Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи пошуку вузьких місць у комп’ютерних мережах

(тема)

Виконав:

студент II курсу, групи КСМм-22-2

Лук'янов О.О.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: проф. Горбачов В.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Лук'янову Олександру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи пошуку вузьких місць у комп'ютерних мережах

затверджена наказом по університету від " 06 " листопада 2023 р. № 1298 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024 р.

3. Вхідні дані до роботи 1) Системи з мережевою структурою; 2) Аналіз підходів до пошуку вузьких місць у мережах; 3)Метод балансу потоків; 4) Інструменти Java для імітаційного моделювання

4. Перелік питань, що потрібно опрацювати у роботі

1) огляд проблеми та постановка дослідження;
2) аналіз проблем вузьких місць у складних системах;
3) аналіз методів пошуку вузьких місць у складній мережевій системі;
4) розробка та впровадження сценаріїв пошуку;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	07.11.23-13.11.23	
2	Формування переліку вимог до програми	14.11.23-20.11.23	
3	Вибір технології розробки та	21.11.23-23.11.23	
	інструментальних засобів		
4	Розробка	24.11.23-06.12.23	
5	Тестування програми	07.12.23-23.12.23	
6	Оформлення матеріалів кваліфікаційної роботи	26.12.24-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Горбачов В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 89 с., 17 рис., 2 табл., 2 дод., 11 джерел.

ВУЗЬКЕ МІСЦЕ, СИСТЕМИ МАСОВОГО ОБСЛУГОВУВАННЯ, РІВНЯННЯ БАЛАНСУ ПОТОКІВ, JAVA.

Метою кваліфікаційної роботи є провести аналіз інформації про вузькі місця в мережі та Інтернеті, а також причини та методи пошуку вузьких місць які базуються на теорії систем масового обслуговування

У ході виконання кваліфікаційної роботи було проведено огляд проблеми та постановка дослідження, аналіз проблем вузьких місць у складних системах, аналіз методів пошуку вузьких місць у складній мережевій системі, розробка та впровадження на мові Java сценаріїв пошуку.

ABSTRACT

Master’s thesis: 89 pages, 17 figures, 2 tables, 2 appendices, 11 sources.

BOTTLENECK, QUEUING SYSTEMS, FLOW BALANCE EQUATION, JAVA.

The major goal of this thesis is to analyze information about bottlenecks in the network and the Internet, as well as the causes and methods of finding bottlenecks based on the theory of queuing systems

In the course of the qualification work, an overview of the problem and formulation of the research was carried out, analysis of problems of bottlenecks in complex systems, analysis of methods of searching for bottlenecks in a complex network system, development and implementation of search scripts in the Java language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – інтерфейс прикладних програм (англ.,Application programming interface)

DA – область даних (англ.,Data area)

DPS – системи розподіленої обробки (англ.,Distributed Processing System)

ISP – інтернет-провайдер (англ.,Internet Service Provider)

MAS – многоагентная система (англ.,Multi Agent System)

OS – операційна система (англ.,Operating System)

PE – показники продуктивності (англ.,Performance Indicators)

QoS – якість обслуговування (англ.,Guality of Service)

ВСТУП

Продуктивність системи виробництва, така як середня пропускна здатність, час відгуку та затримки, і т. д., залежить від потужності машин і ресурсів, доступних в системі. Деякі з них можуть впливати на продуктивність системи більше, ніж інші. Як правило, обмеження системи може бути приписане обмеженням одного або двох машин чи одного або двох типів ресурсів, широко відомих як вузькі місця.

З точки зору системи, вузькі місця є точками застою в системі, сповільнюючи весь ланцюг операцій. Для підвищення продуктивності системи необхідно усунути вузькі місця. Однак визначення та усунення вузьких місць не є тривіальним завданням. В дизайні та дослідженнях ми не можемо напряму просити експертів про співпрацю, як це було запропоновано Cox та Spencer (Cox and Spencer, 1997). Нам потрібно працювати над великою кількістю округлених даних або даних логування для визначення вузьких місць і поліпшення вузьких місць. Ітерація для поліпшення вузьких місць показана на ілюстрації.

Вузькі місця визначаються відповідно до вимог застосунку. Вони розпізнаються різними методами виявлення. Асимптотичні та апроксимаційні методи на основі вузьких місць використовуються для оцінки продуктивності системи. Вузькі місця усуваються за допомогою регулювання параметрів системи.

Ця ітерація повторюється до тих пір, поки вимоги програми не будуть виконані. Багато факторів у системі сприяють утворенню вузьких місць, таких як пропускна здатність машини та кількість операторів. Вузьке місце системи може відрізнятися від різних перспектив, і це може відрізнятися для різних категорій клієнтів. З великими системами вона стає дуже складною. За останнє десятиліття було докладено багато зусиль і представлено різні визначення, методи виявлення, апроксимація та асимптотичні результати.

Але досі не існує загальноприйнятого визначення або методики скринінгу. Це в основному пов'язано з різноманіттям вузьких місць у різних сценаріях застосування. Важко передати теоретичні результати в реальні додатки. Повинно бути зрозуміло, яке визначення; метод виявлення і асимптотичний результат підходять для конкретного сценарію застосування.

У цій статті розглядаються основні причини того, що сприяє утворенню вузького місця і як вузькі місця в виробничих мережах можуть бути визначені, визнані і використані з точки зору додатків. Теорія черг служить математичною основою для нашого дослідження. Різні визначення поділяються на дві основні категорії: визначення на основі продуктивності (PIP) і визначення на основі чутливості.

Перший клас фокусується на продуктивності системи в режимі реального часу, а другий більше фокусується на поліпшеннях, які можуть бути зроблені. Методи виявлення перевіряються відповідно до визначень. Вони поділяються на метрично-орієнтовані методи та індикатори на основі чутливості. Наведені порівняння цих методів виявлення показують, що різні методи виявлення можуть призвести до дуже різних наслідків.

1 РОЗГЛЯД ПРОБЛЕМИ ТА ПОСТАНОВКА ЦІЛЕЙ ДОСЛІДЖЕННЯ

1.1 Актуальність проблеми

Вузькі місця обмежують проект. Кожен проект має одну або кілька вузьких місць, які можуть завадити нам завершити проект. Необхідний ресурс може бути отриманий, критичне завдання може бути не завершеним вчасно або є інша проблема.

Вузькі місця – це «руйнівники» проєктів – вузькі місця існують у кожному проекті і тільки чекають, щоб зіпсувати успіх нашого проєкту (рисунок 1.1).

Головна мета для будь-якого керівника проекту – це виявити і усунути вузькі місця – досвідчений керівник проекту розуміє важливість вузьких місць для успіху проекту. На основі цієї ідеї успішні керівники проектів ретельно працюють над виявленням вузьких місць і їх вирішенням на ранній стадії.

Усунення вузького місця може призвести до подальших вузьких місць – коли ми видаляємо вузьке місце, може виникнути одне або кілька нових вузьких місць, як правило, далі за графіком реалізації проєкту. Будьте в курсі цього факту, оскільки ми повинні «знайти і знищити» будь-які нові вузькі місця.

Рисунок 1.1 – Ілюстрація «вузького місця»

Прикладом цього є те, що при видаленні вузького місця, ми можемо видалити вузьке місце на виробничій конвеєрній лінії, думаючи, що коли вузьке місце буде усунуто, ми виготовимо більше «віджетів». Однак у деяких випадках усунення виявленого вузького місця може призвести до виникнення інших вузьких місць у процесі подальшої збірки і навіть зменшити випуск віджетів замість того, щоб збільшити його.

Усунення вузького місця може мати великі переваги – усунення вузького місця часто економить наш час в рамках нашого проекту і може фактично створити запас часу для інших завдань у нашому проекті. Своєчасне виконання проекту або, можливо, поліпшення графіка, дозволяє знизити вартість проекту і дозволити компанії швидше скористатися перевагами проекту.

Створюючи план проекту, ретельно плануйте завдання, щоб ми могли їх виконати вчасно. Знайти вузькі місця, які дозволять завершити проект раніше, ніж очікувалося, є величезним плюсом, коли це станеться. Вузькі місця визначають пропускну здатність складної системи. Усвідомлення цього факту і внесення поліпшень збільшить грошовий потік.

Вузьке місце в структурі мережі вказує на ресурс, який займає найбільше часу в операціях ланцюга поставок, щоб задовольнити даний попит. Як правило, часто спостерігаються таке явище, як збільшення обсягу запасів перед вузьким місцем і недостатня кількість компонентів після вузького місця. Оскільки коливання не є статистично постійними, явища (надлишок і нестача матеріалу) не завжди відбуваються. У пішому поході вузьке місце – це ланка, яка рухається найповільніше. Проміжок між вузьким елементом і елементом перед ним розширюється та звужується з елементом після нього.

У випадку вузьких місць важливо, щоб вузькі місця визначали пропускну здатність мережі. Якщо людина в вузькому місці може швидше ходити в поході, швидкість всієї групи збільшується. Аналогічно, пропускна здатність зростає, коли збільшується потужність вузького місця в ланцюгу

поставок. Згідно з визначенням вузьких місць, коефіцієнт використання не вузьких місць становить менше 100%. Якщо так, то коефіцієнт використання не вузьких місць збільшиться лише в межах 100% навіть при збільшенні потужності вузького місця і збільшенні пропускної здатності. Якщо використання невузьких місць перевищує 100%, це означає, що місце розташування вузьких місць переміщується до місця, де немає вузьких місць. Якщо вузькі місця недостатньо визначені, ми втрачаємо можливість збільшити пропускну здатність. Є багато випадків, коли енергія використовується для невеликого скорочення витрат і шанс на великий грошовий потік пропускається через відсутність дефіциту. Погодинна вартість при вузькому місці є еквівалентом втрати години для всього ланцюжка поставок, а також втрати пропускної здатності для всього ланцюжка поставок.

Теорія обмежень (TOC) пояснює, чому ідентифікація та управління вузькими місцями збільшує пропускну здатність ланцюга поставок, ефективно використовучи машини і значно збільшуючи прибуток. Якщо збільшення потужності вузького місця становить 0,1% від загальної вартості, решта 99,9% можуть бути витрачені на збільшення пропускної здатності без додаткових витрат.

Бувають такі випадки, коли весь час і енергія витрачається на скорочення витрат, і тому єдиним поліпшенням є зниження робочої швидкості вузького місця з 80% до 60% без поліпшення грошових потоків. Якщо виробнича частина вважає, що різниця у витратах через використання потужностей у стандартному обліку витрат не є її недоліком, то це тому, що вона думає лише про часткову оптимізацію.

1.2 Огляд літератури

Складні системи, такі як зв'язок і комп'ютерні мережі складаються з ряду взаємодіючих частинок (або клієнтів), які демонструють значні явища

перевантаження зі збільшенням ступеня взаємодії. Динаміка таких систем визначаються випадковістю подій, що лежать в їх основі, наприклад, надходження робочих одиниць, і можуть бути описані стохастично як моделі мереж масового обслуговування. Якщо вони зрозумілі, вони дозволяють передбачати досяжну продуктивність системи, оптимізацію конфігурації мережі та здійснення досліджень з планування потенціалу.

Проблема, що аналізується в цій статті, зосереджена на аналізі виробничого процесу з метою виявлення вузьких місць і аналізу стану процесу. Ця проблема дуже важлива з точки зору виробничої організації, адже вузькі місця дуже небезпечні з точки зору часу виробництва. Ми повинні визначити їх, а потім аналізувати весь процес з урахуванням часу безперебійної роботи. Наступним кроком може бути скорочення цього періоду за допомогою методів ощадливого управління.

Згідно з Кембриджським словником англійської мови, ми можемо визначити термін вузьке місце як проблему, яка затримує прогрес. Іншими визначеннями цього терміна є, наприклад, наступне:

- сервіс, система, машина або ресурс, який вже повністю задіяний і тому не може впоратися з додатковим навантаженням. Вузьке місце, також відоме як критичний ресурс, обмежує потік пов'язаних ресурсів;
- вузьке місце у виробництві – це точка, в якій операція досягає або перевищує потужність заводу. Іншими словами, завод або відділ не можуть виробляти достатньо одиниць досить швидко, щоб зберегти решту графіка виробництва або іншої повсякденної рутини на тому ж рівні.

Відповідно до теорії обмежень (TOC), головним завданням ефективного управління виробництвом є пошук і усунення наслідків вузького місця в компанії. З точки зору виникнення вузького місця можна розділити управління виробництвом на три типи:

- управління в ситуації без вузького місця в процесі виробництва;
- управління в умовах вузького місця в процесі виробництва;

- управління в ситуації з безліччю вузьких місць у виробничому процесі.

Відсутність вузьких місць у виробничому процесі означає, що продуктивність виробництва достатня для досягнення очікуваного обороту. Якщо виникає вузьке місце, повинно бути визначено, чи є тільки один виробничий процес або можливі альтернативні виробничі процеси. Корисним інструментом є аналіз маржинального доходу для окремих комбінацій продуктів і процесів. Якщо виявляється, що в процесі виробництва існує кілька паралельних вузьких місць, то рішення стає більш складним. У такому випадку, рішення цих проблем слід знаходити за допомогою методів лінійного програмування.

В подальшому термін «вузькі місця» в контексті цих інструкцій обмежується поняттям «повторювані», на відміну від «одноразових» вузьких місць у трафіку. Повторювані вузькі місця є передбачуваними з точки зору причини, місця, часу доби та приблизної тривалості; наприклад, ті, з якими ми зустрічаємося в наших щоденних поїздках. Точкові вузькі місця є випадковими в залежності від місця розташування і серйозності (в переносному сенсі). Прикладами можуть бути нещасні випадки, погодні явища і навіть «заплановані» події, такі як робочі зони і спеціальні заходи, які є нерегулярними як за часом виникнення, так і за місцем розташування.

«Локалізована» повторюване вузьке місце може розглядатися як певна подія у визначеному місці; В. вниз по смузі, плетіння, перехрестя, або в'їзд чи виїзд. Наприклад, повторювані затори під час руху на перехресті протягом декількох годин на день будуть «локальними», тоді як «мега» вузьке місце або системний затор буде сприйматися як перехрестя з недостатнім розміром.

Повторювані і локалізовані вузькі місця завжди виникають, коли швидкість вхідного трафіку перевищує швидкість вихідного трафіку.

Причинний ефект, як правило, можна пояснити наявністю принаймні одного з наступних двох факторів:

- точки прийняття рішень, такі як в'їзні та виїзні пандуси, зони злиття, зони переплетення, з'їзди зі смуги руху, платні зони та світлофори;

- фізичні обмеження, такі як повороти, підземні переходи, тісні конструкції або відсутність узбіч.

В останні десятиліття досліджено різні підходи до оцінки стаціонарної поведінки замкнутих мереж черг товарних форм з великою кількістю клієнтів. Велика частина літератури ставить за мету розробити чіткі алгоритми для ефективного обчислення постійних або стаціонарних показників ефективності нормалізації, таких як бітові швидкості і довжини черги. Наскільки нам відомо, жоден точний алгоритм не має поліноміального часу виконання щодо кількості маршрутів, клієнтів та черг. Мотивовані цією трудністю, ряд альтернативних методів аналізу стаціонарної поведінки з'явилися в літературі.

В основному вони складаються з:

- використання аналізу середньої величини (MBA) відкриті Рейзером і Лавенбергом для розробки ітераційних алгоритмів або алгоритмів з фіксованою точкою. Хоча ці методи покращують час роботи MVA, немає ніякої гарантії, що вони будуть сходитися до точного рішення, за винятком особливих випадків;

- розробка ефективних методів ідентифікації вузьких місць: Цей підхід спрямований на зменшення розміру мережі шляхом ігнорування впливу станцій, які мало впливають на загальну продуктивність;

- дослідити поведінку мережі в граничному режимі: У нашому контексті закритої черги ми можемо припустити, що в мережі існує черга M / M /, і загальна кількість робіт, скажімо, n, пропорційно збільшує час її роботи. Іншою можливістю є дозволити n зростати нескінченно по відношенню до кількості станцій. Нарешті, ми також можемо дослідити поведінку мережі, коли n зростає до нескінченності, фіксуючи пропорції клієнтів на кожному маршруті.

Оскільки аналіз корисний для стількох кроків у процесі прийняття рішень, то існує безліч методів. Важливо відзначити, що методи широко варіюються; жоден інструмент не може моделювати всі сценарії або пропоновані поліпшення. Тому важливо підібрати підходящий інструмент відповідно до цілей і завдань проекту.

У деяких випадках найбільш економічно ефективний спосіб вирішити вузькі місця – це зробити найпростіші геометричні або операційні поліпшення. Багато з цих рішень можуть бути виконані як проекти забезпечення безпеки, проекти, які не звільняються від федеральних грантів, або навіть проекти робіт з технічного обслуговування. При правильному застосуванні ці стратегії можуть призвести до відмінного співвідношення ціни та якості завдяки меншому розміру, меншій вартості дизайнерського рішення та меншій вартості життєвого циклу, включаючи планування, проектування, розробку, експлуатацію та обслуговування.

Найбільш поширеними стратегіями пом'якшення наслідків є:

- повторна синхронізація сигналу: На багатьох перевантажених смугах можна зменшити вузькі місця простою оптимізацією часу роботи світлофорів або зміщенням їхнього часу між перехрестями;

- перезавантаження: Розмітка смуг для додавання допоміжних смуг або смуг прискорення/сповільнення може збільшити пропускну здатність або ефективніше перенаправляти транспортні потоки. Деякі з найпоширеніших методів усунення включають запобігання переплетенням або крутим поворотам, які спричиняють уповільнення; Решта смуг руху з метою забезпечення більшої кількості смуг руху, хоча і трохи вужчих; або перетворення коротких узбіч на смуги руху;

- знаки та сигнали: Знаки та сигнали можуть бути розроблені таким чином, щоб обмежувати певні види руху на користь інших (наприклад, знак СТОП, знак ДОСТАВКА на невеликих проходах) або забороняти неефективні види руху (наприклад, розворот на півдорозі або поворот ліворуч при перетині інтенсивного зустрічного руху). З іншого боку, ці

стратегії також можуть бути використані для визначення пріоритетності інтенсивного руху, щоб уникнути утворення "вузьких місць" (наприклад, поворот направо на червоне світло і тільки сигнали повороту наліво);

- встановлення датчиків руху по кільцю: Встановлення датчиків руху перед світлофорами може допомогти зменшити черги, динамічно визначаючи пріоритетність найбільш завантажених під'їздів за потреби.

Демонтаж пандусів або під'їзних шляхів (або зміна): Закриття, переміщення, зміна розмірів або об'єднання пандусів, особливо пандусів з низькою інтенсивністю руху, може вивільнити деякі транспортні шляхи. При зміні пандусів тимчасові блокування можуть перевірити гіпотезу. Пандус завжди можна відкрити знову, якщо ефект загоєння гірший, ніж причина проблеми.

В інших випадках можуть знадобитися більш дорогі рішення, в тому числі перепланування або перебудова області поблизу розташування вузького місця. Ці стратегії можуть також виробляти значне співвідношення витрат і користі, але витрати завжди будуть вище, ніж стратегії, перераховані вище. Ці стратегії можуть мати високі фінансові витрати життєвого циклу (планування, проектування, будівництво, експлуатація та технічне обслуговування) або соціальні витрати (наприклад, незалежно від характеру витрат, ці поліпшення повинні бути ретельно сплановані до початку її реалізації. Реалізуються, тому що вони дорогі та складні у втіленні.

Розглянемо випадок, коли можна усунути лише природні вузькі місця. Згідно з Drr> Dir, i! = R це означає, що навантаження ділянки задовольняють умову

$$Drr > Dir > Djr,\ i != r,\ j > R. \tag{1.1}$$

Зрозуміло, що в цьому випадку у нас є

$$\Phi = \Phi' = \{1, 2, \ldots, R\}, \tag{1.2}$$

це означає, що лінійна система складається з R – 1 нерівностей. Оскільки Bm, m, не є порожніми, а вирази представляють використання вузлів, ми можемо отримати комбінації, які наситять всі природні вузькі місця, накладаючи обмеження рівності. Для отримання R-вершин GSS достатньо накласти на них обмеження рівності

$$\sum_r \frac{\beta r}{Dmr} Dir = 1, \forall i \in \Phi, \mathrm{i} \neq \mathrm{m} \quad , \tag{1.3}$$

$$\sum_r \beta r = 1 \quad , \tag{1.4}$$

для кожного м ∈ Ф. Нехай βm, Ф – є рішенням системи. Якщо βm, Ф є допустимим рішенням, тобто він виконує граничні умови, то він являє собою точку перемикання для поведінки всіх вузлів і ∈ Ф, i! = M, тобто точка, в якій станції, що належать Ф \ {m} змінюють свій статус вузького місця / не вузького місця, і це точка, в якій всі вузькі місця природні повинні бути поєднані разом. Зауважте, що βm, Ф можуть мати від'ємні значення, і в цьому випадку розв'язок не є можливим.

Це означає, що в таких випадках GSS не потрібна. З іншого боку, ESG виникає, якщо βm, Ф є можливим рішенням. Надалі ми будемо називати глобальну точку перемикання m для того, щоб отримати робочі розв'язки лінійної системи. GSS задається політопом (багатокутников), який має всі глобальні точки перемикання як кути, і його унікальність випливає з унікальності βm, Ф. Безперервність використання класів по відношенню до нас змушує нас зробити наступне припущення:

$$\lim_{N\to\infty} Uir(N) = \lim_{\beta\to\beta-} \beta r \frac{Dir}{Dmr} = \beta m, \Phi, \mathrm{r} \frac{Dir}{Dmr} . \tag{1.5}$$

Після усунення проблеми межі пропускної здатності системи зростуть, але це не означає, що вони стануть нескінченними, оскільки в більшості випадків інший ресурс стане новою причиною додаткових обмежень. Слід зазначити, що правильна реалізація різних етапів циклу ТОС перетворює існуючі обмеження на можливість для комерційного розвитку. Розуміння обмежень вашої системи – ключ до успіху будь-якого бізнесу. У цьому випадку ми повинні поставити запитання. Що обмежує створення виняткової цінності для наших клієнтів на стратегічному рівні? І що обмежує збільшення потужності потоку на операційному рівні? У повсякденному житті термін "обмеження" має негативні асоціації. Однак теорія обмежень пропонує іншу перспективу. Обмеження системи насправді є важелем всієї системи. Після того, як цілі системи чітко визначені, важливо зрозуміти її обмеження, щоб створити ігрову дошку – стратегії і тактики, які допоможуть нам досягти мети.

1.3 Формулювання цілей дослідження

Основною метою дослідження цього проекту є розробка системного підходу для поліпшення аналізу вузьких місць, який може допомогти будь-якому бізнесу визначити, вивчити, моделювати та пом'якшити вузькі місця на системному рівні, а не зосереджуватися лише на локальних вузькі місця. Це дозволить систематично, ефективно та оперативно ідентифікувати, управляти і зменшувати затори по всій країні. Для досягнення цієї мети, це дослідження переслідує такі цілі:

- визначити вузькі місця в комп'ютерних мережах і показати загальні причини і результати;
- огляд і синтез попереднього досвіду в аналізі вузьких місць на автомагістралях; особливу увагу буде приділено існуючим застосуванням інструментів динамічного мезоскопічного розподілу трафіку (DTA) для аналізу вузьких місць;

- визначення комплексу заходів з ефективності (MOE) для оцінки існуючої мережі автомагістралей, визначення найбільш серйозних вузьких місць в мережі і кількісної оцінки впливу стратегій кандидатів на зменшення вузьких місць системи і умов проїзду;

- розробка методології визначення та пріоритизації вузьких місць на автомагістралях.

1.4 Організація звіту

Цей звіт складається з наступних розділів:

- вступ: забезпечує передумови необхідності системного підходу в аналізі вузьких місць і окреслює обсяг і цілі проекту;

- розділ 1 – визначення показників ефективності: представляє попередній досвід вибору MOEs для аналізу вузьких місць і синтезує принципи, що використовуються для вибору MOEs в цьому дослідженні;

- розділ 2 – огляд літератури: підсумовує поточну практику методів ідентифікації вузьких місць, заходів пом'якшення та інструментів оцінки;

- розділ 3 – розробка структури для оцінювання проектів потенційного вдосконалення: розробляє систему та використовує її для розробки процедури класифікації проектів на основі показників ефективності для оцінювання та ранжування альтернативних варіантів усунення вузьких місць;

- розділ 4 – висновки та узагальнення: підсумовує дослідження і робить висновки щодо досягнень в цьому дослідженні.

Нарешті, в кінці звіту та додатку надається перелік посилань.

2 АНАЛІЗ ВУЗЬКИХ ЗАДАЧ У СКЛАДНИХ СИСТЕМАХ

2.1 Основні поняття та визначення

Словникове визначення вузького місця – це щось (як машина), «що вже працює на повну потужність і тому не може впоратися з додатковими вимогами, які на неї покладаються».

Поширеною аналогією є рух на великій автостраді, яка веде до вузької вулиці. Навіть коли вузька вулиця повністю використовується, вона не може прийняти більше, ніж вміщує. На початку вузької вулиці, звичайно, утворюється черга, яка перешкоджає ширшій вулиці. Вузька вулиця стає вузьким місцем.

Таблиця 2.1 – Узагальнення існуючих визначень вузьких місць [11]

Рік	Автор	Визначення вузького місця
1	2	3
1997	Daganzo	Обмеження, яке розділяє трафік, що стоїть у черзі, і вільний трафік, що йде далі.
2004	Chen et al	На автомагістралі затори виникають майже в один і той самий час практично щодня.
2005	Bertini and Myton	Точка, перед якою стоїть черга, і за якою вільно протікає транспортний потік.
2007	Ban et al.	Ділянка проїжджої частини, пропускна спроможність якої менша або більша, ніж на інших ділянках.

Продовження таблиці 2.1

1	2	3
2011	Cambridge Systematics, Inc.	Специфічна особливість автомагістралі, яка спричиняє регулярні затори через падіння пропускної здатності, сплески інтенсивності руху або обидва ці фактори.
2011	Florida DOT	Локальна ділянка автомагістралі, на якій спостерігається зниження швидкості та затримки через періодичний експлуатаційний вплив або одноразову подію, що впливає на неї.
2012	Rouphail et al.	Критично важлива ділянка проїжджої частини з чергами перед в'їздом і вільним потоком транспорту за ним.
2015	INRIX	Швидкість на дорозі залишається нижче 60 відсотків від базової швидкості більше 5 хвилин.

Вузьке місце відноситься до явища, коли продуктивність системи обмежена через ресурсу або компонента програми. Ресурсом може бути ЦП, пам'ять, диск, а також мережева карта. Компоненти вузького місця є основними причинами небажаної поведінки і низької продуктивності системи.

Є в основному два типи вузьких місць, які можна пояснити наступним чином:

- вузькі місця, пов'язані з насиченістю ресурсів Коли система повністю використовує ресурс або перевищує певний поріг, ситуація вважається насиченням ресурсу. Продуктивність системи погіршується через насичення ресурсів. Різні системні ресурси по-різному коригуються після того, як вони вичерпані. Завантаження процесора близько 100% призводить до переповненої черги і, таким чином, сприяє збільшенню затримок. Коли система має обмежену пам'ять через обмежену фізичну пам'ять або втрату

пам'яті в системі, відбувається постійний свопінг і підкачка, що призводить до низької продуктивності. Коли система стикається з переповненим диском, постійний доступ до диска за межами доступної пропускної здатності ставить в чергу нові запити на ввід/вивід. Умови насичення мережі через повне використання пропускної здатності впливають на новий трафік, перериваючи або затримуючи обробку;

- вузькі місця, пов’язані з конфліктами ресурсів. Система має обмежені ресурси, такі як цикли процесора, пропускна здатність вводу/виводу, фізична пам'ять, буфери, семафори, м'ютекс і т.д., але прикладні процеси в багатозадачному середовищі будуть конкурувати за ці ресурси, вони обмежені і в результаті створюють вузьке місце в продуктивності. Найбільш доречним прикладом є конфлікт за ресурси між різними хмарними користувачами в хмарних сховищах. Конкуренція за різні системні ресурси має значний вплив на погіршення продуктивності системи. Конфлікт за процесор між декількома процесами призводить до перевантаження черги і зниження продуктивності, особливо у віртуалізованій системі, що використовує програми-монополісти. Конфлікти пам'яті також мають великий вплив на продуктивність. Дискові конфлікти між процесами також знижують продуктивність, особливо з навантаженням вводу/виводу через розрив у продуктивності між процесором і вводом/виводом з дисковим навантаженням. Мережеві конфлікти також знижують продуктивність, оскільки в пікові моменти потрібно більше каналів зв'язку, що зменшує ефективну пропускну здатність.

Поведінка вузьких місць відрізняється для різних систем і додатків. Це регулюється взаємодією між компонентами і системою. Є в основному три типи поведінки вузьких місць:

- одиночні вузькі місця. Вузькі місця в системі обумовлені переважним насиченням ресурсів в одній точці або компоненті системи;

- кілька вузьких місць. Два або більше компонентів системи насичені і одночасно сприяють системним вузьким лініям. Це може статися через взаємозалежність компонентів системи;

- зсувні вузькі місця. Це дещо складна проблема, в якій вузьке місце переміщується від одного компонента до іншого або від однієї точки системи до іншої. Це відбувається через взаємозалежність між компонентами. Додаток може змусити інший додаток змінити свою поведінку, і таким чином зміна поведінки додатку переміщує вузьке місце з одного компонента на інший, і так далі.

Щоб краще зрозуміти продуктивність мережі і зрозуміти її поведінку, нам потрібно її виміряти. Існують різні стандартні та нестандартні метрики, доступні для вимірювання. У цьому проекті ми будемо використовувати деякі з найбільш відомих метрик, такі як затримка, втрати тощо. Короткий огляд мережевих метрик наведено нижче:

- доступність. Показники доступності оцінюють надійність мережі, тобто відсоток часу, протягом якого мережа працює без збоїв;

- втрати. Метрики втрат оцінюють відсоток втрачених пакетів через перевантаженість мережі або помилку передачі. Загасання може бути виміряне для одностороннього шляху або двостороннього шляху, якщо це необхідно;

- затримка – це міра часу, необхідного для того, щоб вантаж дійшов до місця призначення від відправника. Залежно від маршруту прямування, це може бути час в обидва боки або тільки на одному маршруті, що називається затримкою в один бік;

- пропускна здатність – це обсяг даних, який може бути переданий через мережу за одиницю часу, як залежний, так і незалежний від поточного мережевого трафіку. Крім цих показників продуктивності, нам потрібно шукати інші нестандартні показники, які часто пов'язані з системою і можуть спричинити погіршення продуктивності мережі. Тому моніторинг системних ресурсів, таких як процесор, пам'ять і мережеве навантаження, дає загальне

уявлення про систему і стан ресурсів. Таким чином, результат не обдурить нас у випадку, якщо система спричиняє проблеми зі зниженням продуктивності.

2.2 Вузьке місце Інтернету

Звичайно, коли ви шукаєте нове офісне обладнання, ви очікуєте, що воно буде працювати краще, ніж ваші старі машини. Іноді вік вашого мережевого устаткування – не єдина проблема, що впливає на продуктивність мережі. У більшості випадків проблемою може стати вузьке місце мережі.

Якщо ви не розумієте, вузьке місце – це термін, названий на честь форми пляшки; вузьке вгорі, ширше внизу. Шийка пляшки призначена для контролю кількості рідини, яка виходить з пляшки (рисунок 2.1).

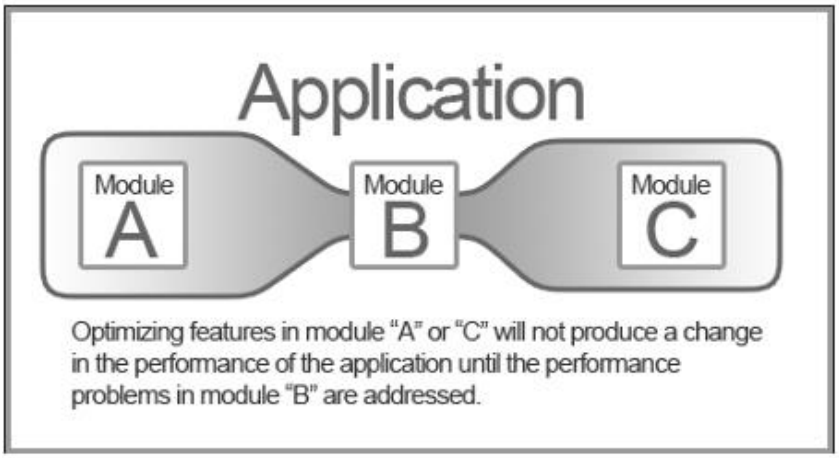

Рисунок 2.1 – Приклад схеми вузького місця

На жаль, це часто простіше сказати, ніж зробити, щоб знайти джерело вузького місця у вашій корпоративній мережі. Якщо ви хочете досягти успіху на робочому місці, вам потрібно знайти джерело проблеми і виправити його якомога швидше. У більшості випадків усунути проблему так само просто, як

і оновити джерело вузького місця, щоб продуктивність була на одному рівні з іншою частиною вашої мережі.

Прикладом цього є оновлення лише одного аспекту вашого ПК і очікується загальне підвищення продуктивності. Наприклад, ви можете оновити свою відеокарту, сподіваючись покращити продуктивність ПК. Це рішення може покращити вигляд екрана, але суттєво не вирішить проблеми з процесором або оперативною пам'яттю. Якщо вони застаріли та вже досягли свого обмеження, оновлення відеокарти не підвищить продуктивність ПК, оскільки це вузьке місце.

Ймовірність вузького місця у вашій мережі зростає зі складністю технології. В результаті, коли мова йде про IT-інфраструктуру зі складними робочими станціями, серверними блоками та іншим критично важливим для бізнесу обладнанням, ви зіткнетеся з значно більшою кількістю мережевих проблем. Наприклад, якщо у вас є робочі станції високого класу, але ваш маршрутизатор не справляється із завданням, це вузьке місце заважає вам максимізувати свій IT-потенціал.

Вузькі місця, як ключові фактори підвищення продуктивності виробничих мереж, інтенсивно вивчалися протягом останнього десятиліття. Однак через складність результатів дослідження існує значний розрив між теорією та практикою. Ще однією перевагою виявлення вузьких місць є те, що новітні дослідження вузьких місць можуть значно зменшити обчислювальну складність, пов'язану з обчисленням продуктивності системи. Підхід, заснований на виявленні вузьких місць, та асимптотичні результати розглядаються для того, щоб показати внесок вузьких місць в оцінку продуктивності та теоретичний аналіз.

Вузьке місце може виникнути в мережі користувача або у структурі пам'яті або на серверах, які знаходяться в надмірному конфлікті для внутрішніх ресурсів сервера, таких як потужність обробки процесора, пам'ять або I / O (ввід / вивід). Це уповільнює потік інформації до швидкості найповільнішої точки на шляху даних. Це сповільнення впливає на

продуктивність застосунків, особливо з базами даних та іншими великомасштабними транзакційними застосунками, і може навіть призвести до аварійного завершення роботи деяких застосунків.

Вузьке місце, як правило, викликано погане проектування мережі або структуру зберігання даних. Несумісність вибору обладнання може бути поширеною причиною. як приклад, якщо сервер робочої групи охоплює порт Gigabit Ethernet, але відповідний порт комутатора, підключений до сервера, має лише старий порт 10/100 Ethernet, повільний порт комутатора може бути вузьким місцем. Ще один недолік дизайну в вантажних палубних мережах – це надмірний вентилятор, де кілька пристроїв зберігання підключені до ідентичного порту комутатора, щоб максимізувати пропускну здатність цього порту комутатора. Наприклад, якщо кілька пристроїв зберігання 4 гігабітних (GB) Fiber Channel підключені до ідентичного порту комутатора, порт комутатора може легко перевантажуватися і викликати проблеми з продуктивністю, коли кілька пристроїв зберігання активні в той же час. У багатьох випадках вузькі місця виникають з плином часу, тому що адміністратори не в змозі задовольнити зростаючі потреби мережі та трафіку зберігання.

Вузькі місця також можуть виникати через погану або неоптимальну конфігурацію перемикачів або адаптерів хост-шини (HBAs). Як приклад, використання декількох портів Fibre Channel для з'єднання пристроїв в межах комутаційної системи зберігання даних може підвищити доступність і продуктивність системи, але якщо пов'язані пристрої не налаштовані на балансування навантаження, багато переваг втрачаються. Вузькі місця можуть виникати через апаратні помилки. У попередньому прикладі припускаємо, що одне з двох з'єднань Fibre Channel вийде з ладу. Незважаючи на те, що перехід на інший ресурс повинен дозволити доступ до пристрою, весь трафік, який раніше був перенесений через два канали, тепер переключається на 1, що може спричинити вузьке місце, якщо трафік перевищує пропускну здатність.

Вузькі місця, як правило, розташовані шляхом систематичного тестування продуктивності мережі на різних пристроях на шляху знань та ізоляції пристроїв, які помітно повільніше, ніж інші точки. Після виявлення вузьке місце, як правило, може бути вирішене шляхом переналаштування, модернізації або заміни відповідного пристрою. На мережевому рівні це може включати в себе оновлення комутатора. Для серверів може допомогти оновлення процесора або пам'яті, або сервер може бути повністю замінений (наприклад, заміна застарілого однопроцесорного сервера на більш пізній двопроцесорний сервер або чотири процесори). Вузькі місця часто можна уникнути, проактивно відстежуючи тенденції транспортного навантаження з плином часу та впроваджуючи поліпшення, перш ніж виникнуть серйозні проблеми.

Якщо ваша мережа працює повільніше, ніж зазвичай, або якщо передавання файлів до сервісу триває довше, або якщо повідомлення електронної пошти не досягло місця призначення, можливо, у мережі є вузьке місце. По суті, це сингулярна точка, де система відстає через широкий спектр питань.

Нижче наведено список найпоширеніших вузьких місць у мережі:

- пропускна здатність каналів. Якщо до мережі підключено більше одного користувача, більш імовірним є вузьке місце. Для комерційних проектів це менш поширено, оскільки вони, як правило, мають більш високу пропускну здатність для підтримки декількох користувачів одночасно. Але, незважаючи на це, хтось може використовувати занадто багато даних, щоб інші почали помічати різницю.Загальними винуватцями є великі завантаження файлів, потокове передавання та відтворення відео. Рішення полягає в тому, щоб знайти, який з них використовує всю цю смугу пропускання і обмежити його використання для відновлення швидкості;

- обмежена пропускна здатність. Ваша поточна мережа також, ймовірно, не досить швидко, щоб підтримувати таку кількість користувачів одночасно. Це поширене серед компаній, оскільки вони продовжують

залучати більше людей до команди, не збільшуючи загальну пропускну здатність для їх підключення. Незважаючи на оптимальне використання, ви, ймовірно, будете задушені мережею через занадто багато людей, які намагаються використовувати підключення одночасно.

Єдиним рішенням є внесення змін у ваше інтернет-з'єднання або пошук способу переміщення користувачів на інше з'єднання, щоб компенсувати деяку частину навантаження:

- сервери перевантажено. Ще однією поширеною причиною того, що компанії страждають від нестачі пропускної здатності, є надмірне навантаження на їх сервери. Оскільки сервери обробляють одночасно велику кількість різноманітних служб, включаючи системи обміну повідомленнями, бази даних та інші рішення, вони можуть виконувати занадто багато операцій одночасно.

Перевантажений сервер – це ознака того, що вам потрібно оновити і встановити новий сервер, щоб розділити завантаження всіх ваших додатків. Це може бути менш здійсненним, щоб уповільнити або зупинити деякі з сервісів, що працюють на диску, так як вони можуть бути невід'ємною частиною різних операцій.

Можливість аналізу вузьких місць мережі вздовж кінцевих шляхів в Інтернеті ідеально підходить для мережевих операторів та наукових співробітників. Наприклад, якщо оператори мережі знають, де знаходяться вузькі місця, вони можуть застосувати інженерію трафіку на міждоменному або внутрішньодоменному рівні для поліпшення маршрутизації. Існуючі інструменти не розпізнають розташування вузьких місць або створюють велику кількість тестових пакетів.

Як я вже згадував раніше, існує безліч форм інтернет-вузьких місць, а саме:

- застряг на початку. Перший тип вузького місця – це функція централізованої моделі, яка використовується сьогодні для доставки контенту в Інтернет. Кожен постачальник вмісту налаштовує веб-сайт в

одному фізичному місці та надає дані, послуги та інформацію всім користувачам Інтернету по всьому світу з цього центрального розташування. Це означає, що швидкість, з якою користувачі можуть отримати доступ до сайту, неминуче обмежена "першою милею" – пропускною здатністю інтернет-з'єднання сайту. Для того, щоб виправдати зростаючу кількість користувачів у цій моделі, не тільки кожен постачальник контенту повинен купувати все більші та більші з'єднання зі своїм провайдером, але інтернет-провайдер також повинен продовжувати розширювати можливості своєї внутрішньої мережі, і те ж саме стосується сусідніх мереж. Оскільки неможливо з таким підходом йти в ногу з експоненціальним зростанням інтернет-трафіку, централізована модель доставки контенту за своєю суттю немасштабована. Є багато прикладів вузьких переходів, що впливають на продуктивність та доступність веб-сайту (рисунок 2.2);

- піринг: точки скупчення. Другий тип вузького місця інтернету відбувається в точках взаємозв'язку: точки взаємозв'язку між незалежними мережами. Причини, чому точки пірінга є вузькими місцями, в першу чергу економічні.

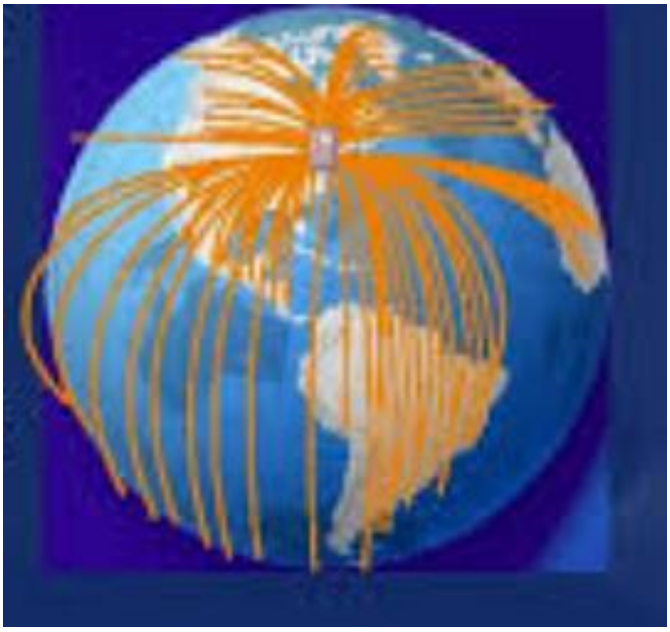

Рисунок 2.2 – Застряг на початку

По-перше, у мереж мало стимулів для створення безкоштовних пірінгових заходів, оскільки в таких механізмах немає можливостей для отримання доходу, але витрати на налаштування є значними. У той же час, жодна з великих мереж не погодиться заплатити іншій велику мережу для обміну трафіком, оскільки з точки зору трафіку, обидві однаково виграють від такої домовленості. В результаті, великі мережі в кінцевому підсумку не збігаються один з одним, а обмежена кількість точок обміну між ними в кінцевому підсумку стає вузькими місцями.

Один з найбільш поширених типів пірінга відбувається, коли невелика мережа купує підключення великої мережі. Проблема, яка виникає в цій ситуації, – це час, необхідний для забезпечення необхідних схем. Незважаючи на те, що в землі вже є багато темних або невикористаних волокон, фірмові телефонні компанії вважають за краще встановлювати нові волокна для кожної необхідної схеми, щоб збільшити свої доходи. В результаті на запуск нових схем обміну трафіком йде від 3 до 6 місяців.

При встановленні запитаної схеми трафік може збільшитися вище очікуваних, що призведе до повного зв'язку – вузького місця. Це також правда, що це не має економічного сенсу для мережі, щоб платити за невикористану потужність. Таким чином, мережа, як правило, набуває достатньої потужності для обробки поточних рівнів трафіку, так що всі пірінгові посилання є повними. На жаль, така практика експлуатації посилань на повну потужність створює серйозні вузькі місця. Посилання, що працюють під дуже високим використанням, мають високу швидкість втрати пакетів і дуже високу затримку (через час очікування маршрутизатора) навіть для пакетів, які можуть пройти. В результаті веб-продуктивність сповільнюється.

Хоча було висловлено думку, що вузькі місця будуть розсіюватися через консолідацію телекомунікацій, недавні дослідження про структуру Інтернету та склад його трафіку свідчать про інше. Зокрема, він показує, що

Інтернет є і буде як і раніше складатися з тисяч мереж, жодна з яких окремо не несе значну частину трафіку доступу користувачів.

На рисунку нижче представлений відсоток загального трафіку доступу, що обробляється кожною з 7400+ інтернет-мереж.

Трафік доступу до мережі, виміряний в бітах / сек., складається з пакетів даних від клієнтів або серверів, безпосередньо підключених до мережі. На відміну від цього, транзитний трафік у мережі складається з пакетів даних, що надходять з інших мереж. Трафік доступу до мережі – це обсяг трафіку, який ваш кінцевий користувач і база абонентів веб-сайту відправляє в Інтернет, тобто це показник обсягу трафіку, який мережа сприяє загальному навантаженню інтернет-трафіку.

Цей графік показує, що трафік доступу користувачів досить рівномірно розподілений між 7400 мережами, причому жодна мережа не становить більше 5%, а більша частина становить менше 1%. Тому проблема пееринга вузького місця – це явно масштабна проблема, властива опточному інтернету.

Попередні дослідження історичних даних показують, що кількість мереж, з яких складається Інтернет, зростає, що є ще однією ознакою того, що вузькі місця, швидше за все, збережуться, а не зникнуть;

- зламаний кістяк. Третій тип вузького місця інтернету – це пропускна здатність великих, далеко-магістральних мереж, які формують кістяк Інтернету. Оскільки сьогоднішня централізована модель контент-послуг вимагає, щоб майже весь інтернет-трафік перетинав один або кілька магістралей, пропускна здатність цих мереж повинна бути в змозі зростати так само швидко, як інтернет-трафік.

Пропускна здатність мережі визначається пропускною здатністю кабелів і маршрутизаторів. Оскільки оптоволокно є дешевим, доступним і здатним забезпечити високу пропускну здатність, пропускна здатність кабелю не є проблемою. Натомість, саме маршрутизатори на кінцях оптоволоконних кабелів обмежують пропускну здатність магістралі.

Швидкість апаратного і програмного забезпечення маршрутизаторів для пересилання пакетів завжди обмежена сучасними технологіями. Історично склалося так, що маршрутизатори не забезпечували такої функціональності якості обслуговування (QOS), як комутатори операторів зв'язку, а збільшення пропускної здатності маршрутизаторів не встигало за зростаючими об'ємами трафіку. Інтернет. Насправді, багато провайдерів використовують мережі з комутацією IP через ATM, оскільки IP-маршрутизатори не в змозі задовольнити їхні потреби в трафіку. Однак мережа ATM дорожча в налаштуванні та обслуговуванні.

І хоча провайдери магістральних мереж також витрачають багато грошей на модернізацію своїх маршрутизаторів, щоб впоратися зі зростаючим попитом на трафік, врешті-решт вони значно перевищать свої можливості. Така велика невідповідність між попитом на пропускну здатність магістральної мережі та її пропускною спроможністю робить магістральну мережу все більш серйозним вузьким місцем;

- остання миля. Більшість користувачів інтернету розуміють, що таке "остання миля": це обмежена пропускна здатність вашого модемного з'єднання зі швидкістю 56 Кбіт/с з вашим провайдером. Однак поширеною помилкою є думка, що усунення цього вузького місця вирішить всі проблеми з продуктивністю інтернету, забезпечивши високошвидкісний доступ до нього для всіх бажаючих. Насправді, обмежуючи доступ до Інтернету, 56 Кбіт/с модеми рятують Інтернет від колапсу. Якби всі користувачі мали доступ до Інтернету через мультигігабітний кабельний модем або DSL-модеми, інші три типи вузьких місць зробили б Інтернет нестерпно повільним. Іншими словами, Інтернет настільки швидкий, наскільки повільним є його найповільніше з'єднання. Це можна легко зрозуміти на прикладі кабельного провайдера @Home. Порівняно із загальною кількістю користувачів Інтернету, @Home має дуже малу абонентську базу: лише 1 мільйон абонентів порівняно з 377 мільйонами, тобто лише 0,3%.

Однак ці широкосмугові абоненти становлять 5% трафіку в Інтернеті. Таким чином, вища пропускна здатність з'єднання відповідає вищому попиту на пропускну здатність Інтернету. Якби у 376 мільйонів користувачів Інтернету були б широкосмугові з’єднання, Інтернет зник би.

Оскільки Інтернет вже працює майже на повну потужність, його три інші головні вузькі місця повинні бути видалені, перш ніж мільйони додаткових користувачів широкосмугового зв'язку.

3 АНАЛІЗ МЕТОДІВ ПОШУКУ ВУЗЬКИХ МІСЦЬ У СКЛАДНИХ МЕРЕЖЕВИХ СИСТЕМАХ

3.1 Формальний опис вузьких місць у мережі

Складні системи, такі як комунікаційні та комп'ютерні мережі, складаються з ряду взаємодіючих частинок (або клієнтів), які демонструють значні явища перевантаження зі збільшенням ступеня взаємодії.

Динаміка цих систем визначається випадковістю подій, що лежать в їх основі, наприклад, прибуттям робочих одиниць, і може бути стохастично описана у вигляді моделей мереж масового обслуговування. Якщо вони є керованими, то вони дозволяють прогнозувати досяжну продуктивність системи, оптимізувати конфігурацію мережі та проводити дослідження з планування пропускної спроможності. Ці цілі, як правило, важко досягти без математичної моделі, оскільки реальні системи є величезними;

Важливим питанням, пов'язаним з інфраструктурою інформаційних технологій (IT), є визначення проблемних місць, які зазвичай називають "вузькими місцями" (bottlenecks). Ці точки перевантаження – це ресурси, які проектувальник повинен досліджувати, щоб досягти значних поліпшень, а їх знання дає точну інформацію про продуктивність системи. Відомо, що саме найбільш критичні ресурси обмежують загальну продуктивність. Оскільки кількість вузьких місць, як правило, набагато менша за загальну кількість ресурсів, такої якісної поведінки можна досягти з обмеженими обчислювальними можливостями. Однак проблема їх виявлення не є тривіальною, оскільки вони можуть переміщатися між різними ресурсами в залежності від ряду факторів, наприклад, від співвідношення робочих навантажень. Крім того, сучасні комп'ютерні системи є динамічними, самоконфігуруються, самооптимізуються, і в цьому контексті потрібні швидкі та ненав'язливі методи ідентифікації. Моделі мереж із закритими

чергами широко використовуються в літературі для проведення вищезгаданого аналізу.

Приклад 1: Аналіз структури вузьких місць.

Аналітична сила теорії структури вузьких місць ґрунтується на її здатності фіксувати вплив вузьких місць і потоків один на одного і, зокрема, точно визначати їх кількісно. Ця здатність може бути застосована до широкого спектру мережевих проблем. Наприклад, використання похідних виду dT / dcl є природним способом вивчення проблеми оптимальної модернізації мережі. Похідна загальної пропускної здатності від пропускної здатності кожної ланки показує, які ланки необхідно модернізувати, щоб мати максимальний вплив на загальну продуктивність мережі. Інші проблеми, пов'язані з проектуванням мережі та плануванням пропускної здатності, можуть бути вирішені за допомогою подібних методів. Теорія структури вузьких місць також висвітлює питання управління потоками, такі як маршрутизація та інженерія трафіку. Наприклад, якщо ми хочемо збільшити продуктивність певного високопріоритетного потоку і знаємо, які потоки мають низький пріоритет, ми можемо розрахувати похідні пропускної здатності високопріоритетного потоку, щоб визначити, який з низькопріоритетних потоків слід формувати. Ми також можемо зробити точні кількісні прогнози щодо того, наскільки це втручання покращить продуктивність. Існують також застосування в інших сферах. Наприклад, визначення того, де конкретний потік даних є вузьким місцем, хто контролює вузьке з'єднання і як решта трафіку в мережі впливає на потік даних, може допомогти в моніторингу та управлінні угодами про рівень обслуговування (SLA).

Однією з проблем аналізу мережі на практиці є те, що мережеві умови змінюються щосекунди. Необхідність аналізувати мережі в режимі реального часу висуває ще більш жорсткі вимоги до продуктивності, які попередня техніка не могла задовольнити.

Існує два алгоритми побудови структур вузьких місць. Псевдокод представлений в алгоритмі 1 під назвою ComputeBS (лістінг 3.1). Під час кожної ітерації головного циклу розривається ряд зв'язків, що означає, що швидкості всіх потоків, які вони перетинають, назавжди фіксуються. До цієї групи належать ті, чия "справедлива частка корисності" sl в цій ітерації (рядок 12) є найменшою з усіх ланок, з якими вони мають спільне живлення (рядок 13). У рядку 15 визначаються швидкості всіх річок, що перетинають маршрут 1, які не були визначені раніше, і маршрут та його потоки позначаються як розв'язані (рядки 18 і 19). Він також додає правильно орієнтовані ребра на графік структури вузьких місць – шляхи до потоків, які їх звужують (рядок 16) і потоків до шляхів, які вони перетинають, але не звужують (рядок 17). Алгоритм повертає структуру вузьких місць G = hV, Ei, параметри з'єднань {sl, l L} та прогнозовані витрати {rf, ∀f ∈ F}. Ця процедура є аналогічною до запропонованого алгоритму, але з додатковою логікою побудови графічного представлення структури вузьких місць. Її обчислювальна складність становить O (H · | L | 2 + | L | · | F |), де L – множина сполучень, F – множина потоків і H – максимальна кількість сполучень, яке може перетнути потік.

Лістінг 3.1 – ComputeBS(N = <L, F, {cl , ∀l ∈ L}>)

```
1: L := Набір посилань у вхідній мережі;
2: F := Набір потоків у вхідній мережі;
3: Fl := Набір потоків, що проходять через посилання l;
4: cl := Ємність посилання l;
5: B := Набір посилань на вузькі місця;
6: rf := Швидкість потоку f;
7: Lk := Набір нерозв'язаних посилань при ітерації k;
8: C k := Набір вирішених потоків при ітерації k;
9: L0 = L; C 0 = ∅; k = 0;
10: E = ∅;
11: while C k != F do
12: s kl = (cl − P ∀f∈Ck∩Fl rf )/|Fl \ Ck|, ∀l ∈ Lk;
13: u kl = min{s kl | F1∩ Fl != ∅, ∀l ∈ Lk}, ∀l ∈ Lk;
14: for l ∈ Lk, s kl = u kl  do
15: rf = s kl, ∀f ∈ Fl \ Ck;
```

```
16: E = E ∪ {(l, f), ∀f ∈ Fl \ Ck };
17: E = E ∪ {(f, ll ), ∀f ∈ Fl \ Ck and ∀l 0 ∈ Lf | s kl != u kl
};
18: Lk = Lk \ {l};
19: C k = C k ∪ {f, ∀f ∈ Fl};
20: end for
21: Lk+1 = Lk; C k+1 = C k ;
22: k = k + 1;
23: end while
24: B = L \ Lk; V = B ∪ F; sl = s k l , ∀l ∈ B;
25: G = (V, E);
26: return hG, {sl , ∀l ∈ L}, {rf , ∀f ∈ F}i;
```

Ми описуємо алгоритм FastComputeBS (лістінг 3.2), вдосконалений алгоритм розрахунку структури вузьких місць з асимптотично швидшим часом виконання, ніж ComputeBS. Цей алгоритм розв'язує посилання по одному, але на відміну від ComputeBS, він зберігає посилання в незграбній структурі даних, відсортовані відповідно до того, скільки пропускної здатності вони можуть виділити потокам, що проходять через них. Це дозволяє алгоритмові розв'язувати посилання у правильному порядку без пошуку набору посилань на кожній ітерації, уникаючи таким чином затратного обчислення min {} алгоритму 1 (рядок 13). FastComputeBS зменшує асимптотичний час виконання обчислення структури вузького місця структури до O (| E | · log | L |), де | E | кількість граней у структурі вузького місця і | L | це кількість посилань. За визначенням, існує границя для кожної пари потоку і зв'язок, який він перетинає. Таким чином, час виконання є майже лінійним з розміром запису.

Лістінг 3.2 – FastComputeBS(N = <L, F, {cl , ∀l ∈ L}>)

```
1: V = ∅; E = ∅; rf = ∞, ∀f ∈ F;
2: for l ∈ L do
3: al = cl; # доступна ємність
4: sl = al/|Fl |; # достатня частка
5: MinHeapAdd(key = sl, value = l);
6: end for
7: y while F ⊄V do
8: l = MinHeapPop();
9: for f ∈ Fl such that rf ≥ sl do
```

```
10: E = E ∪ {(l, f)};
11: if f ∉ V then
12: rf = sl ;
13: V = V ∪ {f};
14: for l 0 ∈ Lf such that rf < sl 0 do
15: E = E ∪ {(f, l0 )}
16: al 0 = al 0 − sl
17: sl 0 = al/|Fl \ V|;
18: MinHeapUpdateKey(value = l 0 , newKey = sl 0);
19: end for
20: end if
21: end for
22: V = V ∪ {l};
23: end while
24: return hG = hV, Ei, {sl, ∀l ∈ L}, {rf , ∀f ∈ F}i;
```

За допомогою програмного забезпечення, такого як Packet Sniffer, легко перехоплювати або захоплювати мережеві пакети, які проходять через мережу.

Для перехоплення пакетів у мережі він складається з інтерфейсу програмування. Libpcap використовується для перехоплення мережевих пакетів безпосередньо з мережевої карти. Ця бібліотека є інтегрованою функцією операційної системи. На UNIX-подібних системах перехоплення пакетів реалізовано у бібліотеці libpcap. У Windows використовується порт libpcap, відомий як WinPcap. Він пропонує функції перехоплення і фільтрації пакетів. Якщо операційна система не має цієї бібліотеки, її можна встановити пізніше, оскільки вона доступна з відкритим кодом (рисунок 3.1, рисунок 3.2).

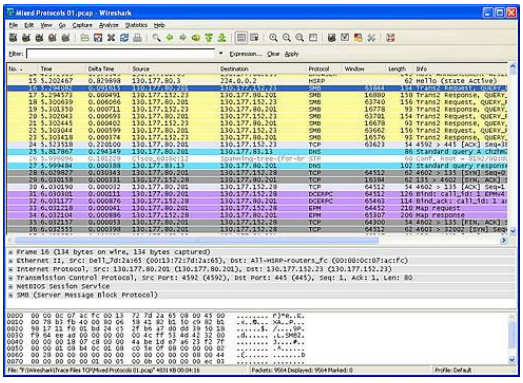

Рисунок 3.1 – Приклад перехоплення пакетів

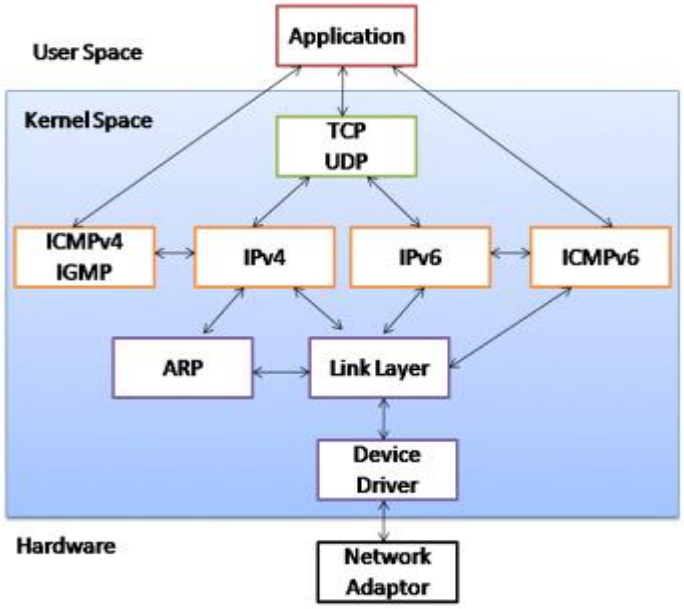

Рисунок 3.2 – Потік пакетів від рівня додатків до мережевого адаптера

Коли мережева карта отримує пакет, вона спочатку порівнює MAC-адресу, вказану в пакеті, зі своєю власною MAC-адресою. Якщо вони збігаються, мережева карта приймає пакет, інакше – фільтр. На ілюстрації показано типову мережеву інтерфейсну карту (NIC). Режим роботи, в якому мережева карта опрацьовує лише адресовані їй пакети, називається нерозбірливим (non-promiscuous). Для того, щоб перехоплювати пакети, мережевий адаптер повинен бути переведений у режим прослуховування (promiscuous). Таким чином, програми-шпигуни перехоплюють пакети, переводячи мережеву карту у режим безладного перехоплення, а отже, отримуючи всі пакети, навіть якщо вони не призначені для неї. Щоб перевести мережеву карту у безладний режим, достатньо надіслати спеціальний виклик ioctl () до відкритого сокета на цій карті, і пакети буде передано ядру.

Приклад 3: Зміщення вузького місця.

Залежно від типу аналізованої системи виробництва, іноді може бути досить важко визначити вузькі місця. Roser, Nakano і Tanaka ввели метод виявлення зсувних вузьких місць в 2002 році, тому що вони виявили, що

часто використовувані методи (метод використання і метод довжини черги) мали кілька недоліків.

У методі використання, коли машина з найвищим завантаженням вважається вузьким місцем, є три основні недоліки: По-перше, використання на різних машинах часто дуже схоже, що ускладнює висновки про те, яка з них дійсно є вузьким місцем. По-друге, сам метод обмежений рамками стаціонарного використання, і, по-третє, він не може виявити поточне вузьке місце, а лише дає нам середнє значення вузького місця за тривалий період часу. Іншим поширеним методом є аналіз довжини черг або часу очікування перед операціями. Елемент з найбільшою довжиною черги або часом очікування вважається вузьким місцем. Roser та ін. вбачають перевагу цього методу в тому, що він дозволяє розпізнавати поточні вузькі місця, але вважають, що його недоліки є різноманітними. Основним недоліком є те, що багато виробничих ліній мають обмежений час очікування або взагалі не мають буферів, тому вузькі місця не можуть бути виявлені за допомогою цього критерію. Якщо кількість вхідного матеріалу перевищує пропускну здатність системи, терміни затримки і час очікування не можуть показати, де знаходиться вузьке місце. Якщо розміри пакетів варіюються від машини до машини, вузькі місця можуть бути визначені неправильно. Нарешті, Roser та ін. вказують, що час очікування залежить від багатьох факторів і розподіляється нерівномірно, що ускладнює оцінку точності середньої довжини черги або періоду очікування з плином часу. Для подолання вищезгаданої перешкоди Roser та ін. представляють метод виявлення вузьких місць, що зміщуються (Roser, Nakano and Takana 2002). За допомогою цього методу можна визначити та відстежувати поточні зміни вузького місця, а також середнє значення вузького місця за обраний період часу. Метод базується на тривалості часу, протягом якого машина працює без перерв. Активний час – це момент, коли машина змушує наступну машину чекати. На практиці це означає кожен раз, коли машина перебуває в роботі, на ремонті, для заміни інструменту або на обслуговуванні. Поточне вузьке місце

– це обладнання з найдовшим безперервним активним періодом у будь-який момент часу. Для визначення середнього показника "вузького місця" аналізуються дані за обраний період і визначаються поточні "вузькі місця". Потім вимірюється відсоток часу, протягом якого машина є єдиним вузьким місцем, і відсоток машини, яка є частиною мінливого вузького місця. Потім ці два відсотки додаються, і чим більша сума, тим більший вплив на машину в загальній системі. Roser та Alabama зазначає, що метод виявлення "змінних вузьких місць" має багато переваг порівняно з іншими методами. Однак у цього методу є деякі обмеження. Хоча метод показує нам кілька вузьких місць і класифікує їх за розміром, неможливо побачити, що потрібно поліпшити з кожним обмеженням.

Приклад 4: Метод SCORE.

Метод SCORE (The Simulation-based Constraint Removal) був розроблений для вирішення деяких з описаних вище проблем, пов'язаних з найбільш поширеними методами виявлення перевантажень. Метод SCORE використовує багатоцільову оптимізацію на основі симуляції для пошуку найбільш вигідних поліпшень для виробничої лінії. Шляхом встановлення покращених значень для таких параметрів, як тривалість циклу, час безвідмовної роботи і час ремонту, мінімізації кількості змін при максимізації продуктивності, знаходять рішення, які пропонують найбільш вигідне покращення продуктивності з найменшою кількістю змін в системі. Параметри, які найчастіше зустрічаються в результатах оптимізації, розглядаються як перше обмеження, другі за величиною – як друге, і так далі. Таким чином можна визначити порядок різних вузьких місць у системі. Основних недоліків методу SCORE, на думку Pehrsson, три:

- метод вимагає великої обчислювальної потужності для вирішення проблем в розумні терміни;
- імітаційна модель для виробничої лінії має важливе значення;
- конкретні параметри SCORE повинні бути реалізовані в моделі і інтеграція з алгоритмом управління повинна бути проведена.

3.2 Аналіз методів пошуку вузьких місць

Аналіз вузьких місць у мережах передачі даних є предметом інтенсивних досліджень з 1988 року, коли Van Jacobson запропонував перший алгоритм для боротьби з перевантаженням. Цей прорив буквально врятував Інтернет від колапсу через перевантаження.

Однак більшість досліджень останніх трьох десятиліть ґрунтувалися на ідеї, що продуктивність передачі даних визначається виключно пропускною спроможністю вузького місця і часом обміну даними на цьому шляху. Таке бачення призвело до появи десятків алгоритмів управління перевантаженням, заснованих на визначенні (явно або неявно) продуктивності кожного вузького місця потоку.

Вузькі місця можуть виникнути в будь-який час і в будь-якому місці при великомасштабному аналізі даних. Для того, щоб вчасно уникнути вузьких місць, ми повинні спочатку правильно розпізнати їх під час виконання. Lube здійснює онлайн виявлення вузьких місць у зібраних показниках продуктивності в режимі реального часу. Ми розглянули два методи виявлення вузьких місць з часових рядів ключових показників ефективності. Одна з них – проста статистична модель: алгоритм авторегресійного інтегрованого змінного середнього (ARIMA), який апроксимує майбутнє значення за допомогою лінійної функції минулих значень і минулих помилок; інша – модель неконтрольованого машинного навчання: алгоритм змінної прихованої моделі Markov (Sliding Hidden Markov Model, SlidHMM), який може автономно вивчати неявну кореляцію між кількома показниками продуктивності.[14]

Метод 1: APIMA.

Модель ARIMA, яку представили Box та Jenkins, широко використовується в аналізі часових рядів. Як поєднання авторегресійної моделі (AR) та модель змінного середнього (MA), модель ARIMA визначається наступним рівнянням:

$$y_t = \theta_0 + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \cdot\cdot\cdot + \varphi_p y_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdot\cdot\cdot - \theta_q \varepsilon_{t-q}, \quad (3.1)$$

де y_t і ε_t позначають дійсне значення та випадкову похибку в момент часу t;

φi (i = 1, 2, ..., p) і θ_j (j = 1, 2, ..., q) – коефіцієнти, задані моделлю;

p і q — цілі числа, які вказують авторегресивні (AR) та змінні середні поліноми (MA) відповідно;

де y_t і ε_t позначають дійсне значення та випадкову похибку в момент часу t;

φi (i = 1, 2, ..., p) і θ_j (j = 1, 2, ..., q) – коефіцієнти, задані моделлю;

p і q — цілі числа, які вказують авторегресивні (AR) та змінні середні поліноми (MA) відповідно.

Наприклад, якщо q = 0 тоді рівняння. (1) зведено до AR-моделі порядку p; Якщо p = 0 тоді рівняння. (1) зводиться до моделі MA порядку q. Загальна модель ARIMA представлена у вигляді ARIMA (p, d, q), де d – ступінь відмінності перетворення для стаціонарності даних. Метод ARIMA складається з трьох ітераційних кроків. По-перше, ми перевіряємо стаціонарність вхідного часового ряду за допомогою тесту Дікі-Фуллера. Якщо часовий ряд не є стаціонарним, ми перетворюємо його на стаціонарний часовий ряд, застосовуючи відповідний ступінь диференціації (визначається d). По-друге, ми досліджуємо автокореляційну функцію (АКФ) та часткову автокореляційну функцію (ЧАФ) вхідного часового ряду, щоб оцінити відповідні p та q (рисунок 3.3).

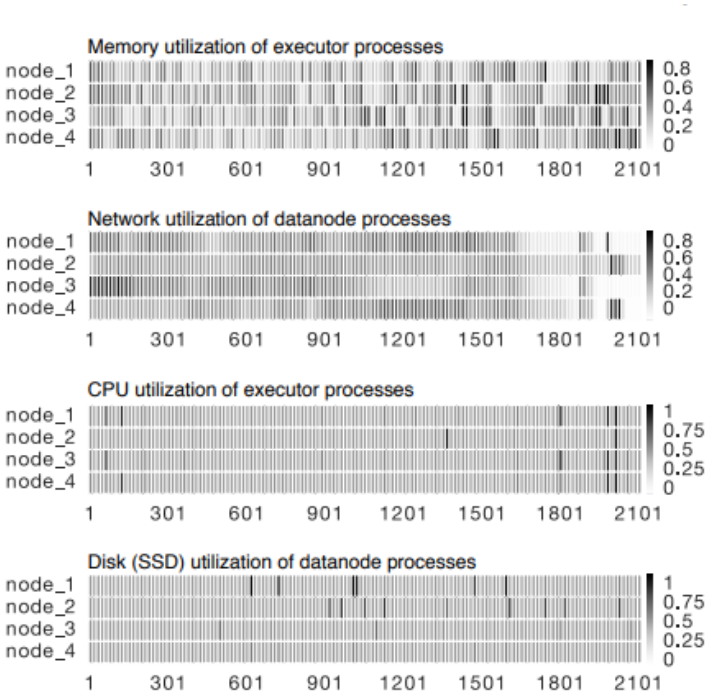

Рисунок 3.3 – Теплові карти показників продуктивності

По-третє, ми перевіряємо залишки (фактичні значення мінус скориговані значення), щоб діагностувати, чи є модель належною. Якщо модель не підходить, ми повторюємо попередні кроки. Ми створюємо одновимірну ARIMA-модель для кожного показника ефективності, а не векторну ARIMA-модель для всіх показників, оскільки вона, як правило, "перенастроюється" через занадто велику кількість незначущих комбінацій параметрів. Для підтримки виявлення вузьких місць в режимі онлайн ми регулярно оновлюємо ARIMA-модель за допомогою показників ефективності, які постійно надходять.

Метод 2: Slid XMM.

Прихована модель Маркова (HMM) виводить послідовність прихованих станів, які відображаються на послідовність спостережуваних станів. Надавши HMM часовий ряд спостережуваних KPI (від O1 до Od), ми можемо отримати можливі Ok KPI в майбутньому.

HMM зазвичай визначається як кортеж з трьох елементів: (A, B, π) у наступних позначеннях:

$$Q = \{q_1, q_2. . . , q_N\}, \text{ прихована послідовність станів,} \quad (3.2)$$

$$O = \{O_1, O_2. . . , O_k\}, \text{ послідовність станів спостереження,} \quad (3.3)$$

$$A = \{a_{ij}\}, a_{ij} = Pr\ (q_j \text{ at } t + 1 \mid q_i \text{ at } t), \text{ матриця переходу,} \quad (3.4)$$

$$B = \{b_j\ (k)\}, b_j\ (k) = Pr\ (O_k \text{ at } t \mid q_j \text{ at } t), \text{ матриця викидів,} \quad (3.5)$$

$$\pi = \{\pi_i\}, \pi_i = Pr\ (q_i \text{ at } t = 1), \text{ розподіл початкового стану,} \quad (3.6)$$

де t – часова мітка.

ПММ вивчає приховані стани, засновані на алгоритмі максимізації очікування, алгоритмі Baum-Welch. Цей алгоритм ітеративно шукає параметри моделі (A, B,), які максимізують ймовірність Pr(O | μ), найкраще пояснення послідовності спостережень. Траєкторії використання купи JVM на Рисунку 3.4 показують чітку періодичну закономірність. Знаючи стани, що лежать в основі цієї моделі, HMM виводить майбутні показники продуктивності для планування з урахуванням вузьких місць. Щоб підтримувати виявлення вузьких місць під час виконання, HMM повинен оновлюватися в режимі онлайн. Однак таке оновлення в режимі онлайн призводить до значних витрат часу та простору, оскільки алгоритм Baum-Welch повинен перераховувати вхідні дані старого та нового часових рядів. Тому ми пропонуємо використовувати модель Sliding Hidden Markov (SlidHMM), яка є змінною версією класичної HMM і була спеціально розроблена для онлайн-оцінки часових рядів високої щільності (рисунок 3.4).

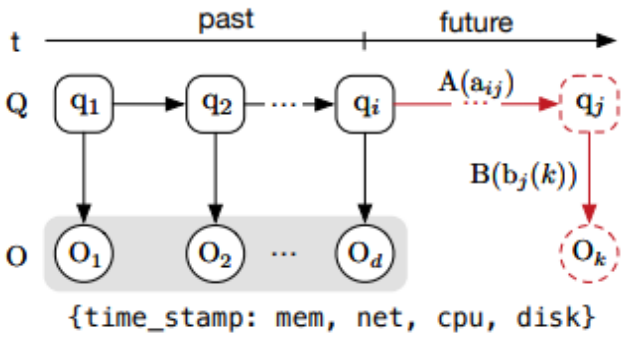

Рисунок 3.4 – Прихована модель Маркова

Ядром SlidHMM є змінне вікно, яке приймає нові спостереження і переміщує застарілі. На етапі навчання SlidHMM застарілі спостереження замінюються апроксимацією методом змінного середнього. На відміну від звичайного HMM, SlidHMM поступово оновлюється за допомогою часткового розрахунку у фіксованому вікні спостережень. Таким чином, він покращує ефективність виявлення вузьких місць як у часі, так і в просторі.[14]

Метод 3: пакетний сніффер (Packet Sniffer).

За допомогою програмного забезпечення, такого як переглядач пакетів або, як його ще називають пакетний сніффер, мережеві пакети, що подорожують мережею, можуть бути легко перехоплені або захоплені. Сніффер перехоплює ці пакети, переводячи мережеву карту в режим перемішування, а потім розшифровує їх, і вони можуть бути використані для різних успішних цілей. Пакет, надісланий від одного вузла до іншого (від джерела до одержувача), повинен пройти через багато проміжних вузлів. Вузол, мережева карта якого налаштована у режимі перемішування, як правило, отримує пакет (рисунок 3.5).

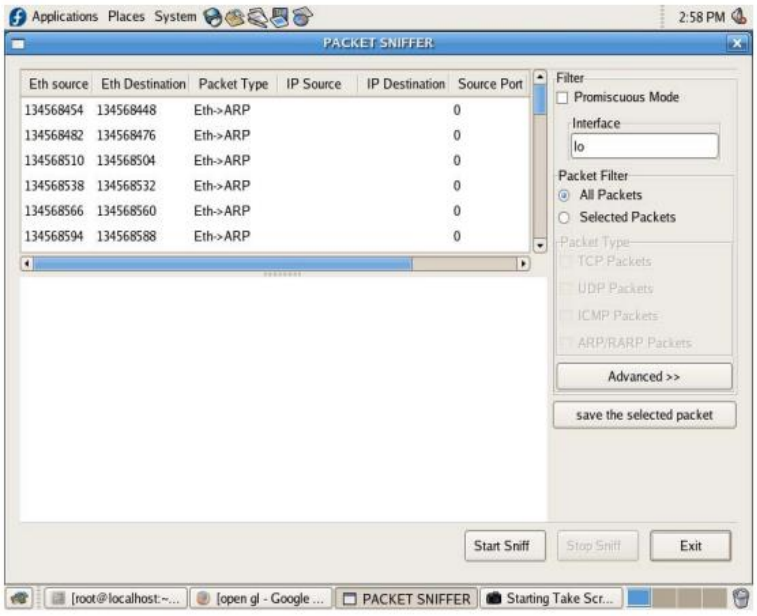

Рисунок 3.5 – Sniffer пакетів під час захоплення сесії

Коли пакети надходять до мережевого адаптера, вони копіюються в пам'ять драйвера пристрою, а потім передаються в буфер ядра. З ядра вони використовуються користувацьким агентом. У ядрі Linux libpcap використовує сокет "PF_PACKET", який обходить більшу частину обробки пакетного протоколу, що виконується ядром[8]. Кожен сокет має два пов'язані буфери ядра для читання і запису. У Fedora 6 розмір кожного буфера за замовчуванням становить 109 568 байт. У нашому сніфері пакетів, коли створюється сеанс перехоплення у реальному часі, пакети копіюються з буфера ядра до буфера, створеного libpcap. Буфер обробляє лише один пакет за раз для обробки додатком, перш ніж буде скопійовано наступний пакет. Новий підхід, застосований при розробці нашого сніффера пакетів, полягає у підвищенні продуктивності сніффера пакетів за рахунок того, що libpcap використовує той самий буферний простір між простором ядра і додатком (рисунок 3.6).[8]

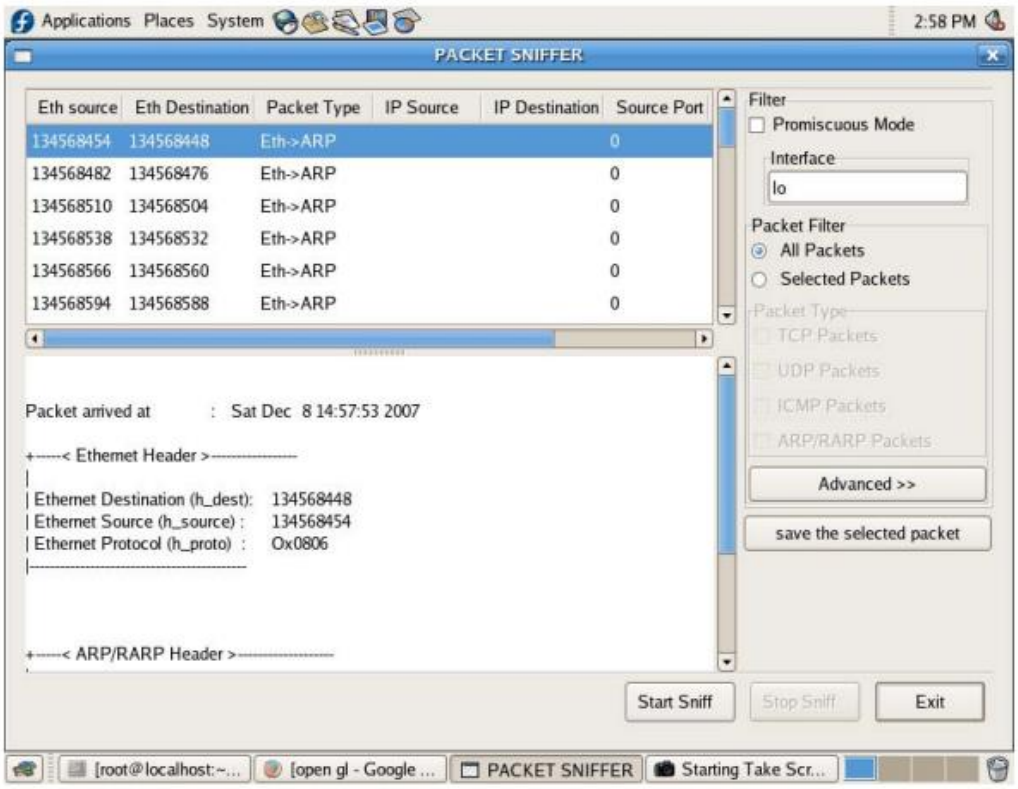

Рисунок 3.6 – Деталі вибраного пакета у пакетному снаффері

Ця операція, безсумнівно, є потенційною загрозою безпеки, тому виклик дозволений тільки для кореневого користувача. Якщо припустити, що "sock" містить вже відкритий сокет, наведені нижче інструкції зроблять свою справу:

```
 (sock, SIOCGIFFLAGS and ether);
етрек.ifr_flags | = IFF_PROMISC;
ioctl (sock, SIOCGIFFLAGS and ether);
```

Перший ioctl зчитує поточне значення прапорців з карти Ethernet; прапори є ORed з IFF_PROMISC, який активує нерозбірливий в засобах режим, і записується назад на карту з другим ioctl.

4 РОЗРОБКА ТА ВПРОВАДЖЕННЯ СЦЕНАРІЇВ ПОШУКУ

4.1 Java-інструменти для імітаційного моделювання

Використання методів моделювання для оцінки продуктивності комп'ютерних та комунікаційних систем є добре відомою методологією для планування, координації, оптимізації та дослідження корисності потужності. Цей процес вимагає використання аналітичних або стимулюючих методів, які часто базуються на моделях мереж масового обслуговування. Незважаючи на значні дослідницькі зусилля, присвячені ефективному розв'язанню цих моделей у минулому, існує дуже мало безкоштовних пакетів Java з відкритим вихідним кодом, які дозволяють проводити комплексні дослідження з оцінки продуктивності на основі мереж масового обслуговування та характеризувати робоче навантаження на етапах моделювання.

Java Modeling Tools (JMT) являє собою набір інструментів для оцінки продуктивності, планування продуктивності і характеризації робочого навантаження комп'ютерних і комунікаційних систем, який був вперше опублікований в 2006 році під відкритою ліцензією. Існує декілька сучасних алгоритмів для точного, наближеного та асимптотичного аналізу мереж масового обслуговування (QN), стохастичних мереж Петрі (SPN) та гібридних моделей, що поєднують в собі елементи обох. Користувачі можуть визначати та аналізувати моделі за допомогою розширеного графічного інтерфейсу або керованих майстрів, що робить інструмент особливо привабливим для оцінювання роботи викладачів, для моделювання продуктивності в промисловості та для студентів, які досліджують експерименти з класичними формалізмами моделювання продуктивності. JMT, зокрема, складається з шести інструментів, найпопулярнішими з яких є JSIMgraph і JSIMwiz, інструменти для моделювання дискретних подій, та

JMVA, аналітичний розв'язувач. JSIMgraph та JSIMwiz надають графічні інтерфейси та інтерфейс на основі майстра для JSIM, механізму моделювання дискретних подій JMT. Вони надають візуальну можливість задавати мережі черг і мережі Петрі, налаштовувати симуляції та перевіряти результати аналізу (середні значення, розподіли, детальні записи). JSIM представляє евристики для фільтрації перехідних процесів, які прискорюють обчислення стаціонарних оцінок та довірчих інтервалів, таких як MSER-5 та спектральний аналіз [5]. Натомість JMVA є аналітичним розв'язувачем для мереж у формі продукту. На додаток, до точних методів розв'язання, таких як аналіз середнього значення, JMVA пропонує наближені розв'язувачі, засновані на ітераціях з фіксованою точкою (наприклад, Bard-Schweitzer, Linearizer) та нормалізації алгоритму на основі констант (наприклад, RECAL, CoMoM). Інструмент також підтримує залежні від навантаження робочі станції, такі як багатосерверні черги та аналіз "що, якщо". Інші додатки в рамках JMT – це JABA, аналітичний вирішувач для аналізу вузьких місць в мережевих чергах у вигляді продукту з трьома класами обслуговування, JMCH, візуальний симулятор ланцюга Markov та JWAT, інструмент для аналізу робочого навантаження на основі зовнішніх файлів даних, включаючи кластерний аналіз та статистичний аналіз робочого навантаження.

Відмінною рисою JMT є те, що його дизайн покращив зручність і доступність інструменту завдяки широкому графічному інтерфейсу користувача, а не розкриттю технічних деталей. Зокрема, інструмент може проводити користувачів через прості інтерфейси майстрів. Крім того, параметризація експериментів вимагає мінімальної взаємодії з користувачем. Таким чином, JMT приховує складність реалізації основних алгоритмів, значно скорочуючи криву навчання для недосвідчених користувачів. Ця особливість також робить інструмент цікавим для освітніх цілей. Однак спрощення інтерфейсу не знижує технічний рівень інструменту. JMT реалізує найсучасніші методи моделювання дискретних подій та аналітичного

оцінювання моделей масового обслуговування, про що ми розповімо в наступних частинах цієї статті. Інтеграція різних додатків, що входять до складу інструменту, а також зв'язок між графічним інтерфейсом користувача і алгоритмами, що лежать в його основі, базується на XML. Це, разом з моделлю розробки з відкритим вихідним кодом, дозволяє легко поєднувати алгоритми JMT із зовнішнім програмним забезпеченням. Наприклад, OPEDo використовує аналітичні алгоритми JMT для оптимізації моделей черг. Крім того, JMT може виконувати параметричний аналіз "що, якщо", який корисний для оцінки чутливості розрахунків продуктивності до змін у властивостях моделі (рисунок 4.1).

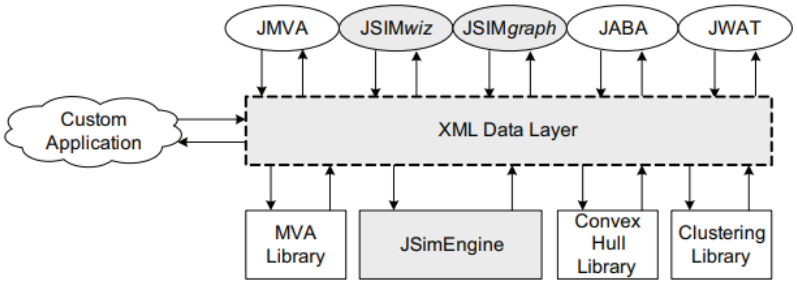

Рисунок 4.1 – Архітектура JMT

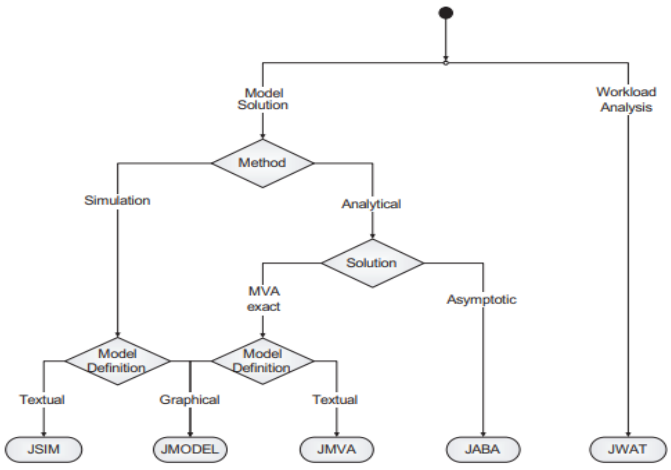

Рисунок 4.2 – Організація JMT Suite

Архітектура JMT складається з набору графічних застосунків у вільному з'єднанні, які взаємодіють через XML з центральним алгоритмічним модулем, що складається з двигуна моделювання (JSIMengine) та бібліотеки аналітичних функцій (рисунок 4.2).

Архітектура має наступні елементи:

- JSIM – симулятор дискретних подій для аналізу моделей мереж масового обслуговування. Інтуїтивно зрозуміла послідовність вікон майстра допоможе вам налаштувати властивості мережі. Механізм моделювання підтримує різні розподіли ймовірностей для опису часу між прибуттями та обслуговуванням, наприклад, експоненціальний, гіпер-експоненціальний, рівномірний, Erlang та Pareto. Генерація випадкових чисел базується на двигуні Mersenne Twister. Також можна відтворити певну послідовність випадкових чисел для імітації реальних навантажень із зібраних лог-файлів. Політики, що залежать від навантаження, можуть бути задані будь-якою функцією від поточної довжини черги. JSIM підтримує незалежні від стану політики маршрутизації, наприклад, Markovan або round robin, а також стратегії, залежні від стану. Наприклад, маршрутизація до сервера з мінімальним навантаженням, або з найкоротшим часом відгуку, або з мінімальною довжиною черги. Оцінюються такі показники продуктивності, як пропускна здатність, навантаження, час відгуку, час простою і довжина черги. Механізм імітаційного моделювання підтримує кілька розширених функцій, які не дозволені в моделях форми продукту, а саме: діапазони кінцевої пропускної здатності (тобто блокування), сервери вузлів розгалужень і класи пріоритетів. Регіони з обмеженою пропускною спроможністю можуть включати спільні або специфічні для класів обмеження на чисельність споживачів. Для аналізу результатів моделювання використовуються онлайн-методи отримання даних про перехідні процеси, засновані на спектральному аналізі. Також можливий імітаційний аналіз, в якому виконується послідовність симуляцій для різних значень параметрів;

- JMODEL – простий у використанні графічний інтерфейс для механізму моделювання, що використовується в JSIM. Він включає ті ж функції JSIM з інтуїтивно зрозумілим графічним робочим простором, як показано на рисунку 4.3. Це дозволяє просто описати структуру мережі, а також спростити визначення функцій виконання, таких як блокування ділянок. Крім того, топологію мережі можна зберегти для подальшого використання, наприклад, для демонстрації в класі або домашнього завдання (рисунок 4.3);

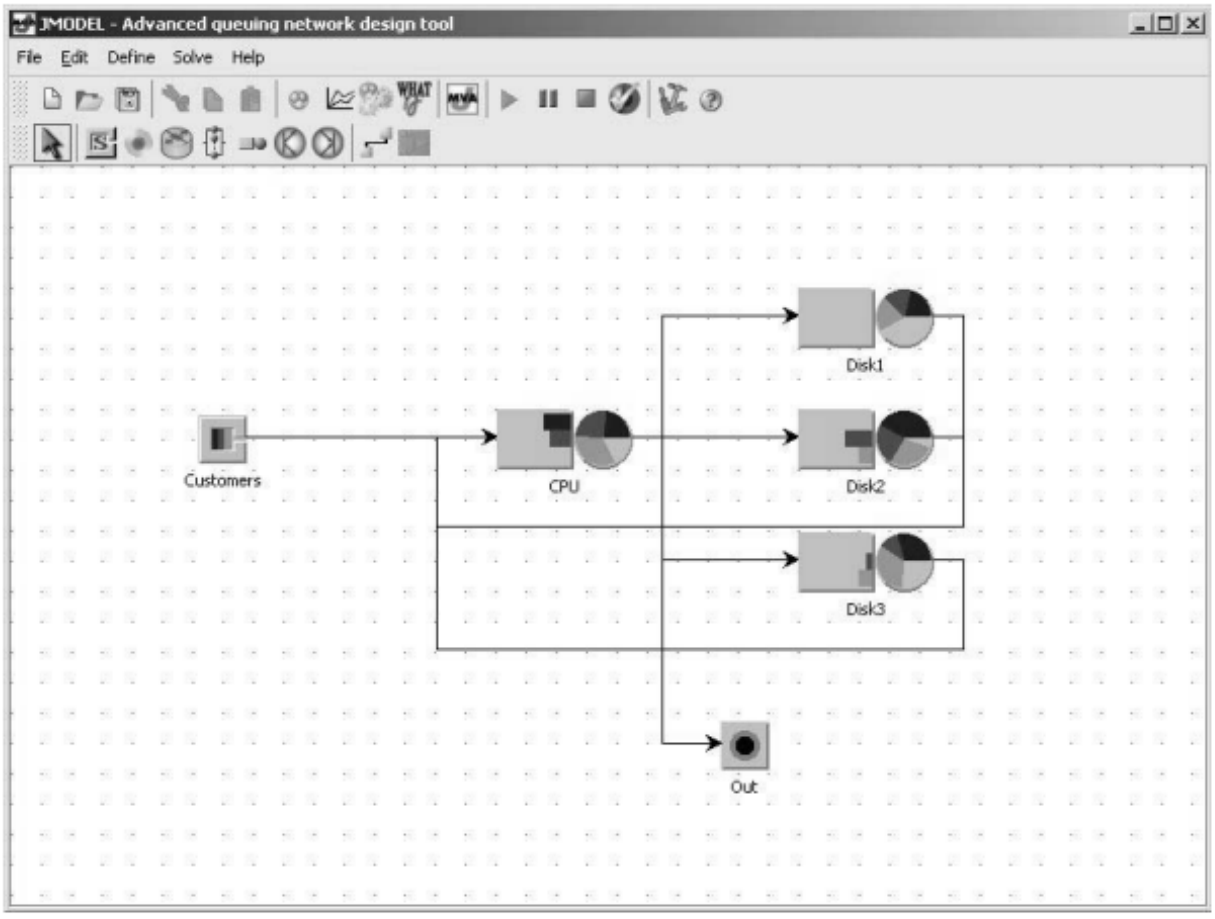

Рисунок 4.3 – Графічна модель JMODEL

- JMVA – призначений для точного аналізу одно- або багатокласових моделей мереж масового обслуговування у вигляді продукту, які обробляють відкриті, закриті або змішані робочі навантаження. Використовується

класичний алгоритм рішення MVA. Структура мережі визначається текстовими майстрами з функціями перетворення ймовірностей в середні показники відвідуваності (і навпаки). Розраховані показники ефективності такі ж, як і для JSIM. Допускається аналіз "що, якщо";

- JABA: інструмент для виявлення вузьких місць у закритих мережах у вигляді продукту за допомогою ефективних алгоритмів опуклого шару. Інструмент підтримує три моделі класів робіт. Можливі вузькі місця можуть бути визначені відповідно до різних комбінацій класів замовників. Можна ефективно аналізувати моделі з тисячами черг. Визначаються сектори насичення, тобто поєднання класів замовників, які насичують більше одного ресурсу одночасно;

- JWAT – Підтримує фазу характеристики робочого навантаження з акцентом на даних веб-журналу. Для вхідних файлів передбачені деякі стандартні формати, такі як HTTP-файли журналів Apache, а також можуть бути вказані користувацькі формати. Імпортовані дані можна спочатку проаналізувати для отримання одновимірних або багатовимірних даних за допомогою описових статистичних методів (наприклад, середні значення, кореляції, гістограми PDF, секторні діаграми, діаграми розсіювання). Ці методи дозволяють визначити центри тяжіння кластерів, а потім оцінити середнє робоче навантаження і вимоги до обслуговування, які будуть використані для параметризації моделі. Інструмент також містить інтерфейс до інструменту CLUTO Similarity Clustering.

4.2 Java-інструменти для розв'язку лінійних рівнянь

Розв'язування систем лінійних рівнянь є поширеною проблемою в комп'ютерних науках. Хоча перші розв'язки цієї задачі були знайдені ще в античні часи (наприклад, метод виключення Гауса був відкритий в Китаї близько 2000 років тому), в Європу він прийшов завдяки книгам з алгебри Ісаака Ньютона. У 1969 році Volker Strassen знайшов алгоритм множення

шляхом ділення і правила, швидший за метод виключення Гауса (часова складність якого становить O (n2.81)) при розв'язуванні систем лінійних рівнянь за рахунок швидкого знаходження оберненої величини. Це відкриття значно розширило інтерес до проблеми і стало великим науковим проривом у цій галузі. Відтоді математики працюють над пошуком та оптимізацією нових алгоритмів для цієї задачі, адже на великих матрицях невелика зміна швидкості роботи алгоритму може значно підвищити ефективність.

Системи лінійних рівнянь зустрічаються майже у всіх видах інженерних і наукових застосувань чисельних обчислень. Існує два методи розв'язання таких систем: прямі методи, засновані на розкладанні матриць на множники, та ітераційні методи. JAMA – базовий пакет лінійної алгебри для Java. Надає користувацькі класи для побудови та маніпулювання щільними та дійсними масивами. Він повинен пропонувати достатню функціональність для рутинних задач, природно упакований і зрозумілий для неспеціалістів. Він має слугувати стандартним класом масивів для Java і пропонується як такий на форумі Java Grande Forum, а потім у Sun. MathWorks та NIST розробили реалізацію, що є суспільним надбанням і безпосередньо пов'язана з таким класом. Ми випускаємо цю версію для публічного обговорення. Немає жодних гарантій, що майбутні версії JAMA будуть сумісні з нею.

Схожий матричний пакет, Jampack, також був розроблений в NIST та Університеті Меріленда. Обидва пакети народилися з потреби оцінити альтернативні проекти для реалізації масивів на Java. JAMA базується на єдиному класі масиву в строго об'єктно-орієнтованому фреймворку. Jampack використовує більш відкритий підхід, який підходить для розширення користувачем. Виходить, що основна відмінність між пакетами для звичайного користувача полягає в синтаксисі операцій над матрицями.

Клас Matrix надає базові операції цифрової лінійної алгебри. Різноманітні конструктори створюють масиви з двовимірних масивів чисел з плаваючою комою подвійної точності. Різні get та set забезпечують доступ до підмасивів та елементів масиву. Базові арифметичні операції включають

додавання і множення матриць, матричні правила і вибрані поелементні операції над матрицями. Зручний процес друку матриць також включено.

П'ять базових декомпозицій матриць, що складаються з пар або трійок матриць, векторів перестановок і т.п., дають результати у п'яти класах розкладів. До цих розкладів можна отримати доступ за допомогою класу Matrix для обчислення розв'язків одночасних лінійних рівнянь, визначників, обернених та інших матричних функцій.

П'ять класів декомпозицій:

- декомпозиція Холеського за додатньо-визначеними симетричними матрицями;
- LU-декомпозиція (гаусове усунення) прямокутних матриць;
- QR декомпозиція прямокутних матриць;
- декомпозиція власних значень симетричних та несиметричних квадратних матриць;
- декомпозиція сингулярних значень прямокутних матриць.

Поточна версія JAMA має справу лише з дійсними матрицями. Ми сподіваємось, що в майбутніх версіях будуть розглядатись і комплексні матриці. Це було відкладено, оскільки важливі проектні рішення не можуть бути прийняті, поки не будуть вирішені деякі питання, пов'язані з реалізацією комплексів у мові Java.

Розв'язання лінійних рівнянь методом оберненої матриці є складним, коли система має більше 3 рівнянь і 3 невідомі змінні. Тому для розв'язання системи лінійних рівнянь, яка має N лінійних рівнянь і N невідомих змінних, перевагу надають методу Гауссового виключення.

Гаусове виключення проводиться в два етапи:

- верхня трикутна матриця;
- підстановка назад.

Ці два етапи методу Гауссового виключення відрізняються не операціями, які ви можете використовувати на них, а результатом, який вони дають. Крок прямого виключення стосується скорочення рядків, необхідного

для спрощення розглядуваної матриці в її ешелонованій формі. Мета цього кроку – показати, чи має система рівнянь, представлена в матриці, тільки один можливий розв'язок, нескінченну кількість розв'язків або не має жодного. Якщо визначено, що система не має розв'язку, то немає сенсу продовжувати скорочувати рядки матриці до наступного кроку.

Якщо є можливість отримати розв'язки для змінних, що входять до лінійної системи, то Гаусове виключення виконується з кроком зворотної заміни. Цей останній крок створює масштабовану скорочену форму матриці, яка, в свою чергу, забезпечує загальний розв'язок системи лінійних рівнянь.

Правила Гауссового виключення такі ж самі, як і правила трьох елементарних операцій над рядками, іншими словами, ви можете алгебраїчно оперувати рядками матриці наступними трьома способами:

- поміняйте два рядки;
- помножте рядок на константу (будь-яка ненульова константа);
- додайте рядок в інший рядок.

Отже, розв'язання лінійної системи з матрицями за допомогою Гаусового виключення виявляється структурованим, організованим і досить ефективним методом.

Якщо у вас є наступна система лінійних рівнянь, що містить три рівняння для трьох невідомих:

Крок 1. Перетворення лінійної системи в доповнену матрицю:

$$\begin{aligned} &x + y + z = 3 \\ &x + 2y + 3z = 0 \\ &x + 3y + 2z = 3 \end{aligned} \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 1 & 2 & 3 & 0 \\ 1 & 3 & 2 & 3 \end{array}\right]$$

Крок 2. Проведемо скорочення рядків (скористаємося Гаусовим виключенням), щоб спростити матрицю:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 2 & 4 & -6 \\ 0 & 0 & -3 & 6 \end{array}\right) \xrightarrow{R_2 - R_1 \to R_2} \left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 1 & 2 & -3 \\ 1 & 3 & 2 & 3 \end{array}\right) \xrightarrow{R_3 - R_1 \to R_3}$$

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 1 & 2 & -3 \\ 0 & 2 & 1 & 0 \end{array}\right) \xrightarrow{2R_2 \to R_2} \left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 2 & 4 & -3 \\ 0 & 2 & 1 & 3 \end{array}\right) \xrightarrow{R_3 - R_2 \to R_3}$$

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 2 & 4 & -6 \\ 0 & 0 & -3 & 6 \end{array}\right) \qquad \begin{array}{l} x + y + z = 3 \\ 2y + 4z = -6 \\ -3z = 6 \end{array}$$

Крок 3. Отримання матриці:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 2 & 4 & -6 \\ 0 & 0 & -3 & 6 \end{array}\right)$$

Зауважте, що на цьому етапі ми бачимо, що ця система лінійних рівнянь може бути розв'язана з унікальним рішенням для кожної змінної. Те, що ми вже зробили, є першим кроком до скорочення черг: пряме виключення. Ми все ще можемо спростити цю матрицю (що поверне нас до другого кроку підстановки), але це не є необхідним, оскільки на цьому етапі система може бути легко розв'язана.

Отже, ми розглянемо отриману систему, щоб вирішити її безпосередньо:

$$\begin{array}{c} x + y + z = 3 \\ 2y + 4z = -6 \\ -3z = 6 \end{array}$$

З цього набору, ми можемо автоматично спостерігати, що значення змінної z є: z = -2. Ми використовуємо це знання, щоб підставити його в другому рівнянні, щоб вирішити y, і підстановка значень y і z в перших рівняннях, щоб вирішити x:

$$\text{Маємо: } z = -2$$

$$2y + 4z = -6 \rightarrow 2y + 4(-2) = 2y - 8 = -6$$

$$2y = 2 \rightarrow y = 1.$$

Обчислення значення x:

$$x + y + z = 3 \rightarrow x + (1) + (-2) = x - 1 = 3 \rightarrow x = 4.$$

І кінцевим рішенням системи є:

$$x = 4, y = 1, z = -2.$$

4.3 Розробка та впровадження сценаріїв пошуку

Після глибокого аналізу вузьких місць в системах масового обслуговування і комп'ютерних мережах, практичний розділ присвячений програмі і реалізації Java-коду, який аналізує даний сценарій графа, а потім повертає вузьке місце вузлів і найбільш зручні шляхи між вузлами.

Як описано в попередніх підрозділах вище, програма java використовує два алгоритми «Dijkstra» і «Gaussian elimination».

Для того, щоб реалізувати сценарії пошуку, щоб знайти вузькі місця, є кілька кроків, яких слід дотримуватися:

- знаходження лінійних рівнянь пропускних здатностей Xi;
- розрахунок використання Ui;
- пошук вузлів вузького місця;
- виведіть надійний шлях між заданими вузлами.

Алгоритм Dijkstra розв'язує задачу пошуку найкоротшого шляху від точки на графі (джерела) до пункту призначення. Ви можете знайти найкоротші шляхи від заданого джерела до всіх точок графа одночасно, тому цю задачу іноді називають задачею про найкоротші шляхи з одним джерелом. Ця стаття допоможе вам зрозуміти концепції, що лежать в основі алгоритму Dijkstra, використовуючи прості, зрозумілі приклади та ілюстрації.

Щоб більш детально зрозуміти, як працює програма, я проаналізую систему відкритої черги і виконаю кроки, зазначені вище.

На наведеному графі показана пряма відкрита система масового обслуговування, в одному напрямку, з 7 вузлами і 15 вершинами (рисунок 4.4).

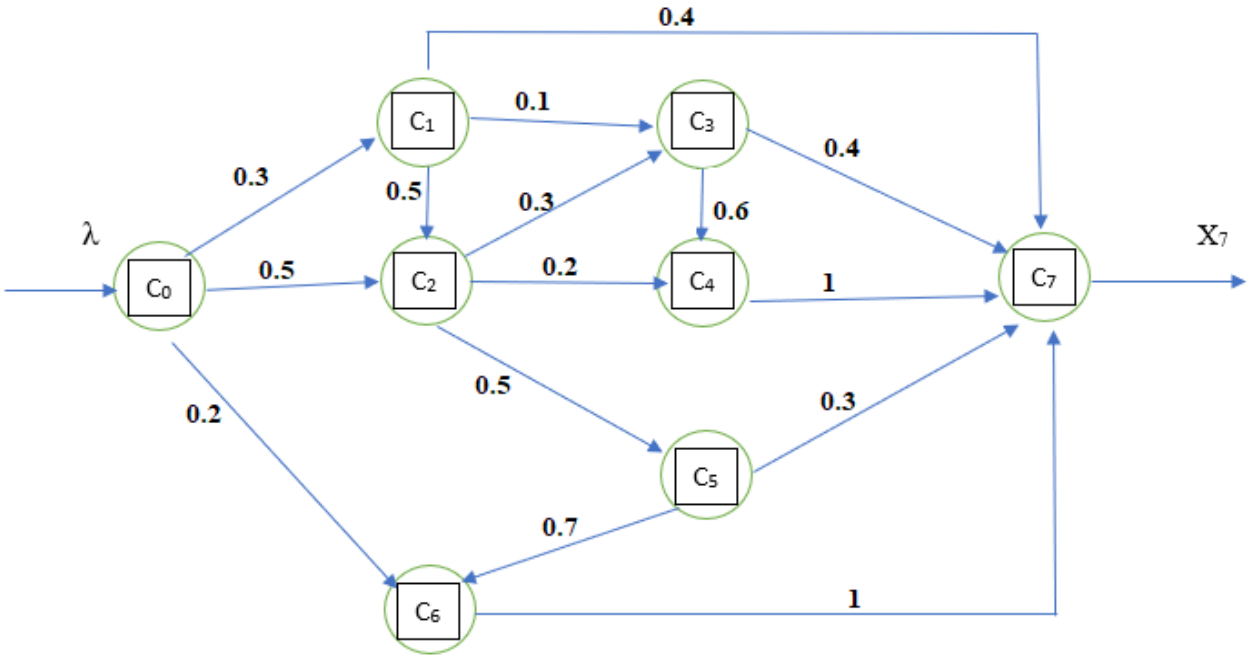

Рисунок 4.4 – Граф

Java код розглядає збалансовані за потоком системи, це означає, що кількість прибулих на даному пристрої повинна бути (майже) такою ж, як кількість виходів з цього пристрою в період спостереження.

4.3.1 Операційні змінні

Операційна змінна – це формальний символ, який позначає значення деякої величини, яку можна виміряти протягом періоду спостереження. Вона має єдине, конкретне значення для кожного періоду спостереження. Операційні змінні бувають або незалежними (базові величини), які безпосередньо вимірюються протягом періоду спостереження, або залежними (похідні величини), які обчислюються на основі базових величин.

У нашій задачі першим кроком, який ми запрограмували, є функція, яка відповідає за зчитування вхідних даних користувача та формування матриці для обчислення пропускної здатності кожного вузла Xi, а потім коефіцієнта використання Ui.

Лістінг 4.1 – readMatrixByUser

```
public void readMatrixByUser() {
    int i, j;
    Scanner in = null;
    try {
        in = new Scanner(System.in);
        System.out.println("enter arrival rate λ");
        lambda = in.nextInt();
        System.out.println("enter number of equations");
        m = in.nextInt();
        System.out.println("enter number of unknowns");
        n = in.nextInt();
        A = new double[m][n];
        C = new double[m][n];
        System.out.println("Enter the elements of the matrix");
        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++) {
                A[j][i] = in.nextDouble();
                C[i][j] = A[j][i];
                if (i == j) {
                    if (j == 0)
                        A[j][i] = 1.0;
                    else
                        A[j][i] = -1.0;
                }
            }
        }
```

```
            B = new double[n];
            B[0] = lambda;
            for (int k = 1; k < n; k++)
                B[k] = 0;
        } catch (Exception e) {
        }
}
```

Спочатку користувач вводить значення лямбда, кількість рівнянь та невідомих у цілочисельному форматі. У нашому прикладі λ = 35 запитів/сек і є вісім рівнянь та невідомих (X0,....X7).

Оскільки ми працюємо зі збалансованою системою, це означає, що λ = X0. Рівняння балансу потоку будуть наступними:

$$Xj = \sum_{i=0}^{k} Xiqij, j = 0, \dots, k$$

.

Якщо ми застосуємо це математичне відношення в нашій системі, ми отримаємо вісім рівнянь нижче:

$$X_0 = \lambda = 35, \quad (4.1)$$

$$X_1 = 0.3*X_0, \quad (4.2)$$

$$X_2 = 0.5*X_0 + 0.5*X_1, \quad (4.3)$$

$$X_3 = 0.1*X_1 + 0.3*X_2, \quad (4.4)$$

$$X_4 = 0.2*X_2 + 0.6*X_3 , \quad (4.5)$$

$$X_5 = 0.5*X_2 , \quad (4.6)$$

$$X_6 = 0.2*X_0 + 0.7*X_5, \quad (4.7)$$

$$X_7 = 0.4*X_1 + 0.4*X_3 + 1*X_4 + 0.3*X_5 + 1* X_6. \quad (4.8)$$

Лістінг 4.2 – Функція solve

```
public double[] solve(double[][] A, double[] B) {
        for (int p = 0; p < n; p++) {

            int max = p;
            for (int i = p + 1; i < m; i++) {
                if (Math.abs(A[i][p]) > Math.abs(A[max][p])) {
                    max = i;
                }
            }
            double[] temp = A[p];
            A[p] = A[max];
            A[max] = temp;
            double t = B[p];
            B[p] = B[max];
            B[max] = t;
            if (Math.abs(A[p][p]) <= EPS) {
                throw new RuntimeException("The Matrix is
singular or nearly singular");
            }

            for (int i = p + 1; i < m; i++) {
                double alpha = A[i][p] / A[p][p];
                B[i] -= alpha * B[p];
                for (int j = p; j < n; j++) {
                    A[i][j] -= alpha * A[p][j];
                }
            }
        }

        x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double sum = 0.0;
            for (int j = i + 1; j < n; j++) {
                sum += A[i][j] * x[j];
            }
            x[i] = (B[i] - sum) / A[i][i];
        }

        for (int i = 0; i < n; i++)
            System.out.println("X" + i + " = " +String.
format("%.4f", x[i]));

        return x;
    }
```

Оскільки тепер за введеними даними ми обчислюємо пропускну здатність, то для розв'язання цих лінійних рівнянь використовується алгоритм Гауса.

Метод "solve (A, B)" – це функція пошуку розв'язків системи одночасних лінійних рівнянь, яка полягає в тому, що спочатку розв'язується одне з рівнянь зі змінною, а потім підставляється цей член в решту рівнянь. Результатом є нова система, в якій кількість рівнянь і змінних на одиницю менша, ніж у вихідній системі. Таку саму процедуру застосовують до іншої змінної, і процес скорочення триває доти, доки не буде отримано рівняння, в якому єдиною невідомою є остання змінна. Розв'язання цього рівняння дозволяє "підставити" це значення в попереднє рівняння, яке містить цю змінну і невідому для іншої змінної. Цей процес триває доти, доки не будуть обчислені всі вихідні змінні.

Якщо матриця є одниною (вона не має оберненої), буде виведено повідомлення про помилку, що означає, що матриця є одниною і не має рішення.

Коефіцієнти кожного рівняння будуть представлені у вигляді рядка в матриці розміром n*m. Користувач вводить елементи матриці, після чого програма виконує обчислення і виводить результати, як показано на рисунку (рисунок 4.5).

```
*****************Calculate the Throughput Xi***********************
enter arrival rate λ
35
enter number of equations
8
enter number of unknowns
8
Enter the elements of the matrix
0 0.3 0.5 0 0 0 0.2 0
0 0 0.5 0.1 0 0 0 0.4
0 0 0 0.3 0.2 0.5 0 0
0 0 0 0 0.6 0 0 0.4
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0.7 0.3
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
X0 = 35.0000
X1 = 10.5000
X2 = 22.7500
X3 = 7.8750
X4 = 9.2750
X5 = 11.3750
X6 = 14.9625
X7 = 35.0000
```

Рисунок 4.5 – Розрахунок пропускної здатності Xi

4.3.2 Пошук вузького місця між двома вибраними вузлами

Як ми вже згадували в теоретичній частині, існує багато методів пошуку вузького місця в системі масового обслуговування. Відповідно до наданої інформації ми вирішили використати метод утилізації, який буде перевіряти завантаженість кожного вузла, якщо він належить до інтервалу [0.6, 0.72], що означає, що вузол в середньому завантажений на 60% – 72%.

Ми маємо пропускну спроможність, що зберігається в масиві, тому ми можемо обчислити коефіцієнти використання, застосувавши співвідношення Ui = Si * Xi, після чого ми можемо знайти вузькі місця в системі масового обслуговування. Вузьким вважається той вузол, значення використання якого знаходиться в інтервалі [0.6, 0.72].

Лістінг 4.3 – Функція findBottleneck

```
public int findBottleneck(int node1 , int node2 ,
ArrayList<Double>list)
    {
        S = new double[n];
        U = new double[n];
        for(int i=0;i<list.size();i++)
        {
            S[i] = list.get(i);
            U[i] = S[i]*x[i];
        }
        for(int j=node1;j<=node2;j++)
        {
            if(U[j]>=0.6 && U[j]<=0.72) {
                b_Index = j;
            }
        }
        return b_Index;
    }
```

Метод findBottleneck отримує індекс двох вузлів і масив, в якому будуть зберігатися значення середнього сервісу Si, які будуть надані користувачем. Алгоритм цієї функції дійсно базовий і нічого складного в ньому немає. Спочатку оголошуються два масиви, один з яких буде містити

послідовно значення Si, а інший – утиліти. Перший цикл for відповідає за заповнення порожніх двох масивів S[] та U[], другий for шукає вузьке місце між node1 та node2 і зберігає його індекс у змінній b_index та повертає його як ціле число.

Користувач вводить середні значення обслуговування на консолі, після чого програма виконує обчислення і порівняння, щоб знайти вузьке місце між двома заданими вузлами (рисунок 4.6).

```
*****************Find Bottleneck node**********************
enter Si values:
0.015 0.045 0.03 0.04 0.05 0.027 0.04 0.01
node1:
1
node2:
5
The bottleneck node is: 2
```

Рисунок 4.6 – Пошук вузького місця в системі черги

Як показано на рисунку 4.6, користувач ввів значення Si в одному рядку і в двох вузлах.

Коли функція findBottleneck прочитає дані, наведені користувачем, ми отримаємо наступні значення використання:

$U_0 = 0.525$;

$U_1 = 0.4725$;

$U_2 = 0.6825$;

$U_3 = 0.315$;

$U_4 = 0.46375$;

$U_5 = 0.307$;

$U_6 = 0.5985$;

$U_7 = 0.35$;

Якщо ми застосуємо метод утилізації, очевидно, що вузол 2 є вузьким місцем, тому що $U_2 = 0.6825$ і те, що ми бачимо в результаті на консолі.

4.3.3 Пошук надійного шляху між двома вузлами

Тепер, коли ми знаємо вузьке місце, наше завдання – знайти найбільш надійний шлях між двома вузлами, щоб уникнути затримок і низької пропускної здатності. Щоб знайти цей шлях, ми використовуємо алгоритм Dijkstra.

Алгоритм виконує ряд кроків для знаходження найзручнішого шляху:

а) створіть множину "відвіданих", яка відстежує вершини, що містяться у дереві найкоротшого шляху, обчислює і заповнює мінімальну відстань від джерела. Спочатку це речення пусте;

б) присвоїти значення відстаней усім вершинам вхідного графа. Ініціалізує всі значення відстані як INFINITE. Присвоює початковій вершині значення відстані 0, тому вона вибирається першою;

в) хоча він і відвідуваний, але не включає в себе всі вершини;

1) виберіть вузол u, якого не існує у відвіданому вузлі й має мінімальне значення відстані;

2) додати u в «відвідані»;

3) оновити значення відстані до всіх сусідніх вершин від u, щоб оновити значення відстані, прокрутіть всі сусідні вершини. Для кожної сусідньої вершини v оновити значення відстані до v, якщо сума значення відстані до u (від джерела) та ваги ребра u-v менша за значення відстані до v, і зберегти шлях у масив батьків.

У наведеному вище розділі ми отримали покроковий процес алгоритму Dijkstra, тепер давайте розглянемо алгоритм на нашому прикладі.

Лістінг 4.4 – Функція dijkstra_Algorithm

```
void dijkstra_Algorithm(double graph[][], int src, int dest) {
        double dist[] = new double[n];
        int[] parents = new int[n];
        parents[src] = -1;
        Boolean visited[] = new Boolean[n];
        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }


dist[src] = 0;
        for (int count = 0; count < n - 1; count++) {
            double u = minimumDistance(dist, visited);
            visited[(int) u] = true;
            for (int v = 0; v < n; v++)
                if (!visited[v] && graph[(int) u][v] != 0 &&
                        dist[(int) u] != Integer.MAX_VALUE &&
                        dist[(int) u] + graph[(int) u][v] <
dist[v]) {
                    parents[v] = (int) u;
                    dist[v] = dist[(int) u] + graph[(int) u][v];
                }
        }

        print(dist, src, dest, parents);
    }
```

Ми обчислимо найкоротший шлях між вузлом 0 і вузлом 7 в графі.

Для того, щоб більш чітко зрозуміти алгоритм Dijkstra, ми будемо використовувати табличний калькулятор, який метод створює неявно.

Таблиця 4.1 – Таблиця Dijkstra Калькулятор

Visited	0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8	9
0	0	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
1	-	0.525	0.525	Infinity	Infinity	Infinity	0.525	Infinity
2	-	-	0.525	0.9975	Infinity	Infinity	0.525	0.9975

Продовження таблиці 4.1

1	2	3	4	5	6	7	8	9
6	-	-	-	0.9975	1.2075	1.2075	0.525	0.9975
3	-	-	-	0.9975	1.2075	1.2075	-	0.9975
7	-	-	-	-	1.2075	1.2075	-	0.9975
4	-	-	-	-	1.2075	1.2075	-	-
5	-	-	-	-	-	1.2075	-	-

Алгоритм дій.

Крок 1. Під час виконання алгоритму кожен вузол буде позначено мінімальною відстанню до вузла 0, оскільки ми вибрали вузол 0.

У цьому випадку мінімальна відстань дорівнює 0 для вузла 0. Крім того, для решти вузлів, оскільки ми не знаємо цієї відстані, вони будуть позначені як нескінченність (∞), за винятком вузла 0 (на даний момент позначено як виконане).

Крок 2. Тепер буде перевірено сусіди вузла 0, тобто вузла 1, 2 і 6. Почнемо з вузла 1, тут ми додамо мінімальну відстань поточного вузла (0) з вагою ребра (0.525), яка пов'язала вузол 0 з вузлом 1 і отримає 0.525.

Тепер, це значення буде порівнюватися з мінімальною відстанню вузла 1 (нескінченність), найменше значення – це те, що залишається мінімальною відстанню вузла 1, як в цьому випадку, 0.525 менше нескінченності, і позначає найменше значення для вузла 1.

Крок 3. Тепер той же процес перевіряється сусіднім вузлом 2 і отримує 0.525. Знову ж таки, 0.525 порівнюється з мінімальною відстанню вузла 2 (нескінченність), і позначаємо найменше значення.

Те ж саме повторюється з вузлом 6 і позначається 0.525 як найменше значення на вузлі 6.

Оскільки, всі сусіди вузла 0 перевіряли, отже вузол 0 позначено як відвіданий.

Крок 4. Тепер ми перейдемо до поточного вузла, що вузол ще не відвіданий і з найменшою мінімальною відстанню. Тут, вузол 1 є невідвіданим з мінімальною відстанню 0.525, позначеним як поточний вузол.

Повторюємо алгоритм, перевіряючи сусіда поточного вузла при ігноруванні відвіданого, тому вузли 2, 3 і 7 будуть перевірені.

Крок 5. Після цього вузол 1 позначено як відвіданий. Наступним вибраним вузлом є вузол 2, його не відвідано і він має найменшу останню відстань. Ми повторюємо алгоритм і перевіряємо вузли 3, 4 і 5. Ми порівнюємо поточне значення з мінімальною відстанню кожного вузла, і відзначаємо найменше значення. Вузол 2 позначено як відвіданий.

Алгоритм буде повторюватися і вибирати мінімальне використання вузла, поки всі вузли графа не будуть переглянуті (рисунок 4.7).

Як показано на рисунку 4.6 надійний шлях між вузлом

0 і 7 є 0 -> 1 -> 7.

```
*****************Find the reliable path**********************
source node:
0
destination node:
7
The distance from node 0 to node 7 is: 0.9975 | Shortest Path: 0 1 7
```

Рисунок 4.7 – Пошук надійного шляху між двома вузлами

4.3.4 Пошук вузьких місць на множині вузлів

Остання частина полягає в знаходженні вузьких місць в наборі даних вузлів. Користувач вводить набір утилізацій вузлів і метод «subSetBottlenecks()» приймає вхідні дані як аргумент і порівнює кожне значення, якщо воно належить до інтервалу [0.6,0.72].

Метод приймає arraylist як аргумент, який буде містити значення застосувань, наведених користувачем. Потім програма друкуватиме

налаштування вузьких вузлів в порядку, в якому вони набираються, порівнюючи їх значення з інтервалом [0.6,0.72] (рисунок 4.8).

Лістінг 4.5 – Функція subSetBottlenecks

```
public void subSetBottlenecks(List<Double> util)
    {
        System.out.print("The bottleneck nodes are: ");
        for(int i=0;i<util.size();i++)
            if(util.get(i)>=0.6 && util.get(i)<=0.72)
                System.out.print(util.get(i)+" ");

        System.out.println();
    }
```

```
*****************Find the bottleneck set**********************
enter the utilizaton values:
0.4 0.6 0.68 0.3 0.72 0.8
The bottleneck nodes are: 0.6 0.68 0.72
```

Рисунок 4.8 – Пошук вузьких точок підмережі

Як висновок до практичної частини, основна мета програми java полягає в тому, щоб допомогти в обчисленні систем лінійних рівнянь і знайти вузькі місця в комп'ютерній мережі. Ця програма є зручною для користувачів, її легко зрозуміти та реалізувати, і вона може бути застосована для різних типів графіків (Directed, Undirected або Weighted Graph).

ВИСНОВОК

У цій статті розглянуто та проаналізовано різноманітні підходи, які досліджують вузькі місця у виробничих мережах. Ми зосередили нашу увагу на трьох аспектах: визначеннях, методах виявлення, а також апроксимації та асимптотичних результатах аналізу на основі вузьких місць.

Ми узагальнили поточні результати з практичної точки зору, з метою надання корисних рекомендацій у застосуванні. Наші спостереження можна підсумувати наступним чином:

- визначення вузьких місць ґрунтується на вподобаннях користувачів, які можна поділити на дві категорії: На основі PIP та на основі чутливості. У першому випадку акцент робиться на продуктивності в реальному часі, а в другому – більше уваги приділяється можливим покращенням.

- методи виявлення – це методи дослідження особливостей для визначення того, які ресурси або утиліти відповідають умовам вузького місця, та які тісно пов'язані з визначеннями. Ані методи виявлення на основі PIP, ані індикатори на основі чутливості не можуть запропонувати універсального рішення. Різні методи виявлення можуть мати дуже різні результати. Метод виявлення для програми слід вибирати відповідно до вимог програми.

- використовуючи поведінку граничного трафіку, аналіз на основі вузьких місць дозволяє швидко проаналізувати властивості системи з меншими обчислювальними зусиллями. Методи апроксимації можуть бути використані для швидкої оцінки продуктивності, а асимптотичні формули можуть допомогти зрозуміти, як системи реагують на зміну параметрів. У комплексних мережах важливіше зрозуміти вплив параметрів, моделювання яких зазвичай вимагає значних обчислювальних затрат, і аналіз вузьких місць є дуже корисним у цьому випадку.

Хоча вузькі місця у виробничих системах вивчаються вже більше десяти років, майбутні напрямки залишаються багатообіцяючими. Конкретизуючи вимоги, можна отримати загальне визначення, а методи виявлення можуть забезпечити більш точне і надійне виявлення завдяки перевагам технології інтелектуального аналізу даних і обчислювальних потужностей. Аналіз на основі перевантажень буде більш важливим для розуміння поведінки складних систем і, ймовірно, поширюється на планування інтернет-трафіку і призначення бездротових каналів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. https://www.diva-portal.org/smash/get/diva2:797718/FULLTEXT01.pdf (англійською мовою)

2. https://www.researchgate.net/publication/326712261_Identification_of_bottlenecks_and_analysis_of_the_state_before_applying_lean_management (англійською мовою)

3. https://www.researchgate.net/publication/328337894_The_Case_Study_of_Bottlenecks_Identification_for_Practical_Implementation_to_the_Theory_of_Constraints (англійською мовою)

4. https://ops.fhwa.dot.gov/publications/fhwahop09042/fhwahop09042.pdf (англійською мовою)

5. Горбачов В.О Лук'янов О.О. Методи пошуку вузьких місць у комп’ютерних мережах. Одинадцята міжнародна науково-технічна конференція ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ, ХНУРЕ, листопад 2023 р.

6. https://www.researchgate.net/publication/2800540_Bottleneck_Analysis_For_Computer_And_Communication_Systems_With_Workload_Variabilities_Uncertainties (англійською мовою)

7. https://core.ac.uk/download/pdf/29402955.pdf (англійською мовою)

8. http://polaris.imag.fr/jonatha.anselmi/anselmi08bottleneck.pdf (англійською мовою)

9. https://www.semanticscholar.org/paper/Bottleneck-analysis-and-traffic-congestion-Qadeer-Khan/b81efa9940a98dbcbbf25490d81dc1c69b89a036 (англійською мовою)

10. https://www.duo.uio.no/bitstream/handle/10852/54073/Thesis_PratikTimalsena.pdf?sequence=1 (англійською мовою)

11. https://hal.archives-ouvertes.fr/inria-00282200/ (англійською мовою)

12. https://connect.ncdot.gov/projects/research/RNAProjDocs/2016-10%20Final%20Report.pdf (англійською мовою)

13. https://www.cs.hmc.edu/~geoff/classes/hmc.cs147.201201/slides/class26_queueingnetworks_beamer.pdf (англійською мовою)

14. https://www.reservoir.com/wp-content/uploads/2021/03/Bottleneck_Structures_High-Precision_Network_Performance_Analysis.pdf (англійською мовою)

15. https://iqua.ece.toronto.edu/papers/hwang-tnse17.pdf (англійською мовою)

16. https://mosimtec.com/5-insightful-bottleneck-analysis-examples/ (англійською мовою)

17. https://sites.google.com/a/hc.books-now.com/en113/9780080491455-14coediGEmohi31 (англійською мовою)

18. https://www.economics.uci.edu/files/docs/workingpapers/2014-15/14-15-06.pdf (англійською мовою)

19. https://itstillworks.com/detect-bottleneck-30031.html(англійською мовою)

20. https://tulip.co/glossary/what-is-bottleneck-analysis-everything-you-should-know/ (англійською мовою)

21. http://www.cse.cuhk.edu.hk/~cslui/CSC5480/akamai-bottlenecks.pdf (англійською мовою)

22. https://math.nist.gov/javanumerics/jama/ (англійською мовою)