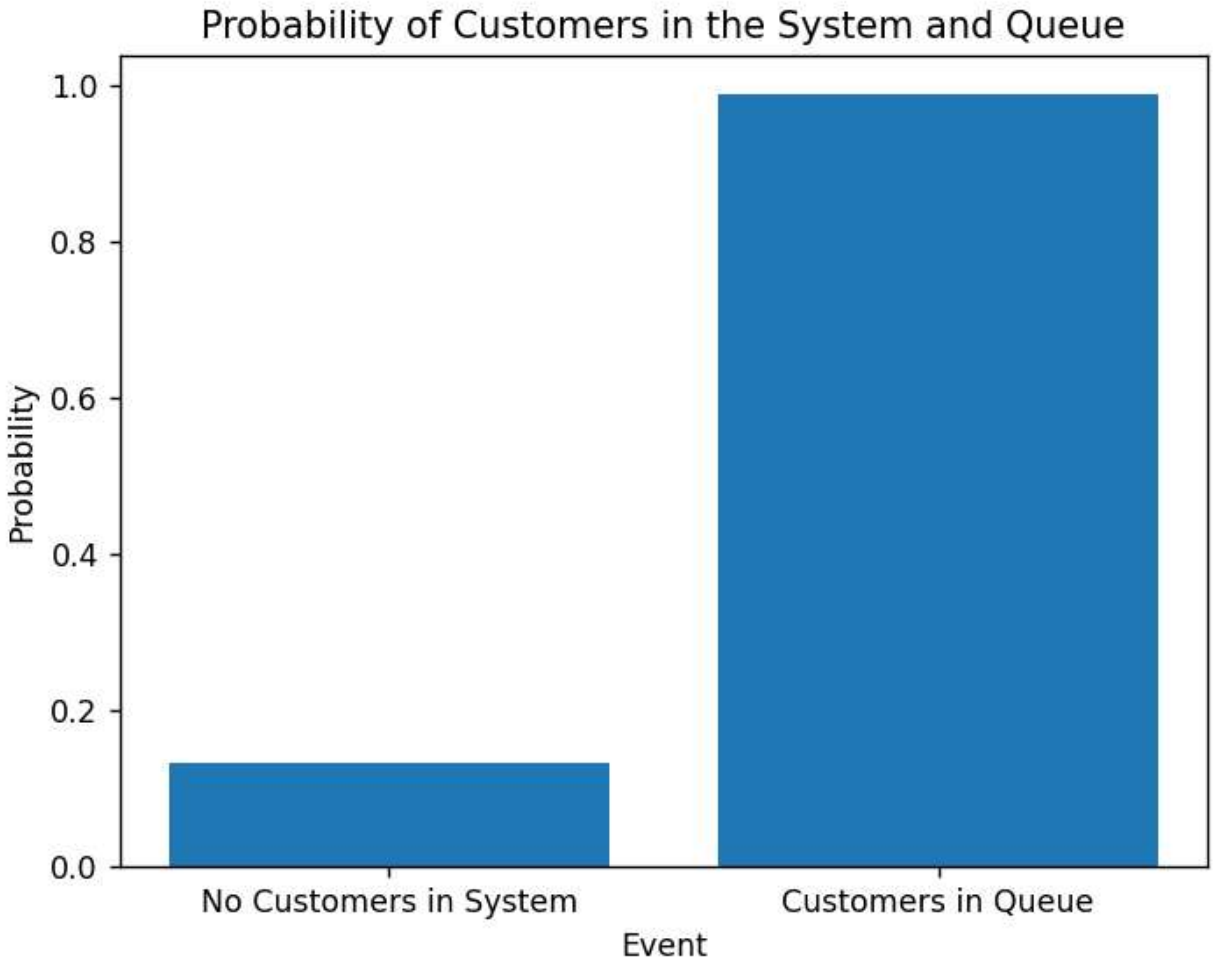

Рисунок 2.9 Ймовірності системи

У середньому, використовувалося 3.5 серверів. Кількість переключень $L_3 = 11$. Як видно з результатів, по кожному параметру керована система переважає своєю ефективістю систему з трьома серверами, але відстає від системи з чотирма серверами. Тож, варіант з чотирма серверами все ще краще.

На такому рівні це так. Але при використанні керування, ставиться якась задача, і у даному випадку ця задача полягає у оптимізації системи. Тож, перейдемо до задачі оптимізації цієї системи.

2.5 Перевірка оптимальності

Враховуючи те, що є три варіанта існування цієї системи, знайдемо значення функції якості для кожного з них.

За формулою

$$L(H) = C_1 L_1(H) - C_2 L_2(H) - C_3 L_3(H)$$

отримуємо

$$L(H_1) = 87468$$
$$L(H_2) = 88236$$
$$L(H_3) = 92306$$

Час від часу інтенсивність може змінюватися. Тож, для тесту змінимо параметри інтенсивності входу та виходу. $\mu,\lambda = \frac{1}{1.5}, 2$.

Тоді:

$$L(H_1) = 115551$$
$$L(H_2) = 124956$$
$$L(H_3) = 137631$$

Як можна побачити, при різних параметрах результат залишається одним. Параметри, такі як середній час у черзі, середня кількість у черзі будуть також зберігатися однаково відносними, хоча варто зауважити, що для першої моделі системи значення теоретичної навантаженності серверу буде дорівнювати 1, тож при цих параметрах протипоказано використовувати її.

Тож, стратегія, де кількість серверів адаптивно змінюється, є найбільш оптимальною серед трьох. Хоча у стратегії чотирьох серверів швидкість обслуговування буде швидше, прибуток від бізнесу буде більшим саме при H_3.

Дуже важливо зауважити те, що у інших умовах та з іншими параметрами може бути навпаки, де стратегія з перемиканнями буде надто великою морокою.

Тож, поки зовнішні умови саме такі, це оптимальна стратегія

ВИСНОВКИ

У роботі розглянута система масового обслуговування, проаналізовані її параметри. Були обговорені способи аналізу та напрямки оптимізації системи. Побудоване керування, завдяки якому вдалося оптимізувати функцію якості. Система була стохастичною, тому довелося враховувати це при побудові керування. Ми впевнилися у тому, що методів адаптивного керування вдосталь вистачає, щоб використати їх для побудови керування до системи.

В результаті аналізу роботи системи можна було зробити висновки про її ефективність та продуктивність. Наприклад, на основі статистики часу очікування клієнтів та завантаження серверів можна визначити оптимальну кількість серверів для досягнення максимальної продуктивності. Також можна розглянути варіанти впровадження розподіленої системи обслуговування.

В даному випадку було розглянуто систему автомийки, яка працює в умовах постійних перепадів електропостачання. Для оптимізації роботи системи було запропоновано використовувати адаптивне керування, яке регулює кількість серверів в залежності від спостережуваної довжини черги та середнього часу очікування, що і було використано. Також було наголошено на тому, що кожну окрему систему можна розглядати з різних боків у контексті керування та оптимізації.

Додатковим побажанням для бізнесу може бути мінімізація нових витрат, які можуть з’явитися. Це може включати зниження витрат на активні сервери, ефективне використання ресурсів та мінімізацію втрат від перемикань станів системи. Аналіз функції якості може врахувати цей аспект та підібрати оптимальну стратегію, яка забезпечує найнижчі витрати.

Іншим побажанням може бути забезпечення надійності та якості обслуговування. Це може включати мінімізацію часу перебоїв, забезпечення швидкого реагування на зміни умов. Функція якості може враховувати ці критерії та допомогти вибрати оптимальну стратегію, яка забезпечує найвищу надійність та якість обслуговування.

Також, якщо є потреба, можна побудувати інше керування цією системою, або взагалі її реорганізувати. Цілком можливий варіант цієї системи у вигляді системи M/M/N/K, де K це обмежена черга.

Важливо також подбати про вартістні коефіцієнти. Якщо ситуація в Україні стабілізується, збитки від використання серверів можуть набагато зменшитися.

Якщо є впевненість у тому, що бізнес успішний, то варто подумати про розширення. Збільшити кількість серверів, у випадку, якщо є потреба у послугах, що надає бізнес, тобто при постійному зростанні інтенсивності надходжень клієнтів.

Робота являє собою цінну інформацію, оскільки системи керування існують довкола нас, зустрічаються у кожному аспекті нашого життя. Особливо використовується принцип готовності до випадковостей, адаптація до навколишніх умов.

Тож, оскільки адаптивно керувати можна як процессорами, автомийкою, тема варта вивчення навіть для випадкового користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Математичне та імітаційне моделювання СМО із застосуванням середовища Matlab[Електронний ресурс] – Режим доступу до ресурсу: Посилання на ресурс

2. Прищепа О.В. Керовані системи з обмеженим числом повторів : дис. … канд. фіз.-мат. наук : 01.05.04 / О.В. Прищепа ; наук. кер. Є.О. Лебєдєв. – Київ, 2017.

3. Garikai B.W. An Empirical Analysis of the Queuing Theory and Customer Satisfaction: Application in Small and Medium Enterprises A Case Study of Croc Foods Restaurant // SSRN Electronic Journal[Електронний ресурс]. – 2013. – DOI: 10.2139/ssrn.2348304– Режим доступу : https://www.researchgate.net/publication/272301753_An_Empirical_Analysis_of_the_Queuing_Theory_and_Customer_Satisfaction_Application_in_Small_and_Medium_Enterprises_A_Case_Study_of_Croc_Foods_Restaurant

4. Mathworks[Електронний ресурс] – Режим доступу: Model Reference Adaptive Control - MATLAB & Simulink (mathworks.com)

5. Dumont G.A., Huzmezan M. Concepts, Methods and Techniques in Adaptive Control // Department of Electrical and Computer Engineering, University of British Columbia.

6. Вікіпедія. M/M/1 queue [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/M/M/1_queue.

7. Вікіпедія. Adaptive control [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Adaptive_control.

8. Теорія систем керування: підручник / В.І. Корнієнко, О.Ю. Гусєв, О.В. Герасіна, В.П. Щокін; М-во освіти і науки України, Нац. гірн. ун-т. – Дніпро: НГУ, 2017. – 497 с.

ДОДАТОК

Лістинг програми

```
import simpy
import random
import matplotlib.pyplot as plt

RANDOM_SEED = 123
NUM_SERVERS = 4
ARRIVAL_RATE = 2
SERVICE_RATE = 1 / 1.5
SIM_TIME = 600
adapt = 0
car_par = []
que = []
que_sys = []
que2 = [0]
switches = [0]
numserv = [NUM_SERVERS]


class CarWash:
    def __init__(self, env, num_servers, service_rate):
        self.env = env
        self.server = simpy.Resource(env, num_servers)
        self.service_rate = service_rate

    def wash(self, car):
        yield
self.env.timeout(max(random.expovariate(self.service_rate), 1))

    def change_servers(self, num_servers):
        self.server = simpy.Resource(self.env, num_servers)


def car(env, name, num, cw):
    car_par.append([])
    print(f'{name} arrives at the car wash at {env.now}')
    car_par[num - 1].append(env.now)
    if num == 1:
        que.append(1)
        que_sys.append(1)
        que2[0] = 1
    else:
        que.append(que[-1] + 1)
        que_sys.append(que_sys[-1] + 1)
        que2[0] += 1
    arrival_time = env.now
    with cw.server.request() as request:
        yield request
        print(f'{name} enters the car wash at {env.now}')
        que.append(que[-1] - 1)
```

```
        que2[0] -= 1
        car_par[num - 1].append(env.now)
        yield env.process(cw.wash(name))
        print(f'{name} leaves the car wash at {env.now}')
        car_par[num - 1].append(env.now)
        print(f'{name} spent {env.now - arrival_time} minutes in the
system')
        car_par[num - 1].append((car_par[num - 1][1] - car_par[num -
1][0]))
        car_par[num - 1].append((car_par[num - 1][2] - car_par[num -
1][0]))
        que_sys.append(que_sys[-1] - 1)


def adaptive_control(env, carwash, num_servers, threshold,
is_adapt):
    if is_adapt == 1:
        ccount = 0
        ccount2 = 0
        while True:
            yield env.timeout(1)


            queue_length = que2[0]

            if queue_length >= threshold and num_servers != 4:
                ccount += 1
                ccount2 = 0
                if ccount == 3:
                    ccount = 0
                    num_servers = 4
                    switches[0] += 1
                    numserv.append(4)

                    carwash.change_servers(num_servers)
            elif queue_length < threshold:
                ccount = 0
            if queue_length < threshold and num_servers == 4:
                ccount = 0
                ccount2 += 1
                if ccount2 == 3:
                    ccount2 = 0
                    num_servers = 3

                    switches[0] += 1
                    numserv.append(3)
                    carwash.change_servers(num_servers)
            elif queue_length > threshold:
                ccount2 = 0
            else:
                numserv.append(NUM_SERVERS)
```

```
def setup(env, car_wash, num_servers, service_rate, arrival_rate):
    i = 0

    while True:
        yield env.timeout(random.expovariate(arrival_rate))
        i += 1
        env.process(car(env, f'Car {i}', i, car_wash))


random.seed(RANDOM_SEED)
env = simpy.Environment()

car_wash = CarWash(env, NUM_SERVERS, SERVICE_RATE)
env.process(setup(env, car_wash, NUM_SERVERS, SERVICE_RATE,
ARRIVAL_RATE))
env.process(adaptive_control(env, car_wash, NUM_SERVERS,
threshold=10, is_adapt=adapt))
env.run(until=SIM_TIME)

print("Average queue length, system length:")
print(sum(que) / len(que), sum(que_sys) / len(que_sys))
print("Average customer waiting time in queue, system:")
count = 0
sum1 = 0
sum2 = 0
wait_que = []
wait_sys = []
for i in car_par:
    if len(i) < 5:
        continue
    else:
        sum1 += i[3]
        sum2 += i[4]
        count += 1
        wait_que.append(i[3])
        wait_sys.append(i[4])
print(sum1 / count, sum2 / count)


c = que_sys.count(0)
c1 = len(que) - que.count(0)

print("Probability of no car in the system:")
print(c / len(que_sys))
print("Probability of car in the queue:")
print(c1 / len(que))

numserv_final = sum(numserv) / len(numserv)
print(numserv_final)
print(count)

print(switches)
```

```
print("Table of selected car parameters:")
print("Car\tArrival Time\tEnter Time\tExit Time\tTime Spent in
Queue")
for i in range(8):
    car_info = car_par[i]
    arrival_time = car_info[0]
    enter_time = car_info[1]
    exit_time = car_info[2]
    system_time = car_info[3]
    print(f"{i +
1}\t\t{arrival_time}\t\t\t{enter_time}\t\t\t{exit_time}\t\t{system_t
ime}")


que_prob = []
que_sys_prob = []
print(f"{sum(que) / len(que)}\t\t\t{sum(que_sys) /
len(que_sys)}\t\t\t{c / len(que_sys)}\t\t{c1 / len(que)}")

if adapt == 1:
    print(ARRIVAL_RATE / SERVICE_RATE, (ARRIVAL_RATE / SERVICE_RATE)
/ numserv_final)
    print(120 * count - sum(numserv) * 3 - 2 * switches[0])
    print(sum(numserv))
else:
    print(ARRIVAL_RATE / SERVICE_RATE, (ARRIVAL_RATE / SERVICE_RATE)
/ NUM_SERVERS)
    print(120 * count - sum(numserv) * 3 * count - 2 * switches[0])
values = [
    120 * count - 3 * count * 5 - 2 * 0,
    120 * count - 4 * count * 5 - 2 * 0,
    120 * count - sum(numserv) * 3 - 2 * switches[0]
]

num_servers = [3, 4, numserv_final]

plt.bar(num_servers, values, width=0.01)
plt.xlabel('Number of Servers')
plt.ylabel('Function Value')
plt.title('Quality Function Values')
plt.show()

time = list(range(len(que)))
time_sys = list(range(len(que_sys)))

plt.figure(figsize=(10, 5))
plt.plot(time, que, label=' Queue Length')
plt.plot(time_sys, que_sys, label='System Length')
plt.xlabel('Total')
plt.ylabel('Length')
plt.title('Queue Length and  System Length')
plt.legend()
```

```
plt.show()

time2 = list(range(len(wait_sys)))
time2_sys = list(range(len(wait_sys)))
plt.figure(figsize=(10, 5))
plt.plot(time2, wait_que, label=' Queue time')
plt.plot(time2_sys, wait_sys, label='System time')
plt.xlabel('Total')
plt.ylabel('Time_Waited')
plt.title('Queue and system time awaited')
plt.legend()
plt.show()


# Calculate probabilities
prob_any_customer_in_sys = (len(que_sys) - que_sys.count(0)) /
len(que_sys)
prob_no_customers_in_queue = que.count(0) / len(que)

# Create histogram data
data = [prob_no_customers_in_queue, prob_any_customer_in_sys]
labels = ['No Customers in System', 'Customers in Queue']

# Plot the histogram
plt.bar(labels, data)
plt.xlabel('Event')
plt.ylabel('Probability')
plt.title('Probability of Customers in the System and Queue')
plt.show()
```