Хмельницький національний університет
Факультет інформаційних технології
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ
Назва теми

КвРКІ. 190118.07.07.01 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2–19–1 _____ В. А. Кліпацький
Підпис Ініціали, прізвище

Керівник _____ О. В. Боровик
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С. М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т. О. Говорущенко
Підпис Ініціали, прізвище

« 26 » червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т. О. Говорущенко

"11" січня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Кліпацький Владислав Андрійович

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ

Керівник проекту (роботи) Боровик О. В,, професор кафедри КІІС

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5

2. Строк подання студентом проекту (роботи) на кафедру 17.06.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) концепцію комунікування в режимі реального часу між віртуальними машинами у хмарному середовищі; проєктування системи керування трафіком в реальному часі на двох рівнях між віртуальними машинами; реалізація масштабованого проміжного програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Комунікація на рівні платформи та інфраструктури

Типова архітектура МППЗ

Передача/прийом драйверів в чергу

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прий
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання « 11 » 01 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітк
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	11.01.2023	виконан
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2023	виконан
3	Робота над розділом 1 – Концепція механізмів та способів комунікування в реальному часі на двох рівнях між віртуальними машинами	01.04.2023	виконан
4	Робота над розділом 2 – Система керування трафіком в реальному часі на двох рівнях між віртуальними машинами	01.05.2023	виконан
5	Робота над розділом 3 – Масштабоване проміжне програмне забезпечення	30.04.2023	виконан
6	Оформлення пояснювальної записки згідно вимог	25.05.2023	виконан
7	Попередній захист ВКР	26.05.2023	виконан
8	Захист ВКР на засіданні ЕК	Червень 2023 року	

Студент Підпис Кліпацький В.А. Ініціали, прізвище

Керівник проекту (роботи) Підпис Боровик О. В. Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ. 190118.07.07.01 ПЗ	Пояснювальна записка	73		
			Графічні матеріали			
2		КвРКІ. 190118.07.07.01 Е8	Комунікація на рівні платформи та інфраструктури	1		
3		КвРКІ. 190118.07.07.01 Е8	Типова архітектура МППЗ	1		
4		КвРКІ. 190118.07.07.01 Е8	Передача/прийом драйверів в чергу	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ. 190118.07.07.01 ПЗ		
Розробив		Кліпацький.В			Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Боровик О.				У	1	1
Н. контр.		Лисенко С.М				ХНУ, КІ2-19-1		
Затв.		Говорущенко Т		26.01				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ»

Автор роботи: *Кліпацький Владислав Андрійович*.

Керівник роботи: *Боровик Олег Васильович*.

Пояснювальна записка: *60 с.*, 5 *рис.*, 1 *табл.*, 3 *дод.*, 52 *джерела*.

Графічна частина: 3 креслення.

СИСТЕМА КЕРУВАННЯ ТРАФІКОМ, МАСШТАБОВАНЕ ПРОМІЖНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВІРТУАЛЬНІ МАШИНИ.

Мета роботи - розробка проміжного програмного забезпечення для комунікації між віртуальними машинами хмарних середовищ.

Об'єктом дослідження є процеси взаємодії між між віртуальними машинами хмарних середовищ.

Предметом дослідження є проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ.

В роботі оцінено затримку мережі за наявності різноманітних шаблонів трафіку та конфігурацій системи, включаючи використання декількох існуючих механізмів керування трафіком Linux. В роботі пропонується система контролю трафіку, що враховує віртуалізацію, яка вирішує ці обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. Напрям роботи був спрямований на досягнення комунікації в режимі реального часу на рівні платформи та інфраструктури через МППЗ та СКТВ відповідно. МППЗ і СКТВ досягають комунікування в реальному часі незалежно на двох рівнях. Потенційне розширення і розробка інтерфейсу дає змогу координувати ці дві системи. На рівні платформи створено МППЗ, яке має диференціацію затримок, ізоляцію послуг через обмеження швидкості та масштабованість за рахунок розподілу навантаження між брокерами повідомлень. На рівні інфраструктури розроблено систему управління трафіком, що враховує віртуалізацію у віртуалізованих хостах.

Підпис студента

Дата

19.06.2023

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

МППЗ - масштабоване проміжне програмне забезпечення

СКТВ - система контролю трафіку з урахуванням віртуалізації

ВМ - віртуальні машини

ЦРО - цілі рівня обслуговування

РОШ - розподілене обмеження швидкості

РН - розподіл навантаження

ЗС - збір сміття

ВСТУП

Комунікація в режимі реального часу має вирішальне значення для нових хмарних застосунків від розумних міст до промислової автоматизації. Новий клас застосунків, критичних до затримок, вимагає диференціації затримки та ізоляції продуктивності у високомасштабованому вигляді у віртуалізованих хмарних середовищах. Розглянемо розробку нової хмарної архітектури та сервісів для підтримки зв'язку в режимі реального часу як на платформному, так і на інфраструктурному рівнях. На рівні платформи створюємо масштабоване проміжне програмне забезпечення (МППЗ) для обміну повідомленнями в режимі реального часу (платформа), яке має диференціацію затримок, ізоляцію служб через обмеження швидкості, масштабованість через розподіл навантаження між брокерами обміну повідомленнями. Ключовим внеском МППЗ є використання складних взаємодій між обмеженням швидкості та розподілом навантаження. На рівні інфраструктури розробляємо систему контролю трафіку з урахуванням віртуалізації (СКТВ) для роботи у віртуалізованих хостах. СКТВ надає нову архітектуру мережевого вводу-виводу, яка забезпечує диференційовану обробку пакетів з обмеженням швидкості при масштабуванні на багатоядерних процесорах. Дослідження оцінюється в хмарному тестовому стенді в контексті застосунків Інтернету речей. Розгортання розподілених застосунків у хмарних середовищах стає все більш поширеним з недавньою появою хмарних мереж, IoT, Industry 4.0 та 5G. У цих областях хмарні застосунки зазвичай мають дві основні вимоги: масштабованість (вони повинні обробляти дуже паралельні з'єднання від розподілених датчиків, виконавчих механізмів або пристроїв хмарного краю; наприклад, інтелектуальні транспортні засоби [1] контролюють сигнали світлофора по місту на основі даних про обсяг транспортних засобів від тисяч широко розподілених датчиків; затримка (вони повинні підтримувати різноманітність у гарантіях наскрізної затримки, які вони пропонують; наприклад, інтелектуальні програми вимагають затримки в обидва

кінці менше 1 секунди [2], тоді як застосунки промислової автоматизації мають набагато менші вимоги до затримки порядку 0,5-1 мс [3], а програми моніторингу погоди по суті нечутливі до латентності).

На сьогоднішній день задоволення вимог масштабованості було реалізовано за рахунок використання хмарної платформи, яка забезпечує: координаційні запущені платформи для розподіленого розгортання. Для великомасштабних хмарних застосунків проміжне програмне забезпечення (платформа) обміну повідомленнями [3, 5, 6] є одним з найважливіших, оскільки воно підтримує масштабовані парадигми комунікації. Воно забезпечується віртуалізацією, яка запускає застосунки, що працюють на ізольованих віртуальних машинах (ВМ), так що інфраструктура кожного потужного фізичного хоста може бути розділена та надана декільком хмарним клієнтам. З іншого боку, задовольнити вимогу затримки є більш складним завданням, оскільки ресурси платформ та інфраструктури спільно використовуються застосунками з різноманітними об'єктами рівня обслуговування. У таких спільних середовищах загальним рішенням є диференціація латентності (обслуговування) з застосунками, зіставленими з класами послуг, які відповідають із затримкою. Диференціація послуг - це добре вивчена проблема з безліччю можливих рішень. Через свою простоту загальним підходом у спільних платформах виконання або інфраструктурах покладання на пріоритети [6, 7]. Зокрема, постачальники послуг можуть створювати екземпляри (наприклад, брокери обміну повідомленнями, віртуальні машини) для кожного класу послуг і використовувати планувальники процесів на основі пріоритетів або віртуальних машин для визначення пріоритетів цих випадків.

Незважаючи на переваги простоти систем, заснованих на пріоритетах, вони також вимагають контролю доступу до вищих класів пріоритетів. Якщо програма в класі обслуговування з високим пріоритетом (випадково або навмисно) неправильно поводиться і генерує набагато більший мережевий трафік, ніж очікувалося, вона може перевантажити загальні ресурси. Це, в свою чергу, може вплинути на сервісні гарантії інших застосунків, особливо тих,

що знаходяться в класах обслуговування з низьким пріоритетом. Стандартний підхід до пом'якшення цього ризику спирається на обмеження ставки, яка задає верхні межі обсягу трафіку, який програмі дозволено вводити в загальну систему. Спільно підтримуючи диференціацію затримок та обмеження швидкості, загалом досягається зв'язок у режимі реального часу. Однак у хмарних середовищах спеціальна архітектура мережі вимагає комунікування в режимі реального часу, що здійснюється на двох різних рівнях, що є основною проблемою, яку потрібно вирішити.

Отже, в роботі оцінено затримку мережі за наявності різноманітних шаблонів трафіку та конфігурацій системи, включаючи використання декількох існуючих механізмів керування трафіком Linux. В роботі пропонується система контролю трафіку, що враховує віртуалізацію, яка вирішує ці обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. Напрям роботи був спрямований на досягнення комунікації в режимі реального часу на рівні платформи та інфраструктури через МППЗ та СКТВ відповідно. МППЗ і СКТВ досягають комунікування в реальному часі незалежно на двох рівнях. Потенційне розширення і розробка інтерфейсу дає змогу координувати ці дві системи. На рівні платформи створено МППЗ, яке має диференціацію затримок, ізоляцію послуг через обмеження швидкості та масштабованість за рахунок розподілу навантаження між брокерами повідомлень. На рівні інфраструктури розроблено систему управління трафіком, що враховує віртуалізацію у віртуалізованих хостах.

Метою роботи є розробка проміжного програмного забезпечення для комунікації між віртуальними машинами хмарних середовищ.

Об’єктом дослідження є процеси взаємодії між між віртуальними машинами хмарних середовищ.

Предметом дослідження є проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ.

1 ОРГАНІЗАЦІЯ КОМУНІКУВАННЯ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ МІЖ ВІРТУАЛЬНИМИ МАШИНАМИ

1.1 Аналіз механізмів та способів комунікування в реальному часі на двох рівнях між віртуальними машинами

Розглянемо концептуальну комунікацію між ВМ. Ни рис. 1.1 зображено зв’язки між ВМ в хмарному середовищі. На рівні платформи повідомлення доставляються між застосунками через проміжне програмне забезпечення обміну повідомленнями. Застосунки, як відправники / одержувачі повідомлень, розгортаються або зовні (як Інтернет речей), або всередині хмари. Тим часом, оскільки внутрішньо хмарні застосунки працюють у віртуальних машинах, зв'язок між ними проходить через інфраструктурний рівень. Цей зв'язок спеціально обробляється мережевим входом / виходом віртуалізованих хостів.

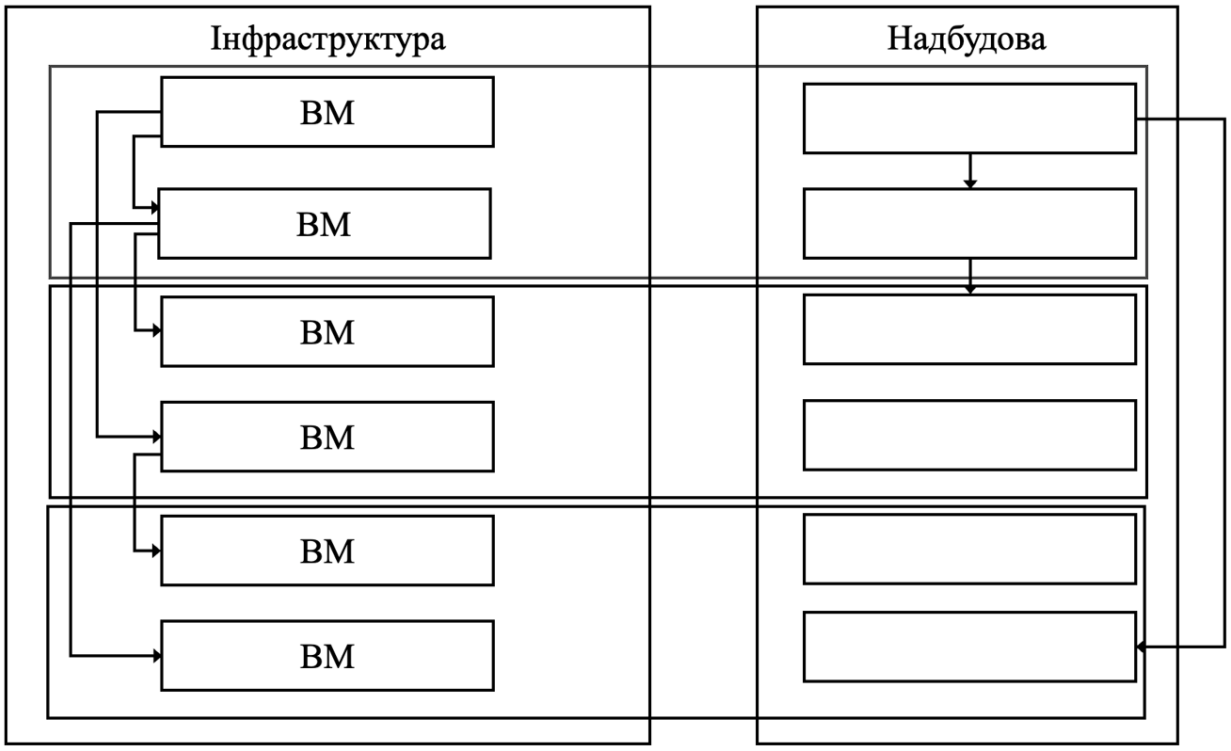


Рисунок 1.1 - Комунікація на рівні платформи та інфраструктури

Розглядатимемо комунікування в реальному часі на двох рівнях через дві системи: МППЗ, масштабоване проміжне програмне забезпечення для обміну повідомленнями в режимі реального часу на рівні платформи; СКТВ, структура управління трафіком з урахуванням віртуалізації на рівні інфраструктури. Для підтримки зв'язку в режимі реального часу обидві системи реалізують диференціацію затримок і обмеження швидкості.

Вимоги до масштабованості та затримки хмарних застосунків IoT вимагають масштабованої платформи вимірювання, яка підтримує диференціацію послуг. Тому, розробляємо і впроваджуємо таке проміжне програмне забезпечення МППЗ. У МППЗ повідомлення кожного класу обслуговування обробляються спеціалізованими (програмними) брокерами, і використовуємо планувальники процесів на основі пріоритетів, наприклад, FIFO в Linux, щоб визначити пріоритетність цих брокерів. Тому, повідомлення з різними вимогами до затримки диференційовані. Оскільки ця частина роботи є простою, то зосереджуємось на обмеженні швидкості та масштабованості (через розподіл навантаження між брокерами) МППЗ. Між цими двома особливостями існують складні взаємодії, які негативно впливають на латентність. Вивчення цього впливу та пропозиція рішення щодо застосування є основним внеском МППЗ. Рівень інфраструктури (віртуалізовані хости), ресурси спільно використовуються мережевими потоками з різними вимогами до затримки. Для забезпечення диференційованого та ізольованого спільного використання мережевих мереж, існуючий контроль трафіку покладається на пріоритетні планувальники пакетів та обмеження швидкості дисциплін масового обслуговування Linux. Однак, оскільки віртуалізація вводить додаткові компоненти, які вимагають ресурсів процесора, існуючий контроль трафіку показує обмеження, які несуть підвищену затримку мережевого трафіку в режимі реального часу. Щоб пом'якшити ці обмеження, пропонується СКТВ, система контролю трафіку з урахуванням віртуалізації, яка забезпечує диференційований та ізольований обмін процесорами між мережевими потоками. Ключовим внеском СКТВ

є забезпечення нової базової архітектури мережевого вводу-виводу у віртуалізованих хостах.

Хмара та її численні екземпляри [8] відкрили нову еру доступу до обчислень, а це, у свою чергу, уможливило вибух розподілених застосунків, зокрема в просторі Інтернету речей [9-12]. Застосунки IoT зазвичай включають великий обсяг даних, що генеруються в багатьох джерелах (датчиках), розподілених по географічно різних місцях, і які повинні бути оброблені та своєчасно використані, наприклад, з метою виконання. Отже, вони вимагають ефективних рішень для передачі та обробки даних. Поєднання притаманної хмарі обчислювальної гнучкості та масштабованих комунікаційних платформ робить її привабливою платформою для застосунків IoT [13-16]. Це призвело до розвитку комунікаційних платформ, таких як Microsoft Azure Service Bus та Amazon AWS IoT. Ці платформи засновані на парадигмі публікації/підписки, яка дозволяє підключатися великій кількості відправників і одержувачів без необхідності складної сітки з'єднань один на один. Типова архітектура для такої системи зображена на рис. 1.2, з такими темами, як абстракція, що використовується для з'єднання видавців (відправників) та передплатників (приймачів) є перспективною для розгляду.

Брокери повідомлень виступають посередниками між видавцями та передплатниками, отримуючи та ставлячи в чергу та пересилаючи повідомлення для дифферентних тем. Видавці публікують брокеру повідомлення на певну тему, а передплатники підписуються на брокерів, щоб отримувати повідомлення з цієї теми. Масштабованість реалізується за рахунок наявності декількох брокерів, між якими можна розподілити робоче навантаження [17, 18], як з різних тем, так і для окремих тем з великим навантаженням на повідомлення. Це забезпечує швидкий доступ до додаткових потужностей у разі потреби. Як і з будь-яким спільним ресурсом, навантаження брокерів повідомлень потрібно контролювати, щоб забезпечити досягнення цілей рівня обслуговування (ЦРО). Це особливо важливо для застосунків IoT, які вимагають своєчасної доставки (і обробки) своїх даних. Якщо додаток/тема (випадково чи навмисно) неправильно

поводиться та генерує набагато більше навантаження на повідомлення, ніж очікувалося, це може перевантажити ресурси платформи (процесор та пам'ять) і, у свою чергу, вплинути на дані інших тем. Стандартним підходом до вирішення цього питання є обмеження ставки обсягу повідомлення кожної теми.

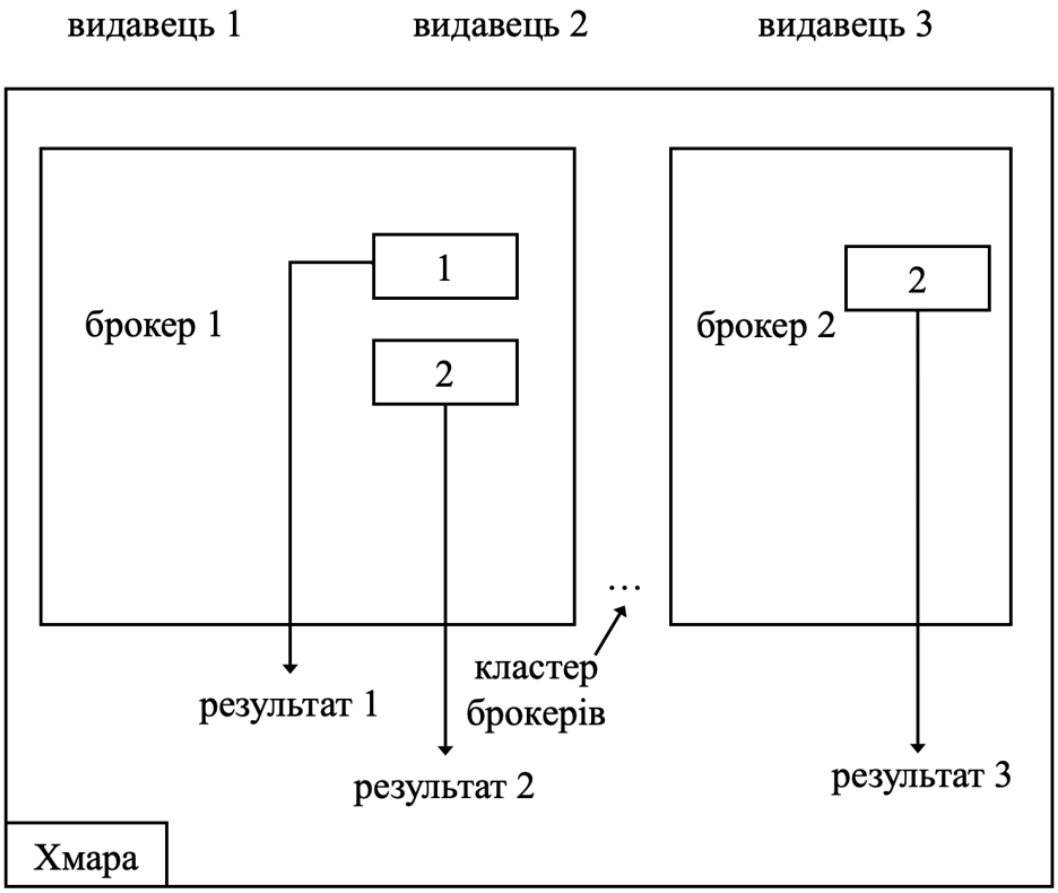


Рисунок 1.2 - Типова архітектура платформи заснована на парадигмі публікації/підписки

Обмеження швидкості використовується в декількох публічних хмарних платформах і зазвичай реалізується через програмний API шлюз [19, 20]. На практиці механізм обмеження швидкості має форму сегмента токенів [7, 21-26], де кожне повідомлення вимагає токен, перш ніж його зможе обробити його брокер. При відсутності жетонів повідомлення, що прийшло, повинно чекати його (як варіант, його можна скинути). Сегмент маркера теми зазвичай є частиною її ЦРО і визначається двома параметрами. Параметр визначає

швидкість, з якою генеруються токени, і, отже, обмежує довгострокову швидкість повідомлень, на яку має право тема. Параметр вказує на максимальну кількість токенів, які може накопичити тема, що в свою чергу, обмежує максимальну кількість повідомлень, які вона може відправити своєму брокеру, не несучи затримки доступу (очікування токенів). Їх комбінація визначає "конверт" робочого навантаження [27], який забезпечує регулятор швидкості.

Розглянемо інтелектуальну систему [28], яка повинна обробляти дані про обсяг засобів, отримані від тисяч датчиків, розподілених по всьому регіону, і реагувати протягом секунди або менше [29], щоб забезпечити належний контроль. У цій системі датчики служать видавцями інформації та хмарними серверами, відповідальними за обробку цієї інформації, як передплатники служби обміну повідомленнями. Потреба в своєчасних відповідях вимагає надання інфраструктури обміну повідомленнями відповідно до ЦРО системи, як правило, у формі гарантії затримки доставки повідомлень. Оскільки кілька тем мають однакові брокери, зустріч із ЦРО вимагає (оцінити), обмежуючи навантаження користувачів на повідомлення. Отже, обсяг повідомлень, які генерують наші інтелектуальні датчики системи, спочатку профілюється, при цьому його профіль використовується для налаштування обмежувача швидкості, тобто для забезпечення невеликої затримки доступу до тих пір, поки вона відповідає до відповідного конверта. Функціональність обмеження швидкості зазвичай реалізується на одному шлюзі. Це централізоване рішення з очевидними обмеженнями, коли справа доходить до масштабованості. Оскільки обмеження швидкості, як правило, відбувається з точки зору одиниць даних застосунків, наприклад, повідомлень, то шлюз вводить додатковий «перехід» програми, оскільки він повинен переконструювати (з пакетів TCP або UDP) ці одиниці даних програми для виконання його функціональності, тобто контролювати швидкість кількості повідомлень, які він пропускає. Цей додатковий перехід програм призводить до непотрібної додаткової затримки, яка може бути

особливо шкідливою для програм у режимі реального часу. Ці недоліки були визнані раніше і мотивували вивчення рішень розподіленого обмеження швидкості (РОШ) [7, 30-34]. Система РОШ включає в себе кілька обмежувачів швидкості, кожен з яких пов'язаний з іншим ресурсом, до якого була призначена заявка, наприклад, брокери. Ці обмежувачі швидкості потім співоцінюються, щоб гарантувати, що між ними сукупний трафік, який вони пропускають, відповідає тій самій загальній оболонці робочого навантаження, що й централізований обмежувач. У контексті інтелектуальної системи різні набори видавців (датчиків) призначаються різним порушникам повідомлень (для розподілу робочого навантаження повідомлень), з окремою функцією обмеження швидкості у кожного брокера, який контролює обсяг повідомлень, які він повинен обробити. Загалом, рішення РОШ вимагає розбиття вихідного сегмента токенів на сегменти субтокенів, призначені окремим ресурсам, і, можливо, динамічно коригування їх у відповідь на коливання робочого навантаження. Натомість, розповсюдження обмежувача швидкості може знадобитися, наприклад, для обробки теми з великим навантаженням. Воно за своєю суттю пов'язане зі збільшенням затримки доступу. Зокрема, факт розподілу сегмента токенів збільшує затримку доступу, яку він вводить. Пропонується впроваджувати та оцінювати можливе рішення для пом'якшення цього покарання, зберігаючи переваги розподілу навантаження. Розглянемо питання розподілу навантаження (РН), яке виглядає наступним чином: набором брокерів повідомлень з існуючим робочим навантаженням, які повинні розподілити видавців нової теми між брокерами, щоб «найкраще» відповідати ЦРО; визначення найкращого полягає в ефективному використанні ресурсів, наприклад, отримання більшої залишкової здатності для рівної продуктивності або здатності підтримувати більш високе робоче навантаження на обмін повідомленнями. Для конфігурації, яка включає нову тему, яка може або призначити всіх своїх видавців одному брокеру з використанням обробки повідомлень у брокера, або вирішити розділити своїх видавців на двох брокерів, кожен з яких потім з використанням обробки повідомлень відповідно.

В останньому випадку сегмент токенів вихідної теми розбивається на два сегменти субтокенів, по одному у кожного брокера, з параметрами, які перевіряють, щоб зберегти ту саму сукупну частоту повідомлень і чергу повідомлень. Відповідь на РН потім вимагає визначення конфігурації, яка найкраще відповідає цілям продуктивності теми (затримки). Це питання було широко досліджено, навіть якщо необхідно проявляти певну обережність у випадках серверів з нерівномірною швидкістю [35]. Як правило, доступ до більшої потужності обробки повідомлень, тобто розподіл робочого навантаження між більшою кількістю процесорів, дає кращу продуктивність через зниження навантаження на окремі процесори, і це вбудовано в більшість стратегій балансування навантаження. Інша ситуація, коли затримка впливає як на затримку обробки повідомлень, так і на затримку доступу, яку може ввести функція обмеження швидкості. Зокрема, розщеплення функції обмеження швидкості негативно впливає на її латентність. Назвемо це штрафом РОШ. Таким чином, відповідь на РН тепер передбачає компроміс між зниженням затримки обробки повідомлень і збільшенням затримки доступу (обмеження швидкості). Враховуючи двопараметричний сегмент токена потрібно розділяти цей сегмент токена на кілька. Розділення збільшує затримку доступу до сегмента токена на прикладі шаблону прибуття повідомлень. Очікувана затримка в сегменті токена повідомлення , швидше за все, призведе до очікування. Збільшення затримки від повільніших ставок токенів$_l$ неминуче і не залежить від припущення про розподіл Пуассона. З іншого боку, той факт, що явно залежить від припущення про процес Пуассона. Це вказує на можливість того, що для різних процесів прибуття цей штраф не завжди може виникати або може бути пом'якшений, якщо правильно сформулювати процес прибуття у кожного брокера. Щоб краще зрозуміти, коли і чому це може бути так, розглянемо, можливо, екстремальний, але ілюстративний приклад синхронізованих видавців, тобто всі генерують повідомлення одночасно, щоб створити сплеск повідомлень. У такому екстремальному сценарії, оскільки видавці розділені між брокерами, відбувається і сплеск, при цьому розмір

черги у кожного брокера зменшується в тій же пропорції, що і розмір всієї множини повідомлень. Всі інші параметри, наприклад, навантаження, будучи однаковими, гарантують, що штраф тепер зникне. Як відображено в РН, МППЗ прагне запропонувати службу обміну повідомленнями (для застосунків IoT), яка є ефективною у використанні хмарних ресурсів і здатною забезпечити гарантії затримки (ЦРО). Загальний ЦРО має форму гарантії затримки хоста, наприклад, 99-й процентиль затримки нижче 1 мс. Це вимагає як контролю робочого навантаження на обмін повідомленнями, що походить від користувачів (за допомогою обмеження швидкості), так і визначення того, як найкраще розподілити це навантаження між ресурсами обробки повідомлень (брокерами). Складність полягає в протилежному впливі розподілу навантаження на обробку повідомлень і затримку обмеження швидкості відповідно. Повторенням проблеми, з якою стикаються рішення РОШ, було б природним орієнтиром. Розділення лише в тому випадку, якщо потрібно. Іншими словами, розповсюджуйте видавців теми серед найменшої кількості брокерів, гарантуючи, що навантаження на обробку повідомлень не призведе до порушень ЦРО. Крім того, якщо розділити вантаж, то можна розколоти сплеск, хоча б в міру можливості. Зокрема, видавці, час передачі повідомлень яких, як правило, корелює і, отже, сприяє формуванню сплеску, повинні бути призначені різним брокерам.

МППЗ включає в себе три принципи: концентрація, яка полягає у визначенні найменшої кількості брокерів, необхідних для задоволення ЦРО нової теми; максимізації мінімального навантаження, а отже, і ставки токена, присвоєної будь-якому брокеру, обізнаність про кореляцію. Призначення видавців брокерам, щоб мінімізувати міжвидавничу кореляцію і, отже, максимально зменшити розрив процесу надходження повідомлень у кожного брокера є важливим кроком. Концентрація спрямована на те, щоб уникнути або мінімізувати покарання РОШ, коли це можливо. Зокрема, штраф РОШ може зростати лінійно і часто суперлінійно з кількістю субтокенів, на які розподіляється робоче навантаження. Отже, природно уникати розділення

теми, якщо навантаження брокера не дає затримки обробки, яка порушує ЦРО. Незалежно від процесу прибуття, затримка доступу до повідомлень, які зазнають затримки, обернено пропорційна ставці токена. Типова архітектура МППЗ зображена на рис. 1.3.

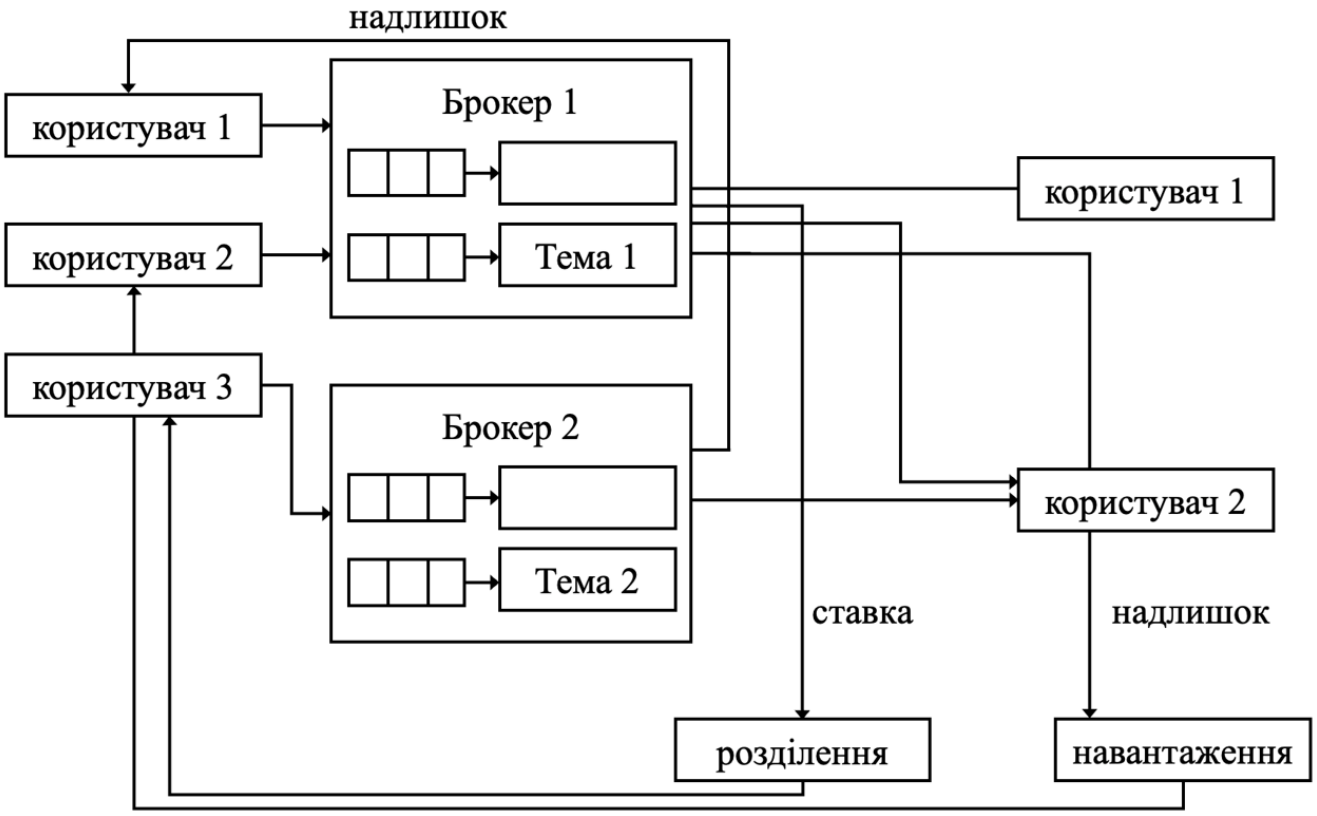


Рисунок 1.3 – Типова архітектура МППЗ

Отже, збереження мінімальної швидкості токенів у сегментах субтокенів якомога вищим є бажаним для досягнення гарантій затримки хоста. Кореляція-обізнаність прагне вибрати видавців, щоб зменшити «сплеск» процесу прибуття в кожен сегмент субтокенів таким чином, щоб паралельно зменшити розмір відповідного розміру множини. Зменшення черг прибуття має принести користь як штрафу РОШ, так і затримці обробки повідомлень у брокерів. Пропонований високорівневий огляд типової конфігурації (рис. 1.3), він складається з окремих брокерів, а видавці певної теми розподілені між одним або декількома брокерами. Основний компонент МППЗ є дистриб'ютором навантаження, який відповідає за розповсюдження видавців

нової теми серед брокерів на основі ЦРО теми та існуючого навантаження брокерів (рис. 1.3). Крім того, адаптер ТВ відстежує стан сегментів субтокенів під час виконання та запускає налаштування їх параметрів у відповідь на зміни трафіку повідомлень. Архітектура процесів розподільника навантаження МППЗ є основним компонентом, відповідальним за реалізацію принципів. Інші компоненти, включаючи адаптер ТВ розглянемо після побудови архітектури.

Розподільник навантаження розроблений на основі трьох принципів. Однак їх реалізація вимагає вирішення двох практичних проблем.

Оцінка потужності. На відміну від традиційних балансувальників навантаження, МППЗ не прагне рівномірно розподіляти навантаження між наявними ресурсами. Натомість принцип концентрації вимагає визначення найменшої кількості брокерів, які можуть вмістити нову тему за умови її ЦРО (цільового показника затримки хоста). Це, і в меншій мірі принцип max-min, по суті, є проблемою «контролю допуску», коли система продовжує призначати видавців брокеру, якщо тема ЦРО не порушується. Такі рішення щодо контролю допуску вимагають точної оцінки кількості видавців, з якими брокер може впоратися відповідно до ЦРО. На практиці оцінка можливостей брокерів є складною, оскільки обробка повідомлень включає набір взаємозалежних і одночасних завдань. Наприклад, проміжне програмне забезпечення обміну повідомленнями з відкритим вихідним кодом [34-37] реалізовано з використанням мови Go [38], яка забезпечує легкий і масштабований паралелізм. Це добре підходить для застосунків IoT, які включають велику кількість одночасних підключень і як основи для впровадження МППЗ. Моделювання поведінки базового планувальника часу виконання, включаючи його залежність від стратегії втрат роботи для використання багатоядерних систем, є нетривіальним. Крім того, залежно від рівня паралельності (кількості видавців) теми та того, як видавці генерують повідомлення, вузькі місця продуктивності мігрують через компоненти. Це робить модельний підхід в основному непрактичним і змушує замість цього покладатися на підхід, заснований на вимірюванні [38]. Зокрема,

використовуємо ітеративні вимірювання, щоб дізнатися, як найкраще розподілити видавців теми між брокерами, досягаючи цільової затримки.

Облік кореляції. Інша проблема дизайну виникає в реалізації принципу кореляції - усвідомлення. Зокрема, на основі еквівалентності, враховуючи, що робоче навантаження має бути розподілене між кількома брокерами, тоді метою МППЗ є визначення призначення видавців, яке призводить до найменшого можливого збільшення між брокерами. Це означає створення процесу прибуття до кожного брокера, який знижує ймовірність того, що незавершена робота перевищує розмір множини. Це складно, оскільки вимагає точної часової характеристики робочого навантаження, створеного окремими видавцями. Розумним варіантом є створення процесу прибуття до кожного брокера з найменшою можливою дисперсією часу між прибуттям. Однак створення таких процесів прибуття з окремих процесів прибуття видавців є обчислювально складним. Крім того, дисперсія відображає лише «стаціонарну статистику» процесу прибуття при кожному розриві, і тому не повністю відображає часову кореляцію. Отже, мінімізація відхилення процесу прибуття у кожного брокера може не завжди усвідомлюють мету мінімізації збільшення навантаження. Альтернативи, які безпосередньо вимірюють тимчасову кореляцію, ще більш складні. Ці проблеми змушують замість цього покладатися на зовсім іншу альтернативу, а саме на кореляційні ключі видавця, вказані користувачем, які відображають семантику на рівні програми IoT.

1.2 Ітеративний розподіл робочого навантаження

Коли надходить нова тема, дистриб'ютор навантаження несе відповідальність за розповсюдження своїх видавців серед брокерів, щоб відповідати темі ЦРО. Важко точно оцінити кількість видавців, яких брокер може розмістити за умови ЦРО. Щоб подолати цю проблему, дистриб'ютор навантаження використовує ітеративний процес, який розподіляє видавців між

брокерами на етапі профілювання на основі вимірювань. Кожна ітерація процесу розподілу навантаження працює наступним чином: розподільник навантаження спочатку оцінює мінімальну кількість брокерів з достатньою доступною потужністю для розміщення нової теми; видавці теми потім призначаються до брокерів відповідно до принципів max-min та кореляційної обізнаності; після того, як видавці були призначені до брокерів, кожен брокер проводить незалежну фазу профілювання, щоб перевірити, чи може він вмістити свій новий робочий варіант без порушення свого ЦРО. Після третього кроку кожен брокер знає, чи може він впоратися зі своїм новим навантаженням або йому потрібно позбутися деяких видавців, щоб відповідати своєму ЦРО. Якщо порушень ЦРО немає, то процес розподілу навантаження вважається успішним і завершується. В іншому випадку брокери, ЦРО яких був порушений, визначають, як багато видавців їм потрібно відхилити, повідомити про цей номер дистриб'ютору та позначити себе як «повні». Потім дистриб'ютор починає нову ітерацію для розповсюдження випущених публікацій, можливо, із залученням додаткових брокерів. Процес закінчується, коли всі видавці призначаються брокерам, які можуть їх розмістити, або в системі закінчуються брокери. Коли це станеться, більше брокерів можуть з'явитися, надіславши запит до хмари. Далі опишемо кожен крок більш детально, за винятком кореляційного призначення видавців, яке є предметом окремого підрозділу. Для простоти спочатку опишемо процес, який використовується на першій ітерації, яка відбувається, коли надходить нова тема. Потім розширюємо обговорення, описуючи, як оновлюються змінні, що використовуються в кожній ітерації. Крок 1 відображає мету концентрації та прагне визначити мінімальну кількість брокерів, необхідну для розміщення нової теми із заданою сукупною частотою повідомлень. Залишкова потужність обробки повідомлень кожного брокера спочатку оцінюється на основі різниці між його максимальною потужністю обробки повідомлень і його поточною частотою повідомлень (сума частоти повідомлень вже призначених брокеру). Потім брокери сортуються в порядку зменшення значення, щоб

визначити мінімальну початкову кількість брокерів, яким слід призначити робоче навантаження нової теми (найменша кількість брокерів, така, що сума їх значень перевищує частоту повідомлень теми). Крок 2 присвячений визначенню того, як найкраще розподілити навантаження на повідомлення нової теми між цими брокерами відповідно до принципів max-min та кореляційної обізнаності. Це передбачає обчислення для кожного брокера квоти робочого навантаження, яку він повинен призначити. Ця квота встановлена на мінімумі залишкової потужності брокера та його справедливої частки норми навантаження. Це максимізує мінімальне призначення у кожного брокера (принцип max-min), якщо не обмежений залишковою ємністю. Після встановлення квот значення капіталізації брокерів оновлюються, щоб відобразити їх новий розподіл, а видавці призначаються брокерам для реалізації цих розподілів у спосіб, що враховує кореляцію. Крок 3 визнає, що оцінки враховують лише частоту повідомлень, і тому ігнорують багато факторів, які впливають на продуктивність, наприклад, вибух прибуття, паралелізм (між підключеннями від різних видавців), взаємодію між робочими навантаженнями тощо. Крок 3, таким чином, покладається на підхід, заснований на вимірюванні, для оцінки фактичної ефективності кожного брокера після кроку 2. Зокрема, крок 3 складається з етапу профілювання, який називають, метою якого є перевірка ЦРО кожного брокера щодо виконання після додавання нових видавців (вимірюючи затримку наскрізного повідомлення), і якщо це не так, то визначення, скільки видавців брокеру потрібно обробити, щоб повернутися до дотримання ЦРО. Ітераційний двійковий пошук запускається для оцінки максимальної підмножини додаткових видавців, яку брокер може вмістити, не порушуючи свого ЦРО. Надлишки видавців повертаються дистриб'ютору вантажу, а брокер позначається як повний. Ця наступна ітерація паралельна першій з оновленими змінними. Окрема змінна записує сукупну непризначену частоту повідомлень від надлишкових видавців, тоді він збільшується, щоб врахувати найменшу кількість додаткових брокерів, необхідних для розміщення ставки таким чином, щоб знову ж таки відповідати

принципам max-min та кореляційно-усвідомлений принципи. Ітерації тривають, доки всі видавці не будуть призначені брокерам, які можуть їх розмістити, або в системі не закінчаться брокери. Незважаючи на свою ефективність, цей ітеративний процес має недоліки. По-перше, його природа, заснована на вимірюваннях, передбачає, що конвергенція може зайняти час. Кількісно оцінюємо ці накладні витрати. По-друге, програми можуть зазнати тимчасового погіршення обслуговування. Це неминуче в будь-якому підході, заснованому на вимірюванні і є наслідком труднощів при побудові досить загальної і точної моделі.

Вимірювання кореляції між видавцями теми є складним завданням. Як результат, ми застосовуємо прагматичний підхід, коли користувачі надають інформацію про кореляцію на основі семантики програми, загальнодоступної в налаштуваннях IoT. Це дозволяє користувачам явно ідентифікувати корельовані потоки даних для полегшення ефективної обробки потоків шляхом спільного розгляду потоків у межах одного розділу. Розділи, які призводять до тимчасової кореляції між видавцями, поширені в багатьох розгортаннях IoT. Наприклад, датчики, що знаходяться в безпосередній близькості один від одного, часто спрацьовують приблизно в один і той же час при виявленні одного і того ж фізичного явища. Видавці, пов'язані з цими датчиками, потім створюватимуть пов'язані шаблони надходження повідомлень до системи обміну повідомленнями. Адаптуємо концепцію ключів розділів у МППЗ та надаємо API користувачам, які дозволяють їм позначати пов'язаних видавців ідентичною кореляцією. Завантажувальник прагне призначити видавцям з однаковим ключем кореляції різних брокерів, тим самим розділяючи «сплески» повідомлень, які вони спільно генерують. Зокрема, дано набір кореляції групи (як вказано користувачем) та кількість видавців, які будуть призначені брокеру, дистриб'ютор завантаження вибирає видавців з кожної кореляційної групи пропорційно кількості видавців у групі.

Спочатку представимо архітектуру, а потім представимо підхід до додавання функціональності МППЗ разом з деякими проблемами. Повідомлення

від видавців надходять через окремі підключення TCP, кожне з яких обробляється підпрограмою. Повідомлення від видавців певної теми потім передаються до іншої теми. Тема має спеціальний буфер, в який переміщується повідомлення. Після того, як тему заплановано до виконання, він запускається до завершення, витягуючи повідомлення зі свого буфера та пересилаючи їх, поки буфер не стане порожнім. Він відправляє повідомлення абонентам через окремі TCP-з'єднання. Середовище виконання Go використовує планувальник крадіжки роботи для планування всіх процедур на багатоядерних платформах. Легкі процедури та планувальник важливі для масштабованості, яким, можливо, доведеться обробляти велику кількість видавців через одного брокера. МППЗ застосовує обмеження швидкості за допомогою сегмента маркерів, який знаходиться в ньому. Під час виконання кожна тема відстежує стан свого сегмента токенів. Коли заплановано процедуру, вона витягує повідомлення зі свого буфера вхідних повідомлень, лише якщо її сегмент маркерів містить маркери. В іншому випадку повідомлення чекають у буфері повідомлень, поки маркери не стануть доступними. Хоча логіка сегмента токена добре зрозуміла, нетривіально реалізувати його точну тимчасову поведінку в середовищі з високим паралелізмом, такому як час виконання Go. Зокрема, сегмент маркера визначає відповідність повідомлення на основі його стану (кількості токенів) та часу суперництва повідомлення. Простий підхід до вимірювання часу прибуття полягає в тому, щоб позначити час повідомлень, коли токен зчитує їх з буфера прийому TCP. Однак, через спільне планування, час, коли заплановано читати повідомлення, може демонструвати значні відмінності від того, коли повідомлення спочатку надходять в TCP-буфер. Оскільки логіка сегмента маркера оновлює кількість токенів залежно від того, скільки часу минуло з моменту отримання повідомлення, позначка часу пізнього прибуття може призвести до того, що сегмент маркера помилково затримуватиме повідомлення, які насправді відповідають. Особливе занепокоєння викликає збільшення затримки планування з кількістю повідомлень в системі. Кілька видавців можуть легко вносити великі

тимчасові помилки в поведінку сегмента токенів. Щоб усунути ці помилки, МППЗ використовує мітку часу TCP-рівня, яка позначає прибуття кожного пакета TCP всередині ядра Linux. Кожного разу, коли токен читає дані, позначки часу прибуття (на рівні TCP) також копіюються (як позасмугові дані) у простір користувача. Оскільки призначені для користувача дані (повідомлення) часто не зіставляються з пакетами (фрагментами) TCP у співвідношенні 1:1, кожен з них підтримує зіставлення між отриманими повідомленнями та пакетами TCP і призначає кожному повідомленню мітку часу з правильного пакета TCP. Оскільки мітки часу прибуття на TCP-рівні не залежать від рутинного планування, МППЗ здатний застосовувати обмеження швидкості з більш високою тимчасовою точністю.

1.3 Постановка задачі

З розвитком все більш потужних і гнучких платформ віртуалізації, розподілені м'які застосунки реального часу все частіше розгортаються у віртуалізованих середовищах. Ці засоби створюють нові виклики, коли потрібно гарантування низької та передбачуваної тривалої роботи з виконання.

Для забезпечення досягнення виконання такої мети потрібно виконати такі завдання:

1. Оцінити затримку мережі за наявності різноманітних шаблонів трафіку та конфігурацій системи, включаючи використання декількох існуючих механізмів керування трафіком Linux.

2. Розробити систему контролю трафіку, що враховує віртуалізацію, яка вирішує обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. Додатково СКТВ масштабувати до багатоядерних процесорів і забезпечити справедливий механізм розподілу навантаження.

3. Досягнути комунікації в режимі реального часу на рівні платформи та інфраструктури через МППЗ та СКТВ відповідно.

4. МППЗ реалізувати з обмеженим діапазоном динаміки навантаження, а саме зі змінами навантаження між видавцями в межах певної теми. СКТВ ребалансування призначень запускати тільки в момент створення або завершення роботи гостьового домену. Досягти комунікування МППЗ і СКТВ в реальному часі незалежно на двох рівнях.

5. На рівні платформи створити проміжне програмне забезпечення, яке має диференціацію затримок, ізоляцію послуг через обмеження швидкості та масштабованість за рахунок розподілу навантаження між брокерами повідомлень.

6. На рівні інфраструктури розробити систему управління трафіком, що враховує віртуалізацію у віртуалізованих хостах. Також, визначити обмеження механізмів контролю трафіку дисципліни масового обслуговування Linux, які залишають спільне використання процесора між мережевими потоками незахищеним.

7. Провести експериментальні дослідження з розробленим МППЗ.

2 СИСТЕМА КЕРУВАННЯ ТРАФІКОМ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ МІЖ ВІРТУАЛЬНИМИ МАШИНАМИ

2.1 Система керування трафіком в реальному часі на двох рівнях між віртуальними машинами

Збір сміття (ЗС) в середовищі виконання Go може мати значний вплив на хостову затримку обробки повідомлень. Коли ЗС спрацьовує, середовище виконання Go використовує маркерні токени для позначення розподілу пам'яті, що може споживати до 25% процесорного часу [40, 41]. В результаті виявлено значне збільшення 99-го процентиля в хостовій затримці обробки повідомлень при спрацьовуванні ЗС. Оскільки ЗС зазвичай запускається на вимогу в Go, то можна мінімізувати виклики ЗС за рахунок зменшення динамічного розподілу пам'яті і, отже, уповільнення зростання розміру купи. Створимо попередньо виділений кільцевий буфер для кожного токена, таким чином, щоб дані, що зчитуються з сокета TCP, безпосередньо записувалися в існуючий слот у кільцевому буфері замість того, щоб запитувати а динамічне виділення пам'яті. На додаток до ЗС на вимогу, середовище виконання Go використовує ЗС, якщо в інтервалі в 2 хвилини (за замовчуванням) немає ЗС. Відключимо цю функцію, щоб в МППЗ ЗС спрацьовував виключно на вимогу. Хоча ці оптимізації не повністю усувають рівень ЗС, вони ефективно пом'якшують його для 99-ї процентильної затримки хоста в експериментах. МППЗ виконує розподіл навантаження після прибуття теми. Однак під час виконання трафік теми може зміщуватися між видавцями, а отже, і серед порушників, навіть якщо трафік теми залишається відповідним її глобальному трафіку. Отримані неправильно підібрані конфігурації сегмента субтокенів можуть ввести штрафи РОШ у окремих брокерів. Щоб уникнути цього, МППЗ використовує адаптер ТВ для динамічного налаштування конфігурацій сегмента субтокенів у відповідь на зміни трафіку під час виконання. Зокрема, кожен брокер періодично повідомляє ТВ Adaptor про

швидкість і вибух статистики своїх тем. У реалізації це базується на вікні історії 10 секунд для кожної теми, яке відстежує середню частоту повідомлень та максимальне відставання над цим вікном з кроком 1. TB Adaptor регулює частоту токенів пропорційно середній частоті повідомлень і розміри сегментів токенів у пропорції до максимального відставання. Щоб збалансувати швидкість реагування та стабільність, адміністратори МППЗ можуть регулювати довжину вікна історії та частоту оновлень. Адаптер ТВ лише опрацьовує параметри сегмента субтокенів, щоб уникнути штрафів РОШ, спричинених зміною трафіку серед видавців. Це не вирішує проблеми, які можуть виникнути, коли зміна трафіку перевантажує брокера. Робота з такими сценаріями вимагає можливості міграції видавців серед брокерів. Хоча ця функція була реалізована і використовується на етапі профілювання, її впровадження як механізму виконання залишається складним завданням. Емпірична оцінка МППЗ, а точніше різних принципів де-знаків, на які вона спирається починається з експериментів, спрямованих на ілюстрацію впливу РОШ на затримку хоста, тоді як переходити до кількісної оцінки відносних переваг, отриманих від кожного з трьох принципів МППЗ. Це реалізується завдяки вдосконаленню проектів, які включають принципи МППЗ один за одним.

Генеруємо робоче навантаження для обміну повідомленнями, яке прагне імітувати трафік IoT. У типовому налаштуванні IoT, повідомлення на тему надходять від кількох видавців, при цьому видавці корелюють на один датчик або шлюз, що агрегує групу датчиків. Повідомлення можуть бути ініційованими за часом або подіями. Повідомлення, що ініціюються часом, зазвичай генеруються періодично. Наприклад, інтелектуальні системи можуть вимагати періодичного оновлення трафіку від датчиків. І навпаки, повідомлення, ініційовані подіями, генеруються при виявленні конкретних змін навколишнього середовища. Наприклад, можуть спрацьовувати, коли температура опускається нижче або перевищує певний поріг. Імітуємо трафік, ініційований часом, а видавці періодично надсилають повідомлення. Оскільки не вдалося захистити реальні

сліди повідомлень, ініційованих подіями, тоді апроксимували отриманий трафік за допомогою процесу Пуассона (випадкові події). Як періодичні, так і видавці можуть генерувати повідомлення пакетно, при цьому розмір партії визначає сплеск трафіку. Наприклад, шлюз, що керує групою датчиків, генерує пакет повідомлень, якщо датчики управляються загальним таймером або спрацьовують від однієї і тієї ж події. Вибрано процентиль наскрізної затримки повідомлень як ЦРО у всіх експериментах. Кожного разу, коли дана архітектура не може відповідати 1мс ЦРО, повідомляємо про продуктивність найкращої конфігурації. Крім того, сегменти токенів тем були налаштовані за допомогою фази профілювання, яка збирала репрезентативне відстеження трафіку. Трасування було використано для офлайн-моделювання 10 продуктивності сегмента токенів, використовуючи швидкість токенів, встановлену вище, ніж частота повідомлень теми, і пошук найменшого розміру сегмента, який забезпечував процентиль затримки доступу до сегмента токена нульовий. Вплив штрафу РОШ на затримку хоста за допомогою двох експериментів, які відрізняються навантаженням на повідомлення. В обох експериментах видавці нової теми мають доступ до брокерів, які для узгодженості спочатку всі простоюють. Перший експеримент ілюструє штраф РОШ, коли навантаження на тему низьке, тобто один брокер номінально може задовольнити тему. Другий експеримент розглядає більш важке навантаження, для якого виникає компроміс між доступом до більшої потужності при розподілі навантаження між брокерами і результуючим збільшенням штрафу РОШ. Обидва експерименти покладаються на видавців, які генерують окремі повідомлення (розмір партії 1) відповідно до процесу Пуассона зі швидкістю 10 мсг/с. Перший експеримент включає тему з 1 000 видавців, які розподілені між 1 або 6 брокерами, при цьому сегмент токенів теми кореляційно розділений між ними. Це ілюструє збільшення 99-ї процентильної затримки з кількістю брокерів, на яких розділена тема. Це пов'язано з тим, що в конфігураціях з низьким навантаженням, як і тут, вигода від доступу до більшої потужності брокера невелика і не компенсує штраф РОШ. Другий

експеримент аналогічний за винятком того, що в ньому зараз використовується 6 000 видавців. Вища частота повідомлень перевантажує одного брокера, про що свідчить випадок, який демонструє велику затримку хоста. Розділення видавців між двома брокерами зменшує затримку хоста, оскільки зменшення затримки обробки повідомлень (від нижнього навантаження брокера) перевищує збільшення штрафу РОШ (від розділення сегмента токенів). Однак поширення теми серед більшої кількості брокерів не приносить користі, оскільки збільшення штрафу РОШ знову перевищує зменшення затримки обробки повідомлень. Це підкреслює компроміс. Експерименти дозволяють виділити внесок кожного принципу, оцінюючи їх загальний вплив при об'єднанні в МППЗ.

Оцінюємо вплив концентрації, порівнюючи базовий підхід, який рівномірно розподіляє видавців теми між усіма доступними брокерами, з альтернативним варіантом, який включає принцип концентрації. Він розподіляє видавців між брокерами таким чином, щоб вирівняти загальне навантаження на кожного брокера, але на відміну від основного варіанту, який здійснює цей розподіл між усіма брокерами, він обмежується найменшою можливою кількістю брокерів, яких вимагає тема. Це число спочатку оцінюється на основі частоти повідомлень теми та значень брокерів, а потім перевіряється за допомогою підходу, заснованого на вимірюванні. Порівняння здійснюється для різних робочих навантажень шляхом варіювання кількості видавців, пов'язаних з темою. Як і раніше, видавці мають фіксовану частоту повідомлень і розглядаємо Пуассона та періодичні видавництва. Змінюємо серію процесу генерації повідомлень кожного видавця, змінюючи розмір пакета повідомлень, який вони генерують. Періодичні видавництва незалежні один від одного, з випадково обраною фазою для свого періоду. Експерименти починаються знову з непрацюючими брокерами і повторюються 10 разів. Результати із середнім і стандартним відхиленням 99-ї процентильної латентності, повідомленої для кожної конфігурації. Кількість брокерів, між якими розподілено робоче навантаження теми, також відображається поруч із кожною

точкою даних. Результати якісно узгоджуються в різних сценаріях і ілюструють переваги принципу концентрації (відповідає ЦРО теми для всіх конфігурацій). Цифри, також, виділяють два відносно інтуїтивно зрозумілих фактора. Перший - це вплив сукупної частоти повідомлень на штраф РОШ. Зокрема, зі збільшенням кількості видавців, а отже, і сукупної частоти повідомлень теми, штраф зменшується. Повернення до еквівалентності легко пояснює, чому більш висока швидкість токенів означає менший час генерації токена (час обслуговування) і, отже, меншу затримку, як добре відомо з базової теорії масового обслуговування. Отже, хоча штраф РОШ все ще присутній, висока швидкість повідомлень означає, що ставка токенів у кожного брокера навіть після розділення трафіку 6 способи залишаються достатньо високими, щоб забезпечити порівняно невеликий штраф щодо затримки обробки повідомлень. Інший фактор, який показують цифри, це те, як вибух трафіку посилює штраф РОШ. При застосуванні до процесу прибуття партії розмір та швидкість токена зменшуються на розмір партії, що одночасно сприяє збільшенню затримки (останнє повідомлення в партії розміром 10, яке знаходить порожній сегмент токенів, чекає 10 жетонів). Саме цей коефіцієнт «посилення» відстає від значно гірших показників для пакетних надходжень.

Щоб оцінити вплив принципу max-min, повторно використовуємо сценарії, але припускаємо, що нова тема знаходить брокерів із різним уже існуючим завантаженням повідомлень. Також представляємо новий базовий алгоритм, який розширює, включаючи принцип max-min. Він націлений на розміщення нової теми з найменшою можливою кількістю брокерів, але замість того, щоб прагнути вирівняти загальне навантаження у кожного брокера, він прагне максимізувати мінімальне навантаження на тему, призначене брокеру. Наявність існуючих робочих навантажень на 6 брокерів впливає на розподіл нової теми між брокерами як функцію власного навантаження. Коли він низький, він може помістится на найбільш легкозавантаженому брокера, а решта працюють однаково. Однак продуктивність починає відхилятися, оскільки навантаження на тему зростає (за межами видавців) і його потрібно розділити на кількох брокерів.

Наприклад, коли тема може похвалитися 10 тисячами видавців, тепер її потрібно розподілити між брокерами 1, 2 та 3. Він вирівнює загальне навантаження трьох брокерів, тоді як другий підхід замість цього прагне максимізувати навантаження теми на брокера 3, який, оскільки він має найважче існуюче навантаження, отримує найменшу частку. Це призводить до 99-ї процентильної латентності 1. 50мс для видавців, призначених брокеру 3, тоді різниця в основному виникає через більший штраф РОШ. Продуктивність відображення для нової теми з видавцями, які генерують сплески з 10 повідомлень, і як працюють так само, якщо не краще, оскільки видавці генерують сплески розміром 1. Ця спочатку протилежна інтуїтивній поведінці пояснюється тим, що в цьому сценарії в продуктивності нової теми домінує обробка токенів, пов'язаних з її видавцями. У процесі швидкого прибуття видавця планується рідше, що призводить до меншої кількості одночасно активних токенів, які потрібно обслуговувати. Перейти до планувальника виконання. Таке зменшення накладних витрат на планування призводить до того, що брокер може обробляти більш високе загальне навантаження на повідомлення як за обох підходів. Ще одна цікава поведінка, яке розкривають покращення продуктивності, коли нова тема переходить від 8k до 10k видавців. Причина знову ж таки у відмінності в штрафі ДРЛ. З тисячами видавців «залишкова» кількість видавців, призначених другому брокеру, менша, ніж у видавців 10 тисяч, і, отже, штраф за РОШ вищий. Це ілюструє основну слабкість покладатися лише на концентрацію, оскільки це може призвести до залишкових призначень останньому брокеру, що призводить до дуже високих штрафів РОШ. Знову ж таки, це основна мотивація принципу. Кореляція в тому, як видавці створюють повідомлення, фіксується за допомогою кореляційних груп. Потім порівнюємо принцип з МППЗ, який включає інформацію про групу при розподілі видавців між брокерами (розглядуваний підхід не звертає уваги на цю інформацію, тоді як МППЗ прагне використовувати її для розподілу видавців між брокерами, щоб зменшити розриви прильоту пропорційно зменшенню розміру множини). На шляху до

ізоляції впливу кореляції знову припускаємо, що нова тема надходить до набору непрацюючих брокерів. Крім того, оскільки кореляційні групи можуть бути грубими, наприклад, відображаючи фізичну близькість, а не точну синхронізацію, розглядаємо два сценарії. Перший передбачає, що видавці в межах однієї групи ідеально корелюються, тобто їх час генерації повідомлень точно синхронізований, тоді як другий пом'якшує це припущення, вводячи варіації часу, коли видавці в одній групі генерують повідомлення. У цьому наборі експериментів видавці, позначені як такі, що належать до однієї групи, ідеально синхронізовані за часом генерації повідомлень. Розрізняємо розміри груп у різних експериментах, зокрема більші групи, що відповідають більшим (синхронізованим) чергам повідомлень, створеним кожною групою. Основним наслідком такого збільшення є те, що відповідність 1мс ЦРО для розриву трафіку вимагає поширення теми серед більшої кількості брокерів. Це впливає як на принцип, так і на МППЗ, але той факт, що МППЗ покладається на інформацію про групу, щоб розділити видавців з однієї групи між брокерами, дозволяє йому пом'якшити збільшення штрафу РОШ. МППЗ може отримати доступ до більшої потужності брокера і розривів повідомлень, не несучи значного збільшення штрафу РОШ. Це дозволяє йому відповідати цільовому ЦРО навіть для груп розміром 30. На відміну від цього, принцип мінімуму-максимуму змушений використовувати менше брокерів, оскільки його «сліпе» призначення видавців в кінцевому підсумку призводить до штрафу РОШ, який перевищує переваги розподілу навантаження на повідомлення теми серед більшої кількості брокерів. Переваги розподілу видавців, що враховують кореляцію, зберігаються, коли кореляційна інформація є неточною. Зокрема, час генерації повідомлень видавців у межах однієї групи тепер (рівномірно) розподілений на інтервал, а не ідеально одночасний. Зі збільшенням розміру інтервалу кореляція між видавцями слабшає. Експерименти покладалися на той самий тип видавців з розміром групи, встановленим 25, а інтервал, через який розподілялися повідомлення з тієї ж групи, коливався від 0 мс до 16 мс. Результати знову узгоджуються для періодичних видавців і

демонструють, що, принаймні, коли рівень шуму невеликий (порядку декількох відсотків середнього часу прибуття повідомлень), бурхлива кореляційна інформація все ще допомагає пом'якшити штраф РОШ. Розподіл навантаження МППЗ подано в табл. 2.1.

Таблиця 2.1 – Розподіл навантаження МППЗ

Розміри підмножин	Кількість ітерацій					
	1	2	3	4	5	6
50	1					
75	2	1				
100	4	2	1			
125	5	2	2	1		
150	6	4	2	1	1	
175	8	6	5	3	3	1

Кожен стовпець k дає кількість раундів вимірювання на цій ітерації. В останньому стовпці повідомляється загальний час розподілу навантаження, розмір інтервалу, протягом якого надходять повідомлення видавців, а саме затримка хоста, а також кількість брокерів, за якими видавці розподіляються, зменшуються як для МППЗ, так і для принципу «макс-мін». Така поведінка є прямим наслідком меншої вибуховості, пов'язаної зі збільшенням «поширення» надходжень повідомлень.

2.2 Організація затримки розподілу навантаження

МППЗ використовує лише середнє навантаження на повідомлення брокера, щоб оцінити стан залишкової потужності, доступний для кожного брокера. Цей простий підхід пояснюється тим, що складна внутрішня архітектура робить точне моделювання впливу статистики прибуття вищих замовлень складним, якщо не неможливим. Як результат, МППЗ вдається до

рішення на основі вимірювань, щоб «точно налаштувати» свої початкові (неточні) оцінки потужності та отримані видавці на розподіл брокерів. Дане точне налаштування здійснюється в онлайн-фітинговому компоненті кроку 3 механізму розподілу навантаження. Потенційною перешкодою є те, що такий підхід може зайняти час, щоб зійтися. Значні покращення можливі, якщо навантаження на теми обмежуються кількома добре зрозумілими профілями трафіку, для яких можна розробити індивідуальні моделі (як це може бути у випадку конкретних аплікацій). Така спеціалізація, однак, виходить за рамки цього оцінювання, і замість цього перейдемо до кількісної оцінки затримки розподілу навантаження в одній ілюстративній конфігурації. Зокрема, розглядаємо сценарії, пов'язані з новою темою, що надходить до набору порожніх брокерів. Це останнє припущення, можливо, спрощує процес розподілу навантаження, оскільки брокери однорідні у своїх вільних потужностях. Тим не менш, основні етапи розподілу навантаження МППЗ залишаються. Навантаження на повідомлення теми знаходиться на відносно низькому рівні видавців, кожен з яких має частоту повідомлень і розмір партії, і змінюємо її розрив, змінюючи розмір групи видавців (видавці в групі ідеально синхронізовані). Коли вибуховість низька (розмір малої групи), один брокер може пристосуватися до нової теми, але зі збільшенням вибуховості зростає і кількість необхідних брокерів, причому 6 брокерів в кінцевому підсумку потрібні, коли розмір групи досягне 30.

Оскільки завантаженість теми (частота повідомлень) низька, а брокери спочатку порожні, перше призначення на основі завжди починається з одиниці (вважається, що один брокер має достатню потужність). Залежно від вибуху теми, можуть знадобитися додаткові брокери (що призводить до збільшення брокерів). Це відображається в різних рядках табл. 2.1, де кожен рядок відповідає різному рівню розривності, а всередині даного рядка стовпець представляє одну ітерацію кроку 3 розподілу навантаження. Ітерація розпочинається з призначення видавців брокеру і прагне оцінити, чи виконується ЦРО. Якщо він є, то процес розподілу навантаження успішно завершується. Якщо

це не так, розпочинається бінарний пошук, щоб визначити максимальну кількість видавців, яку може розмістити брокер. Записи в таблиці дають кількість раундів пошуку. Кожен раунд триває 60 секунд, щоб забезпечити репрезентативну вибірку трафіку і є основним фактором затримки розподілу навантаження, загальне значення якої повідомляється в останньому стовпці. Після того, як кількість видавців, яких може розмістити брокер, визначена, надлишкові видавці призначаються наступному брокеру, а k збільшується на 1. Таблиця 2.1 показує, що зі збільшенням розривності (розміру групи) теми зростає і тривалість фази розподілу навантаження. Це очікувано, оскільки оцінка не звертає уваги на розрив робочого навантаження, і тому стає менш точною зі збільшенням. Що стосується бурхливих тем, початкові завдання систематично переоцінюють кількість видавців, які може вмістити брокер. Це запускає повторювані ітерації, кожна з яких вимагає бінарного пошуку. У «найгіршому» випадку (розмір групи 30) кумулятивний ефект цих пошуків призводить до розподілу навантаження, що перевищує 15 хвилин. Це явно довго, хоча і не нерозумно, коли з розгортанням IoT, яке може тривати тижнями або місяцями. Крім того, і як згадувалося раніше, якщо варіації параметрів профілю трафіку можуть бути обмежені, то кращі моделі для оцінки можливі і зменшать кількість ітерацій.

Розподілене обмеження швидкості (РОШ) підтримується багатьма хмарними сервісами за допомогою програмних платформ. Існуючі платформи забезпечують координаційні протоколи для адаптації параметрів обмеження швидкості на основі трафіку різних розділів робочого навантаження. Попередні дослідження розподіленого обмеження ставок, як правило, зосереджувалися на справедливості та точності обмеження розподіленої ставки. Однак попередні роботи над РОШ не розглядали вплив РОШ на затримку. Щоб зменшити експлуатацію або витрати на енергію, було проведено значний обсяг роботи з концентрації робочих навантажень на найменшій кількості серверів, що підлягають ЦРО. Він реалізує концентрацію шляхом спільного розміщення орендарів із сумісними моделями використання ресурсів, тоді як кілька інших систем покращили концентрацію шляхом спільного

розміщення чутливих до затримки робочих навантажень з інтенсивними робочими навантаженнями (пакетна обробка). Використовуємо концентрацію, щоб зменшити штраф РОШ, тобто затримки доступу до підсегмента токенів, що забезпечує обмеження розподіленої ставки. Балансування навантаження зазвичай використовується в розподілених хмарних сервісах. Загальною метою балансування навантаження є збалансування сукупного навантаження або продуктивності на кожному сервері. Балансири навантаження можуть бути реалізовані на площині даних або площині управління. Хоча балансування навантаження є ефективним підходом для досягнення справедливих затримок обробки, що недостатньо збалансувати лише сукупне навантаження за наявності РОШ. Замість цього пропонується принцип max-min, щоб збалансувати навантаження на тему, щоб уникнути надмірного штрафу РОШ. Призначення робочого навантаження з урахуванням кореляцій було досліджено в контексті хмарних обчислень. Цей підхід, як правило, вимагає врахування кореляцій у використанні ресурсів (наприклад, CPU і мережа) на основі трасування робочого навантаження. Щоб зменшити суперечки щодо процесора, спільно розташовуємо антикорельовані бази даних на основі коваріації, тоді як запропоновано консолідувати антикорельовані застосунки на основі кореляції Пірсона. Для оптимізації енергоспоживання консолідуємо слабокорельовані потоки (на основі кореляції Пірсона) в мережах центрів обробки даних. Хоча ці роботи використовують кореляцію для зниження експлуатаційних та енергетичних витрат, використовуємо кореляцію, щоб зменшити штраф РОШ. Крім того, дозволяємо користувачам вказувати групи кореляції на основі семантики застосунків. Цей практичний підхід використовує переваги характеристик домену IoT, щоб уникнути практичної проблеми вимірювання кореляції в дуже паралельних системах обміну повідомленнями. Визначення штрафу РОШ є продуктивним. Він неминуче виникає, коли обмеження швидкості розподіляється між серверами, також у розробці та розробці системи, МППЗ, здатної пом'якшити цей штраф з акцентом на його використання застосунками IoT.

Система МППЗ спирається на три основні принципи: концентрація, max-min і кореляційна обізнаність, і була оцінена емпірично на локальному випробувальному стенді. Оцінка продемонструвала його здатність успішно пом'якшити штраф РОШ, зберігаючи при цьому можливість масштабування, розподіляючи робоче навантаження між серверами, коли це необхідно. МППЗ був розроблений на основі платформи обміну повідомленнями з відкритим вихідним кодом NSQ і є загальнодоступним для використання іншими. МППЗ повністю функціонує, але покладається на рішення на основі вимірювань для точного узгодження довільного робочого навантаження з системними ресурсами. Таке зіставлення в деяких випадках може зайняти багато часу. Перспективне розширення для конфігурацій, що передбачають більш спеціалізовані робочі навантаження, передбачає розробку точного рішення для узгодження ресурсів на основі моделі, яке подолало б цю лімітацію.

2.3 Зв'язок в режимі реального часу на інфраструктурному рівні

Оскільки комп'ютерне обладнання, наприклад, багатоядерний процесор, збільшило потужність, так само збільшилася і використання технології віртуалізації в центрах обробки даних і хмарах. Цей двосторонній прогрес сприяв розвитку інфраструктури як послуги. У такому віртуалізованому середовищі застосунки/служби розгортаються у віртуальних машинах (віртуальних машинах), так що продуктивність мережевого вводу-виводу віртуальних хостів, які будують інфраструктурний рівень, стає критичною компонентою відповідності комунікаційним вимогам розподілених застосунків реального часу. Приклади таких застосувань включають обчислення, де розподілені критично важливі та критично важливі для безпеки завдання розгортаються в декількох службах і підлягають наскрізним термінам. Аналогічним чином, наскрізні обмеження затримки також присутні в корпоративних центрах обробки даних і промислових системах автоматизації, які все частіше розгортаються у віртуалізованих

середовищах. Основна проблема в таких налаштуваннях полягає в тому, що віртуальні машини запускають чутливі до затримки (м'які програми реального часу) ймовірно, будуть розгорнуті в тому ж хості, що і віртуальні машини, які запускають інтенсивні (масові) застосунки. Тому, мережеві потоки в одному віртуалізованому хості повинні спільно використовувати процесорні та мережеві ресурси. Хоча механізми спільного використання досить добре зрозумілі, споживання процесора для обробки мережевого трафіку включає в себе складний спектр взаємодій, які важче передбачити і контролювати. Це ускладнює виконання вимог затримки для розподілених програм у режимі реального часу за наявності конкуруючих програм не в режимі реального часу. У невіртуалізованих хостах, що працюють під управлінням стандартного Linux, рівень дисципліни масового обслуговування реалізує функції контролю трафіку, включаючи класифікацію трафіку, визначення пріоритетів та обмеження швидкості. У ядрі Linux передбачено кілька різних дисциплін масового обслуговування. Зокрема, такі дисципліни, які здатні пріоритезувати мережевий трафік. При поєднанні з ієрархічним сегментом токенів ці дисципліни масового обслуговування можуть досягти диференціального та ізольованого спільного використання між чутливими до затримок та нереальними потоками мережі. Ці дисципліни масового обслуговування, також, використовуються у віртуалізованих хостах на основі широко використовуваної платформи віртуалізації з відкритим вихідним кодом. Відтепер, дотримуючись термінології, використовуємо термін домен замість віртуальної машини. Він використовує керуючий домен під назвою домен 0 для управління іншими доменами (гостьовими доменами). Домен, також, відповідає за обробку мережевого трафіку від імені гостьових доменів. За замовчуванням запускає ядро Linux. Віртуалізація, однак, може ввести пріоритетні інверсії в процедурах передачі і прийому. Ці обмеження призводять до змішаного спільного використання процесора між мережевими потоками, що не може бути вирішено стандартними дисциплінами масового обслуговування Linux. Таким чином, існує ймовірність того, що чутливий до затримки трафік страждає від непередбачуваної

затримки. Управління трафіком з урахуванням віртуалізації мережевого трафіку у віртуалізованих хостах реалізовано на підході домен. Зокрема: визначає вплив механізмів контролю руху та компоненти, пов'язані з віртуалізацією в мережевому шляху; визначає обмеження мережевої архітектури, які залишають спільне використання процесора між мережевими потоками незахищеним; вводить схему управління трафіком, що враховує віртуалізацію, яка забезпечує диференційований, ізольований і справедливий розподіл ресурсів на сучасних багатоядерних процесорах, тим самим забезпечуючи більший контроль затримки та передбачуваність застосування, чутливих до латентності. Використовується керуючий домен на базі Linux для обробки мережевого трафіку з гостьових доменів і до них. Таким чином, мережевий стек схожий на стек стандартного дистрибутива Linux, але з додатковими компонентами, пов'язаними з віртуалізацією. Тому, розуміння того, якою мірою віртуалізовані платформи можуть запропонувати гарантії затримки, вимагає розуміння того, як політики та механізми Linux, включаючи дисципліни масового обслуговування, спільне використання черг передачі та прийому, а також частоту, з якою переривання (сповіщення) генеруються та обслуговуються, взаємодіють у віртуалізованому середовищі. Таким чином, полягає в тому, щоб запропонувати ретельне вивчення взаємодій і того, як вони впливають на затримку при різних конфігураціях трафіку. Розглянемо стандартні процедури передачі та прийому пакетів Linux, які були стабільними з версії 2. 6. Мережевий стек в стандартному Linux зображено схемою на рис. 2.2.

На рис. 2.2 показані підпрограми передачі та прийому в мережі в стандартному Linux. У стандарті Linux пакети з застосунків обробляються мережевим стеком в ядрі Linux. Оскільки віртуалізація-розширення Linux змінюють лише канальний рівень, опускаємо сесійний, транспортний та мережевий рівні на рисунку. Пакети ставляться в чергу у відповідну чергу дисципліни черги на канальному рівні, де Linux реалізує контроль трафіку. Черга драйверів, також відома як кільцевий буфер, черга FIFO, яка тісно взаємодіє з ним.

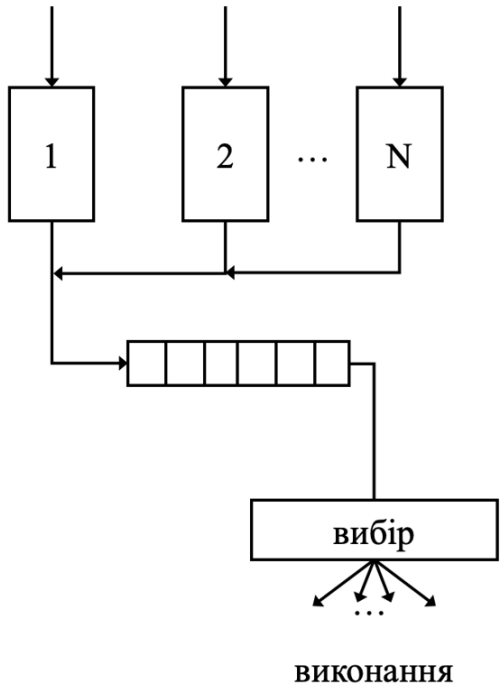


Рисунок 2.2 - Передача/прийом у стандартному середовищі ОС Linux

Дисципліна масового обслуговування: рівень реалізує класифікацію трафіку, визначення пріоритетів та обмеження швидкості. Ці налаштування дозволяють досягти диференційованого та ізольованого спільного використання мережевих мереж між мережевими потоками. У Linux параметри налаштовуються за допомогою команди. За замовчуванням Linux використовує дисципліну черги для контролю руху. Залежно від конфігурації, Linux може пріоритезувати пакети та зменшити затримку черги для застосунків, чутливих до затримок. Дисципліна масового обслуговування, яка має одну чергу на пріоритет. Він працює у співпраці з пакетними фільтрами, які розподіляють пакети з різних потоків (застосунків) по різних чергах. Коли викликається функція черги, то порядок, в якому пакети виводяться з черги, переходить від високого пріоритету до низькопріоритетного. Отже, призначення програм, чутливих до затримок, до черги з найвищим пріоритетом, може забезпечити коротші затримки в черзі. Ще одна дисципліна масового обслуговування, яка працює над зменшенням затримки в черзі має одну чергу на потік, з квантом для кожної черги. Як тільки квант досягнутий, то відповідна черга класифікується як черга з негативним дефіцитом, яка має низький пріоритет. Таким чином, ця політика пропонує

короткі затримки в черзі для застосунків, чутливих до затримок, з низькою пропускною здатністю. На додаток до цих параметрів диференціації, рівень надає ієрархічну множину маркера для формування кожного мережевого потоку. Цей параметр реалізує ізольований обмін, який захищає від затоплення потоками (навмисно чи випадково) неправильної поведінки. Коли використовується дисципліна диференційованого масового обслуговування, важливо, щоб потоки високої пріоритетності відповідно регулювалися НТВ. В іншому випадку потоки з низьким пріоритетом легко страждають від надзвичайного голоду через неправильне поводження з високопріоритетним трафіком. Пакети залишаються в очікуванні в черзі драйверів до наступного кроку. Таким чином, перевантаження в черзі може мати критичний вплив на затримки передачі пакетів. Перевантаження виникає, коли занадто багато великих пакетів пересилається і апаратне забезпечення не здатне обробляти їх досить швидко. Зазвичай існує обмеження на розмір черги драйверів, яка контролює кількість відкладених пакетів. Однак цього контролю недостатньо для запобігання заторів, коли основна частина трафіку NIC складається з великих пакетів. Це обмеження було вирішено в останніх ядрах Linux шляхом введення політики обмеження байтової черги, яка обмежує кількість байтів у черзі драйверів. У співпраці з рівнем, може значно зменшити чергу затримки в черзі драйверів, навіть при наявності великих пакетів. Розмір черги драйверів динамічно обмежений, виходячи з режиму руху і пропускної здатності. Як тільки розмір черги досягає межі, шар містить або скидає наступні пакети. У більшості драйверів мереж, коли пакети успішно відправляються, спрацьовує завершення драйвера. Обробник переривань поміщає пристрій (програмна структура даних, що представляє драйвер) у список опитувань, який є структурою даних на процесор у Linux. В кінці обробника переривань піднімається програмне переривання, обробник якого обслуговує список опитувань. Обробник обробляє мережеві пристрої в списку опитування в круговому порядку, з квантом 64 пакетів. Коли пристрій отримано, обробник викликає метод драйвера. Залежно від драйвера, метод опитування може виконувати дії. У драйвері, який використовується в експериментах, метод

очищає чергу драйверів і отримує пакети з другої черги драйверів. Інші мережеві драйвери мають окремі переривання. Обробник переривань завершення драйверу і очищує чергу драйверів, а обробник переривань піднімає новий драйвер. Метод опитування цих драйверів робить тільки прийом пакетів. Як тільки розмір черги стає нижче обмеження, обробник переривань повідомляє про відновлення випуску пакетів в чергу драйверів. Інтервал між перервами завершення може бути налаштований.

У кластерах і центрах обробки даних, де зв'язок з низькою затримкою життєво важливий, користувачі, як правило, налаштовують невеликий інтервал. Однак занадто часті переривання можуть генерувати великі робочі навантаження на процесор і негативно впливати на прогрес процедур передачі та прийому пакетів. І навпаки, якщо інтервал переривання занадто великий, черга драйверів може стати перевантаженою, оскільки вона оновлюється недостатньо часто. У цьому випадку пакети залишаються в очікуванні на шарі і можуть відчувати тривалі затримки в черзі. Кілька постачальників драйверів пропонують динамічні частоти переривань, які регулюють значення інтервалу залежно від того, чи є трафік низькозатримковим або масовим.

На рис. 2.2 також показано прийом мережі в стандартному Linux, в якому надходження пакетів апаратно переривається, а обробник переривань потім поміщає той самий мережевий робочий пристрій в список опитування і піднімає. Коли обробник отримує і викликає відповідний метод, пакети доставляються з черги драйверів на верхній шар. Символ функції обробника закінчується, коли або жоден пристрій у списку опитування не має пакетів, що очікують на розгляд, або він обслужив велику кількість пакетів. Потік покладається на домен менеджера. Домен 0 для обробки мережевого трафіку. Мережевий стек аналогічний стеку стандартного Linux. На рис. 2.2 показані процедури передачі і прийому драйверів в чергу.

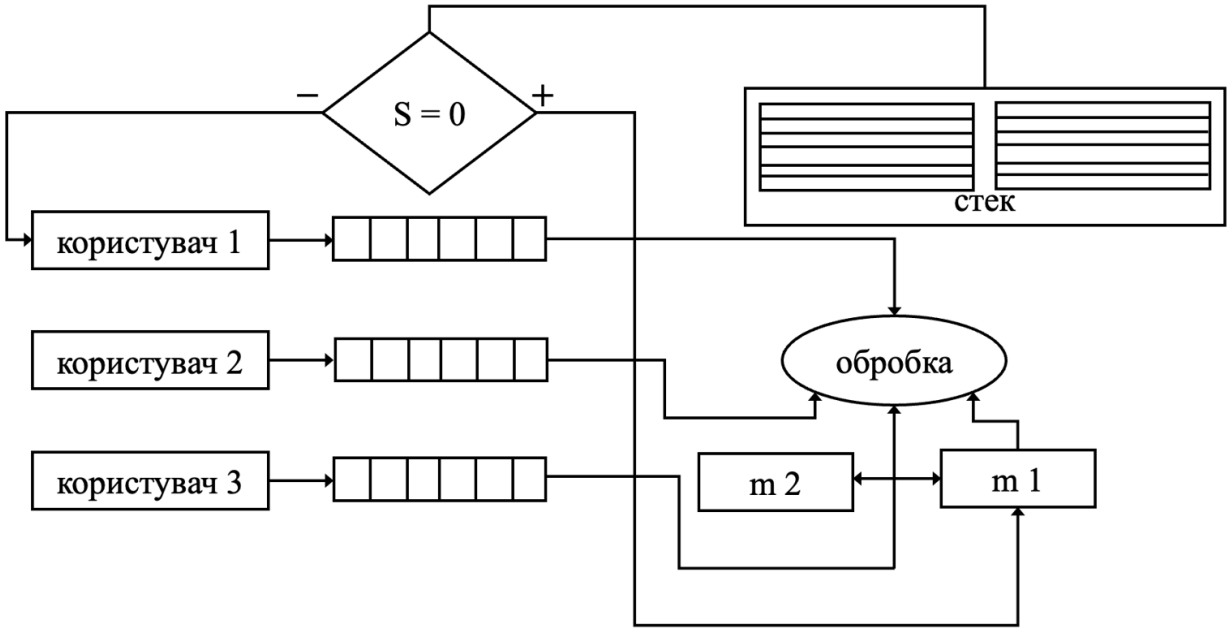


Рисунок 2.2 - Передача/прийом драйверів в чергу

Кожен драйвер є в окремій черзі і йому виділено спеціальний потік ядра для прийому. Коли гостьовий домен має пакет для передачі, то він спочатку повідомляє системі. Потім обробник сповіщень вставляє відповідний пристрій у список опитування та піднімає. Коли обробник запланований, він обробляє всі пристрої в єписку опитування так само, як і в стандартному Linux. Після закінчення роботи функції обробника обробляються інші відкладені драйвери. Якщо обробник сповіщень піднімає до завершення обробки, функція обробника викликається знову після обробки інших незавершених подій. У ситуаціях, коли часто піднімається задача, тому процес може працювати безперервно протягом тривалого періоду часу. Коли надходить пакет, обробник апаратних переривань вставляє пристрій в список опитувань і піднімає очікуючу задачу. Потім обробник доставляє пакети з черги драйверів до черги пристрою призначення. Кожен пристрій має відповідний потік ядра прийому. Коли пакети вставляються в чергу, також запускається відповідний потік. Потім пакети пересилаються з черги до гостьового домену, коли заплановано цей потік. Однак це потрібно почекати, поки обробка не закінчиться, що може спричинити затримки.

У стандартному Linux обробка мережевого трафіку вимагає ресурсів. Отже, коли мережеві потоки, чутливі до затримок, стикаються з конкурентами

не в режимі реального часу (інтенсивними даними), механізми контролю трафіку повинні мати можливість диференціювати та ізолювати (обмеження швидкості) як спільне використання, так і процесора. Однак механізми управління трафіком, які покладаються на налаштування, можуть лише диференціювати та ізолювати спільне використання мереж, залишаючи спільний доступ до процесора незахищеним. Коли чутливі до затримки домени, так і інші інтерферуючі домени передають пакети, всі їх пристрої вставляються в список опитувань і обслуговуються в круговому порядку. Таким чином, пристрій, що містить пакети, чутливі до затримки, може бути відкладений іншими пристроями. Квант за замовчуванням для кожного мережевого пристрою в списку опитування 64 (пакети). Повторне зведення кванту може зняти цю інверсію пріоритету. Однак, є ще одне обмеження, яке цей підхід не може вирішити. Коли обробник пересилає пакет на пристрій, він пробуджує відповідний потік для виконання подальших завдань. Однак можна запланувати лише після завершення обробки. У ситуаціях, пов'язаних з процесором, коли багато доменів (не в режимі реального часу) надсилають пакети з досить високою швидкістю, то може часто підніматися черга очікуючих завдань (за допомогою обробника сповіщень), так що обробник безперервно обслуговує список опитувань, який може надовго затримати запуск потоків. Ця інверсія пріоритету виникає між передачею і прийомом. Просте зменшення кванту для кожного мережевого пристрою в списку опитувань не може вирішити цю інверсію пріоритетів, оскільки тривалість, протягом якої виконується обробка, не залежить від квантового значення. Легко знайти існуючі дисципліни масового обслуговування, які успадковані від Linux, не можуть вирішити ці обмеження, оскільки ці налаштування призначені для спільного використання, тоді як проблеми затримки знаходяться в компонентах, пов'язаних з віртуалізацією, які вимагають ресурсу процесора. Тому, пропонується схема контролю трафіку з урахуванням віртуалізації, в якій мережеві потоки мають пріоритет і

швидкість обмежена для всіх учасників мережі, тим самим досягаючи диференційованого та ізольованого спільного використання ресурсів у dom0.

2.4 Висновок до розділу 2

В результаті пропонується система контролю трафіку, що враховує віртуалізацію, яка вирішує ці обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. Додатково СКТВ масштабується до багатоядерних процесорів і забезпечує справедливий механізм розподілу навантаження. Завдяки СКТВ контролюється використання процесора. Тоді, можна досягти повної структури, яка забезпечує диференційовану та ізольовану обробку трафіку у всіх компонентах мережі. У СКТВ ребалансування призначень запускається тільки в момент створення або завершення роботи гостьового домену. Але під час звичайного часу виконання варіація трафіку гостьових доменів все ще може призвести до несправедливого розподілу навантаження між ядрами процесора домена. Одним з потенційних рішень є покладання на балансер для балансування сповіщень, які надіслані з гостьових доменів, між ядрами процесора під час виконання. У ньому сповіщення розглядаються як апаратні переривання, тому вони мають конфігурації спорідненості, які визначають, які ядра процесора обробляють їх. Можна прихованою програмою динамічно переналаштовувати спорідненість кожного переривання та сповіщення таким чином, щоб обробка переривань/сповіщень була збалансована між ядрами процесора. Щоб завершити таку дрібнозернисту адаптацію у СКТВ, кожного разу, коли спорідненість сповіщення переналаштовується, відповідний пристрій повинен бути відповідно перепризначений. Впровадження та порівняння цього рішення з поточним механізмом ребалансування може бути цінним розширенням СКТВ. На рівні інфраструктури розроблено систему управління трафіком, що враховує віртуалізацію у віртуалізованих хостах. Також, визначено обмеження механізмів контролю трафіку дисципліни масового обслуговування Linux, які залишають

спільне використання процесора між мережевими потоками незахищеним. Щоб пом'якшити ці обмеження, СКТВ надає нову архітектуру мережевого вводу-виводу, яка забезпечує диференціацію затримки використання процесора за рахунок пріоритетної обробки пакетів. Крім того, СКТВ має ізоляцію послуг за допомогою обмеження швидкості та масштабованість за рахунок справедливого розподілу навантаження між багатоядерними процесорами.

3 РОЗРОБКА ТА ВПРОВАДЖЕННЯ МАСШТАБОВАНОГО ПРОМІЖНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальні вимоги до розробки масштабованого проміжного програмного забезпечення

Найпростіший спосіб пом'якшити інверсії пріоритетів серед потоків передачі - поширити обізнаність про пріоритети на пристрої у списку опитувань. Однак інверсії пріоритетів між передачею та прийомом обумовлені перешкодами між обробкою подій та потоків. В результаті, замість того, щоб впроваджувати один пріоритетний планувальник для передачі пакетів у компонентах, пов'язаних з віртуалізацією, СКТВ призначений для забезпечення дрібнозернистого контролю трафіку на основі потоків ядра. У Linux користувачі можуть налаштувати як політику планування, так і пріоритет потоків ядра. FIFO - це попереджувальна політика планування з фіксованим пріоритетом, згідно з якою потік із високим пріоритетом може випереджати запущений потік із низьким пріоритетом. СКТВ спирається на цю концепцію, призначаючи мережевий трафік доменів з високим пріоритетом потокам ядра, а мережевий трафік низькопріоритетних доменів потокам ядра з низьким пріоритетом. Крім того, схожа до диференційованого та ізольованого спільного використання СКТВ забезпечує механізм обмеження швидкості, який використовує процесор потоків з високим пріоритетом. Без цього механізму трафік з низьким пріоритетом вразливий до заповнення процесора малою кількістю завдань. Тому, диференціація, так і обмеження швидкості вперше представлені в єдиному середовищі процесора. Але, СКТВ масштабується до багатоядерного процесора та вводить додаткові проблеми розподілу навантаження. Він представляє кілька (програмних) пристроїв зворотного зв'язку і, відповідно, кілька потоків ядра для обробки передачі пакетів і прийомів до/з різних доменів. Ці потоки налаштовані з різними пріоритетами. Гостьові домени з однаковим пріоритетом (однакові вимоги до затримки) використовують один і той самий пристрій. Усі потоки

заплановано відповідно до політики FIFO. Кількість пристроїв можна налаштувати на основі кількості необхідних рівнів пріоритету. Використано два рівні пріоритету. Домени запускаються в режимі реального часу (з урахуванням затримки) і призначаються нотоку з високим пріоритетом. Також, наявні домени з низьким пріоритетом, що працюють з інтенсивною пропускною здатністю (не в режимі реального часу) прикладними катіонами. Високопріоритетний потік обробляє мережевий трафік доменів із високим пріоритетом, а низькопріоритетний обробляє трафік доменів із низьким пріоритетом. Потік, який запускається завершенням та обробником переривань, має найвищий пріоритет. Він очищає всі пакети, які були передані з черги драйверів і обробляє пакети в черзі драйверів. Оскільки як передача, так і прийом обробляються потоками ядра, видаляємо список опитувань та обробку програмних переривань з СКТВ. Далі розглянемо передачу і прийом пакетів, а також їх взаємодії. Коли домен з високим пріоритетом має пакети для відправки, то він повідомляє домен. Обробник сповіщень запускає відповідний високопріоритетний потік, який може вивільняти потоки з нижчим пріоритетом. Пакети з високопріоритетного домену спочатку ставляться в чергу в черзі відповідного пристрою. Потім потік перевіряє досягнення обмеження черги драйверів. Якщо вона є, тобто черга драйверів перевантажена, вона призупиняється, доки потік не очистить чергу драйверів і не оновить розмір черги. Після очищення черги драйверів потік повідомляє призупинений потік про відновлення (якщо новий розмір черги нижче обмеження). Якщо кілька потоків призупинено, то вони відновляться один за одним, виходячи зі своїх пріоритетів. Кожен може обробляти пакети в черзі у порядку FIFO, оскільки домени джерела цих пакетів мають однаковий пріоритет. Пакети проходять через найвищий рівень і, нарешті, потрапляють в чергу драйверів. Якщо більше немає пакетів з будь-якого домену з високим пріоритетом для черги, а черга порожня, то потік з високим пріоритетом зупиняється, дозволяючи запускати потік з нижчим пріоритетом. СКТВ видаляє його, оскільки вони можуть призвести до інверсії пріоритету, коли потік з високим пріоритетом

повинен обробляти повідомлення, підняті потоками з низьким пріоритетом. Тому, СКТВ повністю обробляє пакети у потоках та мережевому потоці. Розглянемо прийом пакетів у СКТВ. Коли пакети надходять на віртуалізований хост, то обробник апаратних переривань пробуджує мережевий потік для їх обробки. Після того, як потік запланово, то він забирає пакети з черги драйверів і пересилає їх до черги пристрою призначення. Для черги пристрою кожна операція черги пробуджує відповідний потік, який буде заплановано після завершення роботи потоку. Якщо потоки пробуджуються мережевим пристроєм, то вони будуть заплановані на основі їх пріоритетів. Коли заплановано потік, то він доставляє пакети з відповідної черги до цільових доменів. Перешкоди між передачею та прийомом в оригінальному домені для повторного сприйняття може бути попереджено обробником для передачі. У СКТВ передача/прийом трафіку в режимі реального часу обробляється потоками ядра з високим пріоритетом. Отже, перешкоди від передачі або прийому трафіку не в реальному часі обробляється потоками ядра з меншим пріоритетом значно зменшуються. СКТВ досягає диференціації затримки за допомогою пріоритетних потоків ядра. Однак необмежені високопріоритетні мережеві потоки можуть легко голодувати через потоки з низьким пріоритетом. Функція обмеження швидкості СКТВ усуває голодання процесора між пріоритетами. Оскільки обхід мережевого стека Linux (програмного забезпечення) є операцією на пакет, обмеження швидкості пакетів стає ключем до використання процесора. Оскільки мережева архітектура доменів була змінена, то використаємо наступний експеримент для повторної перевірки впливу швидкості пакетів на використання процесора. У експерименті одне ядро процесора домена. У гостьових доменах (на інших ядрах) використано різні розміри пакетів UDP для генерації мережевих потоків, які насичують процесор домена. Максимальна пропускна здатність, так і швидкість пакетів цього одноядерного домена зі збільшенням розміру пакетів зростає стрімко, в той час як максимальна швидкість пакетів залишається відносно постійною. Ці

результати підтверджують, що швидкість пакетів визначає вузьке місце процесора домена, що узгоджується з досвідом роботи в оригінальному ящику Linux. Тому, щоб обмежити споживання процесором мережних потоків, то потрібно використовувати обмежувачі швидкості на основі пакетів, замість байтових. В оригінальному домені користувачі можуть налаштувати обмежувач швидкості для кожного гостьового домену. Однак його реалізація має два обмеження. Цей обмежувач швидкості заснований на байтах, що не може ефективно розподіляти споживання процесором мережевих потоків. Він відрізняється від сегмента токенів. Цей обмежувач швидкості не має незалежного параметра для допустимої серії, що призводить до втрати трафіку. Щоб пом'якшити ці два обмеження, обмеження ставки СКТВ реалізується шляхом додавання одного сегмента токенів на основі пакетів до кожного високопріоритетного пристрою. У високопріоритетному потоці пристрою, трафік з кожного домена повинен проходити через відповідний сегмент токена, де кожен пакет вимагає токена. При відсутності прибуття пакету другий пакет повинен чекати одного. Параметри сегмента токена на основі пакетів визначаються ЦРО-контрактом відповідного гостьового домену. Обмежуючи частоту пакетів доменів з високим пріоритетом, можна усунути голодування процесора від трафіку з низьким пріоритетом. У СКТВ сегменти токенів на основі пакетів застосовуються лише до доменів із високим пріоритетом. Якщо трафік із високим пріоритетом обмежений, трафік із низьким пріоритетом може вільно використовувати весь залишковий ресурс процесора. Сегменти токенів на основі пакетів спрямовані лише на розподіл споживання процесора, тоді як користувачі можуть одночасно використовувати НТВ на рівні для контролю використання.

Розглянемо багатоядерну підтримку СКТВ. Оскільки сучасні багатоядерні процесори широко використовуються у віртуалізованих хостах, стає загальним, що домени виконується на декількох ядрах. СКТВ масштабується до багатоядерного домену. Оскільки більшість операцій передачі та прийому мережі обробляються пристроями зворотного зв'язку, масштабування до декількох

ядер вимагає лише відповідного дублювання пристроїв. Якщо домен має N ядер процесора, то СКТВ створює N пристроїв для кожного пріоритету, при цьому кожен пристрій віднесений одному ядру. Крім того, СКТВ створює один чистий потік на ядро для поліпшення масштабованості прийому пакетів з мережевої мережі. За допомогою цих багатоядерних налаштувань СКТВ зберігає кількість (пріоритетних) пристроїв на кожному ядрі рівною кількості рівнів пріоритету, тим самим диференціація латентності все ще зберігається. Тим часом, обмеження швидкості все ще може бути гарантовано сегментами токенів на основі пакетів без будь-яких змін. Оскільки пристрої дублюються, СКТВ тепер стикається з новою проблемою, суть якої в тому що в рамках пріоритету, як справедливо розподілити навантаження трафіку між пристроями N (на ядро). Разом з архітектурою СКТВ, рішення спирається на те, як пристрої справедливо призначаються пристроям. Пристрій СКТВ розроблено на базі драйвера. У драйвері призначаються потоки згідно з політикою найменшого номера. Після створення нового домену він назавжди призначається тому який з найменшою кількістю. Однак, це призначення має два обмеження, які можуть призвести до несправедливості. Він ігнорує той факт, що пристрої можуть мати різне навантаження на мережевий трафік. Завдання - це одноразове рішення. Тому, враховуючи групу гостьових доменів, різні послідовності створення домену можуть призвести до призначень з надзвичайно різною справедливістю. Тому, пропонується використовувати механізм ребалансування для справедливості розподілу навантаження. У межах кожного пріоритету підтримуємо відсортований список зі зменшенням порядку навантаження на трафік. Після створення та закриття домену, що призводить до потенційної несправедливості оновлюємо відсортований список, вставляючи або видаляючи токени. Після оновлення відновлюємо баланс призначень, запустивши простий жадібний алгоритм, який коригує призначення по одному. При виборі пристрою використовуємо політику. У більшості віртуалізованих хостів створення і відключення гостьових доменів є рідкісними випадками. Тому, цей перебаланс, як правило, запускається з

обмеженим часом. Під час перебалансування міграція займає лише менше 1 наносекунди, виходячи з спостереження. Отже, загальні накладні витрати цього механізму перебалансування обмежені. Оцінювання трьох особливостей ВАТС здійснимо за такими параметрами: диференціювання латентності; обмеження тарифів; багатоядерна підтримка.

Різні фактори можуть затримувати м'який трафік у реальному часі. Оцінюємо затримку та передбачуваність латентності для чутливого до затримки трафіку в рамках реалізації СКТВ та існуючих механізмів контролю трафіку, тобто в оригінальному домені. Оцінка проводиться на випробувальному стенді, що складається з фізичних машин. П'ять інших фізичних машин, розміщених від 1 до 5, працюють під управлінням стандартного Linux. Всі машини оснащені гігабітними мережевими мережами і з'єднані гігабітним комутатором. Оскільки є дрібнозернисті планувальники пакетів, то це гарантує, що великі пакети з розміром, більшим, ніж байт в системі, сегментуються в ядрі, і це дозволяє уникнути довгих затримок блокування в черзі драйверів. Драйвер використовує метод, який викликається обробником, для очищення черги драйверів. Хост отримує одне виділене фізичне ядро процесора. Це звичайна практика обробки зв'язку та переривань, а також рекомендована спільнотою для покращення продуктивності вводу-виводу. Завантажуємо п'ять гостьових доменів, домен 1 до домену 5 на хості 0. Кожен з них закріплений на окремому фізичному ядрі процесора, щоб уникнути впливів з боку планувальника віртуальних машин. У налаштуваннях домен 1 є доменом, чутливим до затримки, а домени від 2 до 5 - доменами, що заважають. Отже, в рамках СКТВ трафік з/до домену 1 обробляється потоком ядра з високим пріоритетом в домені, тоді як трафік, що належить до доменів від 2 до 5, обробляється низькопріоритетним. Затримка в обидва кінці між доменом 1 і хостом 1 вимірюється наступним чином. Пінг домену 1 (з пакетами ICMP) розміщується в один кожні 10 мс, а хост 1 відповідає назад. Ця схема трафіку спрямована на емуляцію поведінки поширених періодичних застосунків у реальному часі. У кожному експерименті записуються значення затримки для 200

пар запит/відповідь ICMP. Затримка хоста важлива для багатьох програм м'якого реального часу, оскільки вона відображає передбачуваність затримки. У домені від 3 до домену 5 запускаємо потоковий тест для моделювання застосунків не реального часу. Драйвер також забезпечує два адаптивних режими: динамічний консервативний і динамічний. Обидва режими динамічно регулюють інтервал переривання на основі типу мережевого трафіку: масового або інтерактивного. Динамічний консервативний режим є режимом за замовчуванням драйвера. Оцінюємо як режими, так і діапазон статичних значень. Затримка високопріоритетного чутливого до затримки трафіку вимірюється в сценаріях з контенцією процесора в домені. Суперечки між процесором можуть виникнути, коли домени з низьким пріоритетом надсилають багато невеликих пакетів. У таких сценаріях часто спрацьовує обробник. Цей вплив можна посилити, встановивши інтервал обробника переривань на невелике значення. У наступному експерименті оцінюємо затримку трафіку з високим пріоритетом за наявності перешкод для потоків з низьким пріоритетом, що складаються з невеликих UDP. Для того щоб виділити вплив різних інтервалів переривання, зосередимося на випадку одного заважає потоку. Оскільки пакети невеликі, то динамічний консервативний режим і динамічний режим, як правило, за замовчуванням встановлюють інтервал переривань на нижню межу свого діапазону.

Затримка за всіма трьома механізмами управління дорожнім рухом зростає зі збільшенням інтервалу переривання.

Це пов'язано з тим, що переривання запускаються рідше з довшими інтервалами переривань. Зростання кількості ядер в одному хості означає, що кілька гостьових доменів можуть співіснувати в межах одного хоста.

Тому, підтримка диференціації затримки з більш, ніж однією областю є важливою проблемою масштабованості.

3.2 Масштабоване проміжне програмне забезпечення

Розглянемо затримку потоку з високим пріоритетом (ICMP) зі збільшенням кількості доменів, що заважають. На продуктивність затримки BATC не впливає кількість потоків, що заважають, оскільки високопріоритетний потік, що обробляє трафік у режимі реального часу, не може бути попереджений нижчими пріоритетними. Іншими словами, СКТВ успішно пом'якшує обмеження 1 і обмеження 2. На відміну від цього, потік з високим пріоритетом збільшує затримку із кількістю потоків, що заважають. Різні компоненти домену сприяють збільшенню затримки пакетів з високим пріоритетом. У конфігураціях затримка в черзі може виникнути коли обробник обслуговує інші пристрої у списку опитувань. Пакети з високим пріоритетом очікують на розгляд у відповідному пристрої, коли в черзі драйверів є перевантаження. Пакети залишаються в очікуванні на рівні. Після того, як пакети відповідей ICMP будуть переадресовані до черги, відповідний потік повинен дочекатися завершення обробки, перш ніж його можна буде запланувати. Зі збільшенням кількості доменів, що заважають, збільшується і кількість пристроїв, вставлених у список опитувань, що сприяє невеликому, але постійному збільшенню затримки. Затримка на шарі також збільшується, оскільки тепер у черзі драйверів більше відкладених пакетів, а обробка переривання завершення у нижній половині затримується, оскільки в драйвері обробник переривань завершення просто вставляє пристрій в список опитувань і залишає обробку нижньої половини (очищення черги драйверів) для виконання обробником нижнього рівня. Зі збільшенням кількості конкуруючих пристроїв (у списку опитувань) пристрій обслуговується рідше. Це призводить до більшої завантаженості в черзі драйверів і, отже, до довших затримок у черзі на рівні. У деяких інших драйверах, які очищають чергу драйверів в апаратному обробнику переривань, пристрої в списку опитувань не будуть затримувати очищення черги драйверів, таким чином, затримка черги на рівні може бути зменшена. Обидва ці внески в більш високу затримку можна віднести до

обмеження 1, але можна побачити, що обмеження 2 має ще більш виражений ефект. Два потоки, що заважають, суттєво впливають на затримку черги. Це викликано тим, що обробник неодноразово обслуговує список опитувань, коли потік часто піднімається обробниками сповіщень та переривань. Це може призвести до затримки на непередбачувано довгий час, який фіксує хвіст розподілу затримки. Ця тенденція дещо змінюється, оскільки кількість потоків, що заважають, збільшується далі за межі 2, оскільки домен піднімається рідше, оскільки додається більше потоків з низьким пріоритетом. У експериментах велика частина доменів піднімається обробниками сповіщень з гостьових доменів. Оскільки процесор домену перевантажений, то не всі пакети з низьким пріоритетом можуть бути обслужені вчасно, і відставання пакетів накопичується в буферах між пристроями і відповідною низькопріоритетною гостьовою мережею. Це відставання не дозволяє відповідним гостьовим доменам розміщувати більше пакетів у буфері, і, таким чином, нові сповіщення не видаються доменом, поки буфер не оновиться. Але він зберігає загальну пропускну здатність заважаючого трафіку постійною та рівномірно розподіленою між потоками на відміну від кожного потоку, який вносить свій власний незалежний обсяг трафіку. Це в значній мірі виключає можливість перевантаження в буфері між кожним доменом і гостьовим доменом. Отже, частота повідомлень з гостьових доменів набагато вище. Наприклад, з 4 потоками, що заважають, вимірюємо частоту без, яка в 100 разів більша, ніж при тій же кількості потоків, кожен з яких вносить свій власний трафік. Ця різниця значною мірою відповідальна за значне збільшення латентності, що спостерігається між найгіршою затримкою. Хоча затримка спочатку відчуває збільшення з додаванням більшої кількості потоків, додавання четвертого потоку, здається, сприяє невеликому зменшенню. Не можна точно визначити джерела зниження, але припустили, що це може бути частково пов'язано з тим, що деякі потоки зараз не завжди мають нові пакети, що, в свою чергу, знизило б швидкість сповіщень. Чутливий до затримки трафік незахищений в домені. СКТВ долає ці обмеження, виділяючи пристрій і пріоритетний потік

ядра для кожного рівня пріоритету, так що потоки в режимі реального часу гарантуються ресурсом процесора і захищені від перешкод трафіку.

Щоб уникнути голодання між пріоритетами, СКТВ вводить обмеження швидкості, додаючи сегменти токенів на основі пакетів до доменів з високим пріоритетом. У кожному з доменів з високим пріоритетом запускаємо токени для генерації мережевого трафіку UDP з фіксованою пропускною здатністю на рівні 50 Мбіт/с. Далі змінюємо розмір пакета, щоб генерувалася інша швидкість пакета. У домені з низьким пріоритетом запускаємо токен для вимірювання пропускної здатності UDP/TCP. Вісь абсцис показує різну загальну швидкість пакетів, що генерується двома доменами з високим пріоритетом. Під тиском високопріоритетного трафіку оцінюємо пропускну здатність UDP і TCP домену з низьким пріоритетом. Без сегментів токенів на основі пакетів пропускна здатність (UDP/TCP) домену з низьким пріоритетом різко падає, коли швидкість пакетів з високим пріоритетом перевищує 100 к/с. Ця деградація викликана голоданням процесора. З іншого боку, якщо додати сегмент токенів на основі пакетів для кожного домену з високим пріоритетом, то ресурс процесора, споживаний трафіком з високим пріоритетом, відповідно розподіляється, тим самим отримуючи трафік з низьким пріоритетом, за достатню кількість циклів процесора. Отже, можемо виявити, що пропускна здатність UDP/TCP домену з низьким пріоритетом успішно захищена.

Розглянемо питання справедливості розподілу навантаження статичного призначення та механізму ребалансування. Залежно від послідовності створення домену, метод може призначати навантаження трафіку з вкрай різною справедливістю. У кращому випадку метод може призначити навантаження на трафік, а іншу половину на інше ядро. У гіршому випадку метод може призначити навантаження трафіку на перше ядро, а частину, що залишилася на друге ядро. У кожному випадку відповідно оцінюємо 99-й процентиль затримки мережевого трафіку, призначеного першому і другому ядрам. Кожна оцінка повторюється 10 разів, тому покажемо її як середню затримку за маркерами, так і стандартне відхилення за планками похибок. При використанні статичного призначення

мережевий трафік на різних ядрах може показувати істотно різну затримку через несправедливий розподіл навантаження. У гіршому випадку затримка мережевого трафіку, призначеного першому ядру більше, ніж на 50 відсотків, ніж в іншому. З іншої сторони, завдяки механізму ребалансування, СКТВ може досягти справедливого призначення таким чином, що два ядра показують однакову продуктивність. Оскільки м'які програми реального часу широко використовуються на віртуалізованих платформах, захист чутливого до затримки трафіку став важливою темою. Управління мережевим входом/виходом може резервувати ресурси вводу-виводу (наприклад, пропускну здатність мережі) для критично важливого для бізнесу трафіку на основі визначених користувачем множини мережевих ресурсів. У Windows Server також забезпечено управління пропускною здатністю мережевого трафіку. У середовищах з мережевими суперечками вони можуть ефективно підвищити продуктивність чутливих до затримки віртуальних машин. Однак, оскільки вони зосереджені на управлінні шириною смуги, то вони можуть не ефективно обробляти інверсії пріоритетів, викликані суперечками процесора, який знаходиться в центрі уваги СКТВ. Тому, існуючі підходи до управління пропускною здатністю та СКТВ є взаємодоповнюючими рішеннями для мережевих та процесорних суперечок сценаріїв відповідно. Він створює кілька потоків хоста для обробки трафіку з різних гостьових віртуальних машин. Однак різним потокам не призначаються пріоритети, що відповідають пріоритетам віртуальних машин. Крім того, оскільки трафік служби потоку, як у стандартному Linux. Він може мати подібні проблеми з інверсією пріоритетів. Наприклад, потік, що обслуговує трафік у режимі реального часу, може бути попереджений потоками для трафіку не в реальному часі або обробниками. Оптимізацію мережевого стека домену можна досліджувати шляхом фрагментації великих пакетів на малі, щоб потоки могли працювати більш ефективно, щоб зменшити затримку в черзі. На додаток до цих модифікацій мережевого стека, також оптимізовано планувальник та мережевий комутатор, щоб ще більше зменшити затримку хоста в налаштуваннях центру обробки даних. При цьому не враховано затримки в чергах на рівні віртуалізації

доменів, тобто пристроїв, які можуть відігравати значну роль забезпечуючи структуру планування в реальному часі. Вона нещодавно включена і реалізує планування пакетів з урахуванням пріоритетності в пристрої. Завдяки їй можуть бути гарантії в режимі реального часу для локальних міждоменних комунікацій. СКТВ прагне поширити ці гарантії на зв'язок з віддаленими хостами. Інша пов'язана тема полягає в тому, як підвищити ефективність комунікування гостьових доменів, виділяючи додаткові ядра для кожного домену. Тому, покращити продуктивність вводу-виводу гостьового домену з декількома процесорами можна делегуючи всю його обробку вводу-виводу спеціальному процесору. Завдяки наявності виділеного процесора, гостьовий домен може ефективніше обробляти переривання з обмеженими накладними витратами на процесор. Аналогічно, він пропонує ефективну та масштабовану паравіртуальну систему вводу-виводу, реалізуючи дрібнозернисте планування вводу-виводу та модель сповіщення про запит/відповідь без виходу вище. Жодна з цих двох систем не прагне визначити пріоритетність мережевого трафіку з різними вимогами в режимі реального часу. Їх мета - підвищити середню продуктивність мережі у віртуалізованих хостах. Мережі, що підтримують механізми пропуску для обходу рівня віртуалізації мережі та доменів для зменшення затримки мережі та спеціалізованій апаратній підтримці мережевого зв'язку, були підтримані комерційними платформами віртуалізації. Крім того, СКТВ не вимагає спеціальної апаратної підтримки. Вона реалізує обмежувачі швидкості та планувальники передачі в апаратному забезпеченні. У той час призначена для поліпшення масштабованості і продуктивності низькорівневої мережі, СКТВ фокусується на пом'якшенні інверсії пріоритетів в рівні віртуалізації над рідним мережевим стеком. Таким чином, СКТВ є ефективною на верхньому рівні.

3.3 Висновки до розділу 3

Напрям роботи був спрямований на досягнення комунікації в режимі реального часу на рівні платформи та інфраструктури через МППЗ та СКТВ

відповідно. Обидві системи мають обмежену роздільну здатність, зберігаючи масштабованість. Хоча оцінка показує багатообіцяючі показники зв'язку в режимі реального часу, МППЗ і СКТВ можуть бути розширені в наступних аспектах. Розподільник навантаження МППЗ спрацьовує в момент створення нової теми. Однак рішення, прийняті в цей час, можуть стати неадекватними через динаміку навантаження під час наступного виконання. МППЗ реалізовано з обмеженим діапазоном динаміки навантаження, а саме зі змінами навантаження між видавцями в межах певної теми. І навіть це рішення обмежене в регулюванні параметрів множину, а субтокена. Більш повне рішення буде обробляти сценарії, коли зміщення навантаження між ядрами процесора. Видавці потенційно перевантажують одного з брокерів, яким спочатку були призначені теми. Аналогічно, таке рішення, також, повинно обробляти запити на збільшення навантаження на задану тему, тобто перегляд параметрів сегмента токена. Крім того, відхід теми може звільнити ємність, щоб інша тема, яка в даний час розділена між кількома брокерами, могла бути консолідована на меншу кількість брокерів, і в процесі знизити штраф РОШ. Щоб задовольнити ці вимоги, МППЗ потребує дрібнозернистого адаптера, який реагує на всю динаміку навантаження, приймаючи рішення з обмеженими витратами. У ВАТС необхідність дрібнозернистої адаптації до динаміки виникає, коли домен має кілька ядер процесора.

МППЗ і СКТВ досягають комунікування в реальному часі незалежно на двох рівнях. Потенційне розширення і розробка інтерфейсу дає змогу координувати ці дві системи. Наприклад, у хмарних середовищах брокери обміну повідомленнями можуть бути розгорнуті в гостьових доменах. За допомогою цього інтерфейсу брокери обміну повідомленнями МППЗ і можуть віддавати ЦРО на рівні тем і профілі навантаження обміну повідомленнями доменів, таким чином, що СКТВ може розподіляти і диференціювати мережеві потоки (обмін повідомленнями) з більш дрібнозернистою деталізацією. На рівні платформи створено проміжне програмне забезпечення, яке має диференціацію затримок, ізоляцію послуг через обмеження швидкості та масштабованість за рахунок

розподілу навантаження між брокерами повідомлень. Ключовим внеском МППЗ є виявлення (розподіленого) негативного впливу обмеження ставки на латентність (штраф РОШ). Щоб зменшити цей штраф, досліджено три принципи розподілу навантаження та розробляємо новий розподільник навантаження, який гарантує затримку ЦРО тем.

ВИСНОВКИ

З розвитком все більш потужних і гнучких платформ віртуалізації, розподілені м'які застосунки реального часу все частіше розгортаються у віртуалізованих середовищах. Ці засоби створюють нові виклики, коли справа доходить до гарантування низької та передбачуваної латенції. В роботі оцінено затримку мережі за наявності різноманітних шаблонів трафіку та конфігурацій системи, включаючи використання декількох існуючих механізмів керування трафіком Linux. В результаті показано, що деякі компоненти, пов'язані з віртуалізацією, вводять інверсії пріоритетів у передачі та прийомі мережі. Оскільки ці обмеження передбачають спільне використання процесора між мережевими потоками, механізми управління трафіком Linux, призначені для спільного використання, не можуть бути ідеальним рішенням. Отже, в роботі пропонується система контролю трафіку, що враховує віртуалізацію, яка вирішує ці обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. Додатково СКТВ масштабується до багатоядерних процесорів і забезпечує справедливий механізм розподілу навантаження. Завдяки СКТВ контролюється використання процесора. Тоді, можна досягти повної структури, яка забезпечує диференційовану та ізольовану обробку трафіку у всіх компонентах мережі.

Напрям роботи був спрямований на досягнення комунікації в режимі реального часу на рівні платформи та інфраструктури через МППЗ та СКТВ відповідно. Обидві системи мають обмежену роздільну здатність, зберігаючи масштабованість. Хоча оцінка показує багатообіцяючі показники зв'язку в режимі реального часу, МППЗ і СКТВ можуть бути розширені в наступних аспектах.

Розподільник навантаження МППЗ спрацьовує в момент створення нової теми. Однак рішення, прийняті в цей час, можуть стати неадекватними через динаміку навантаження під час наступного виконання. Наразі МППЗ працює з обмеженим діапазоном динаміки навантаження, а саме зі змінами навантаження між видавцями в межах певної теми. І навіть це рішення обмежене в регулюванні

параметрів множину, а субтокена. Більш повне рішення буде обробляти сценарії, коли зміщення навантаження між ядрами процесора. Видавці потенційно перевантажують одного з брокерів, яким спочатку були призначені теми. Аналогічно, таке рішення, також, повинно обробляти запити на збільшення навантаження на задану тему, тобто перегляд параметрів сегмента токена. Крім того, відхід теми може звільнити ємність, щоб інша тема, яка в даний час розділена між кількома брокерами, могла бути консолідована на меншу кількість брокерів, і в процесі знизити штраф РОШ. Щоб задовольнити ці вимоги, МППЗ потребує дрібнозернистого адаптера, який реагує на всю динаміку навантаження, приймаючи рішення з обмеженими витратами. У ВАТС необхідність дрібнозернистої адаптації до динаміки виникає, коли домен має кілька ядер процесора. У поточній версії СКТВ ребалансування призначень запускається тільки в момент створення або завершення роботи гостьового домену. Але під час звичайного часу виконання варіація трафіку гостьових доменів все ще може призвести до несправедливого розподілу навантаження між ядрами процесора домена. Одним з потенційних рішень є покладання на балансер для балансування сповіщень, які надіслані з гостьових доменів, між ядрами процесора під час виконання. У ньому сповіщення розглядаються як апаратні переривання, тому вони мають конфігурації спорідненості, які визначають, які ядра процесора обробляють їх. Можна прихованою програмою динамічно переналаштовувати спорідненість кожного переривання та сповіщення таким чином, щоб обробка переривань/сповіщень була збалансована між ядрами процесора. Щоб завершити таку дрібнозернисту адаптацію у СКТВ, кожного разу, коли спорідненість сповіщення переналаштовується, відповідний пристрій повинен бути відповідно перепризначений. Впровадження та порівняння цього рішення з поточним механізмом ребалансування може бути цінним розширенням СКТВ.

МППЗ і СКТВ досягають комунікування в реальному часі незалежно на двох рівнях. Потенційне розширення і розробка інтерфейсу дає змогу координувати ці дві системи. Наприклад, у хмарних середовищах брокери обміну повідомленнями можуть бути розгорнуті в гостьових доменах. За допомогою

цього інтерфейсу брокери обміну повідомленнями МППЗ і можуть віддавати ЦРО на рівні тем і профілі навантаження обміну повідомленнями доменів, таким чином, що СКТВ може розподіляти і диференціювати мережеві потоки (обмін повідомленнями) з більш дрібнозернистою деталізацією.

З недавньою появою мереж IoT, Industry 4.0 та 5G стає загальною тенденцією, що великомасштабні та чутливі до затримки програми розгортаються в хмарних середовищах. Оскільки PaaS та IaaS задовольняють вимогам масштабованості цих хмарних застосунків, двошарова комунікаційна архітектура одночасно вводить новий виклик вимогам реального часу. Тому, фокурс в роботі спрямовано на вирішення цієї задачі та досягнення комунікування в режимі реального часу на двох різних рівнях. На рівні платформи створено проміжне програмне забезпечення (МППЗ), яке має диференціацію затримок, ізоляцію послуг через обмеження швидкості та масштабованість за рахунок розподілу навантаження між брокерами повідомлень. Ключовим внеском МППЗ є виявлення (розподіленого) негативного впливу обмеження ставки на латентність (штраф РОШ). Щоб зменшити цей штраф, досліджено три принципи розподілу навантаження та розробляємо новий розподільник навантаження, який гарантує затримку ЦРО тем. На рівні інфраструктури розроблено систему управління трафіком, що враховує віртуалізацію у віртуалізованих хостах. Також, визначено обмеження механізмів контролю трафіку дисципліни масового обслуговування Linux, які залишають спільне використання процесора між мережевими потоками незахищеним. Щоб пом'якшити ці обмеження, СКТВ надає нову архітектуру мережевого вводу-виводу, яка забезпечує диференціацію затримки використання процесора за рахунок пріоритетної обробки пакетів. Крім того, СКТВ має ізоляцію послуг за допомогою обмеження швидкості та масштабованість за рахунок справедливого розподілу навантаження між багатоядерними процесорами.

Експериментальні результати показують гарантію ЦРО або значне покращення затримки для м'яких застосунків у режимі реального часу.

Запропоновані рішення забезпечують ефективний підхід до розподілених м'яких застосунків реального часу, які все частіше розгортаються у віртуалізованих середовищах.

У першому розділі розглянуто механізми та способи комунікування в реальному часі на двох рівнях між віртуальними машинами, а також ітеративний розподіл робочого навантаження. В розділі оцінено затримку мережі за наявності різноманітних шаблонів трафіку та конфігурацій системи, включаючи використання декількох існуючих механізмів керування трафіком Linux. В результаті показано, що деякі компоненти, пов'язані з віртуалізацією, вводять інверсії пріоритетів у передачі та прийомі мережі. Оскільки ці обмеження передбачають спільне використання процесора між мережевими потоками, механізми управління трафіком Linux, призначені для спільного використання, не можуть бути ідеальним рішенням.

У другому розділі здійснено розробку системи контролю трафіку, що враховує віртуалізацію, яка вирішує ці обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. Додатково СКТВ масштабується до багатоядерних процесорів і забезпечує справедливий механізм розподілу навантаження. Завдяки СКТВ контролюється використання процесора.

У третьому розділі запропоновано МППЗ, яке реалізовано з обмеженим діапазоном динаміки навантаження, а саме зі змінами навантаження між видавцями в межах певної теми. Більш повне рішення буде обробляти сценарії, коли зміщення навантаження між ядрами процесора. Видавці потенційно перевантажують одного з брокерів, яким спочатку були призначені теми. Аналогічно, таке рішення, також, повинно обробляти запити на збільшення навантаження на задану тему, тобто перегляд параметрів сегмента токена. Крім того, відхід теми може звільнити ємність, щоб інша тема, яка в даний час розділена між кількома брокерами, могла бути консолідована на меншу кількість брокерів, і в процесі знизити штраф РОШ. Щоб задовольнити ці вимоги, МППЗ потребує дрібнозернистого адаптера, який реагує на всю динаміку навантаження,

приймаючи рішення з обмеженими витратами. У ВАТС необхідність дрібнозернистої адаптації до динаміки виникає, коли домен має кілька ядер процесора.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Redouane Meddane. Dial Plan and Call Routing Demystified On Cisco Collaboration Technologies: Cisco Unified Communication Manager. 2022. 867 p.
2. National Security Agency. Deploying Secure Unified Communications/Voice and Video over IP Systems. 2021. 72 p.
3. What is Unified Communications (UC)?. URL: https://www.techtarget.com/searchunifiedcommunications/definition/unified-communications (дата звернення: 18.03.2023).
4. Milos Ockay. VoIP: Cisco Unified Communications Manager Express: A Hands-On Approach. 2018. 190 p.
5. Brett Hall, Nik Smith. Practical Cisco Unified Communications Security. 2020. 528 p.
6. What is Unified Communications? Definition, System, Cloud Service, Best Practices and Examples. URL: https://www.spiceworks.com/collaboration/unified-communications/articles/what-is-unified-communications/ (дата звернення: 18.03.2023).
7. Ronald Schlager. Unified Communications BG: Deutsche Ausgabe. 2018. 200 p.
8. Gerardo Barajas Puente. Elastix Unified Communications Server Cookbook. 2015. 553 p.
9. UC Today. URL: https://www.uctoday.com/unified-communications/what-is-unified-communications/ (дата звернення: 20.03.2023).
10. Definition of Unified Communications. URL: https://www.gartner.com/en/information-technology/glossary/unified-communications-uc (дата звернення: 20.03.2023).
11. Jaromír Vrbka. Using Artificial Neural Networks for Timeseries Smoothing and Forecasting: Case Studies in Economics. 2021. 199 p.

12. Convolutional Neural Network: Benefits, Types, and Applications. URL:https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/ (дата звернення: 03.04.2023).

13. James Loy. Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects. 2019. 580 p.

14. Unified Communications and Collaboration. URL: https://www.cisco.com/c/en/us/products/unified-communications/index.html (дата звернення: 05.04.2023).

15. Igor Livshin. Artificial Neural Networks with Java: Tools for Building Neural Network Applications. 2021. 729 p.

16. Phil Kim. MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence. 2017. 168 p.

17. Paulo J.G. Lisboa, Emmanuel C. Ifeachor, Piotr S. Szczepaniak. Artificial Neural Networks in Biomedicine (Perspectives in Neural Computing). 2017. 302 p.

18. Introduction to Convolution Neural Network. URL: https://www.geeksforgeeks.org/introduction-convolution-neural-network/ (дата звернення: 05.04.2023).

19. CNN for Deep Learning. URL: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/(дата звернення: 06.04.2023).

20. What are Convolutional Neural Networks?. URL: https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network (дата звернення: 07.04.2023).

21. Michael Taylor, Mark Koning. The Math of Neural Networks. 2017. 187 p.

22. Stuart Russell, Peter Norvig. Artificial Intelligence: A Modern Approach (Pearson Series in Artifical Intelligence). 2020. 1136 p.

23. Steven D'Ascoli. Artificial Intelligence and Deep Learning with Python: Every Line of Code Explained for Readers New to AI and New to Python. 2022. 285 p.

24. Herbert Jones. Neural Networks: An Essential Beginners Guide to Artificial Neural Networks and their Role in Machine Learning and Artificial Intelligence. 2018. 95 p.

25. Artificial Intelligence - The Future of AI - AI and You. URL: https://egfound.org/projects/digital-revolution-technology-power-you/?gclid=Cj0KCQjwj_ajBhCqARIsAA37s0zFyPiyWDGRbEvBzfLxBUrlQ_TC-JJ1lUPq31uWiR6_wO1zLhmXcqwaAtbkEALw_wcB (дата звернення: 23.03.2023).

26. What Is Artificial Intelligence - Is AI an Existential Risk. URL: https://www.givingwhatwecan.org/cause-areas/long-term-future/artificial-intelligence?gad=1&gclid=Cj0KCQjwj_ajBhCqARIsAA37s0xlMhaLISB47gz7ZM1bDEfEEWobMsoPYZt_owJRa99RYysyaZrsGKoaAhxtEALw_wcB (дата звернення: 24.03.2023).

27. Artificial intelligence (AI), data and criminal justice. URL: https://www.fairtrials.org/campaigns/ai-algorithms-data/?gad=1&gclid=Cj0KCQjwj_ajBhCqARIsAA37s0wfeecXa5kKsZ9dzKlogm--ufMksiiDmdts5uv_sIiRdoWRtUWGC5AaAn88EALw_wcB (дата звернення: 24.03.2023).

28. Artificial Neural Network Tutorial. URL: https://www.javatpoint.com/artificial-neural-network (дата звернення: 29.03.2023)

29. Ivan Nunes da Silva. Artificial Neural Networks: A Practical Course. 2017. 327 p.

30. Mohamad H. Hassoun. Fundamentals of Artificial Neural Networks. 2017. 511 p.

31. A Comprehensive Guide to Convolutional Neural Networks. URL: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/ (дата звернення: 01.04.2023).

32. John Paul Mueller, Luca Massaron. Artificial Intelligence for Dummies (For Dummies (Computer/Tech)). 2021. 368 p.

33. Tom Taulli. Artificial Intelligence Basics: A Non-Technical Introduction. 2019. 199 p.

34. Dan Fitzpatrick, Amanda Fox. The AI Classroom: The Ultimate Guide to Artificial Intelligence in Education (The Hitchhiker's Guide for Educators Series). 2023. 389 p.

35. Michael Gordon Cohen. Artificial Intelligance and generative AI for beginners: An easy guide to learning about the world of AI and AI Generatives such as ChatGPT, Dall-E, Jasper, Midjourney and much more. 2023. 130 p.

36. What are Recurrent Neural Networks?. URL: https://www.ibm.com/topics/recurrent-neural-networks (дата звернення: 09.04.2023).

37. Introduction to Recurrent Neural Network. URL: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ (дата звернення: 09.04.2023).

38. Recurrent Neural Networks Cheatsheet. URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (дата звернення: 12.04.2023).

39. Suman Kumar Swarnkar. Optimized Convolution Neural Network (OCNN) for VBSLR: Optimized Convolution Neural Network (OCNN) for Voice-Based Sign Language Recognition: Optimization & Regularization. 2021. 120 p.

40. Amit Kumar Tyagi, Ajith Abraham. Recurrent Neural Networks. 2022. 412 p.

41. Mohan Kumar Silaparasetty. Beginning with Deep Learning Using TensorFlow: A Beginners Guide to TensorFlow and Keras for Practicing Deep Learning Principles and Applications. 2022. 372 p.

42. Kishita Y., Mizuno Y., Fukushige S., Umeda Y. Scenario structuring methodology for computer-aided scenario design: An application to envisioning sustainable futures, Technol. Forecast. Soc. Chang. 2020. 160 p.

43. Teslyuk V., Kazarian A., Kryvinska N., Tsmots I., Teslyuk T. Automated synthesis method of smart home systems based on the architectural pattern redux. *CEUR Workshop Proceedings*. 2019.Vol. 2533. P. 58-69.

44. Yadin A. Computer Systems Architecture, Chapman and Hall, CRC, 2016. 467 p.

45. Nisan N., Schocken S. The Elements of Computing Systems, second edition: Building a Modern Computer from First Principles 2nd Edition, The MIT Press, 2021. 344 p.

46. Balta-Ozkan N., Davidson R., Bicket M., Whitmarsh L., *Social barriers to the adoption of smart homes, Energy Policy*.2013. Vol. 63. P. 363-374.

47. Barrett S.F. Microchip AVR® Microcontroller Primer: Programming and Interfacing, Morgan & Claypool Publishers, 2019. 374 p.

48. Kravets A.G., Bolshakov A.A., M.V. Shcherbakov Cyber-Physical Systems: Industry 4.0 Challenges (Studies in Systems, Decision and Control, 260), Springer; 1st ed., 2020. 349 p.

49. Papazoglou P. M. An Educational Guide to the AVR Microcontroller Programming: AVR Programming : Demystified (Assembly Language) (Volume 1), CreateSpace Independent Publishing Platform, 2018. 274 p.

50. Monk S. Programming Arduino Next Steps: Going Further with Sketches. McGraw-Hill Education TAB, 2018. 320 p.

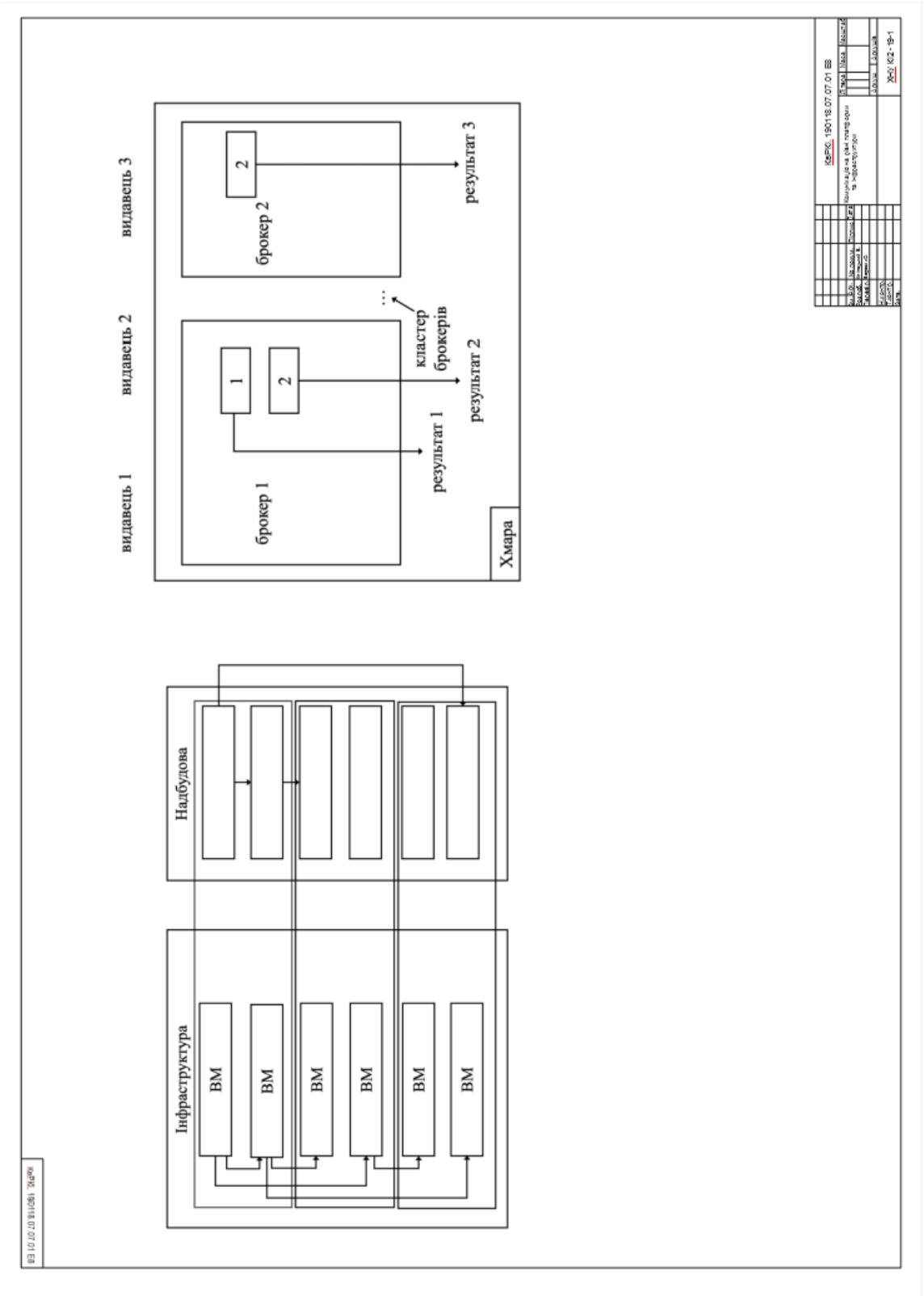
51. Barrett S.F. Microchip AVR® Microcontroller Primer: Programming and Interfacing, Morgan & Claypool Publishers, 2019. 374 p.

52. Hu Y., Tilke D., Adams T. et al. Smart home in a box: usability study for a large scale self-installation of smart home technologies. *J Reliable Intell Environ*. 2016. Vol. 2. P. 93-106.

Додаток А

(обов’язковий)

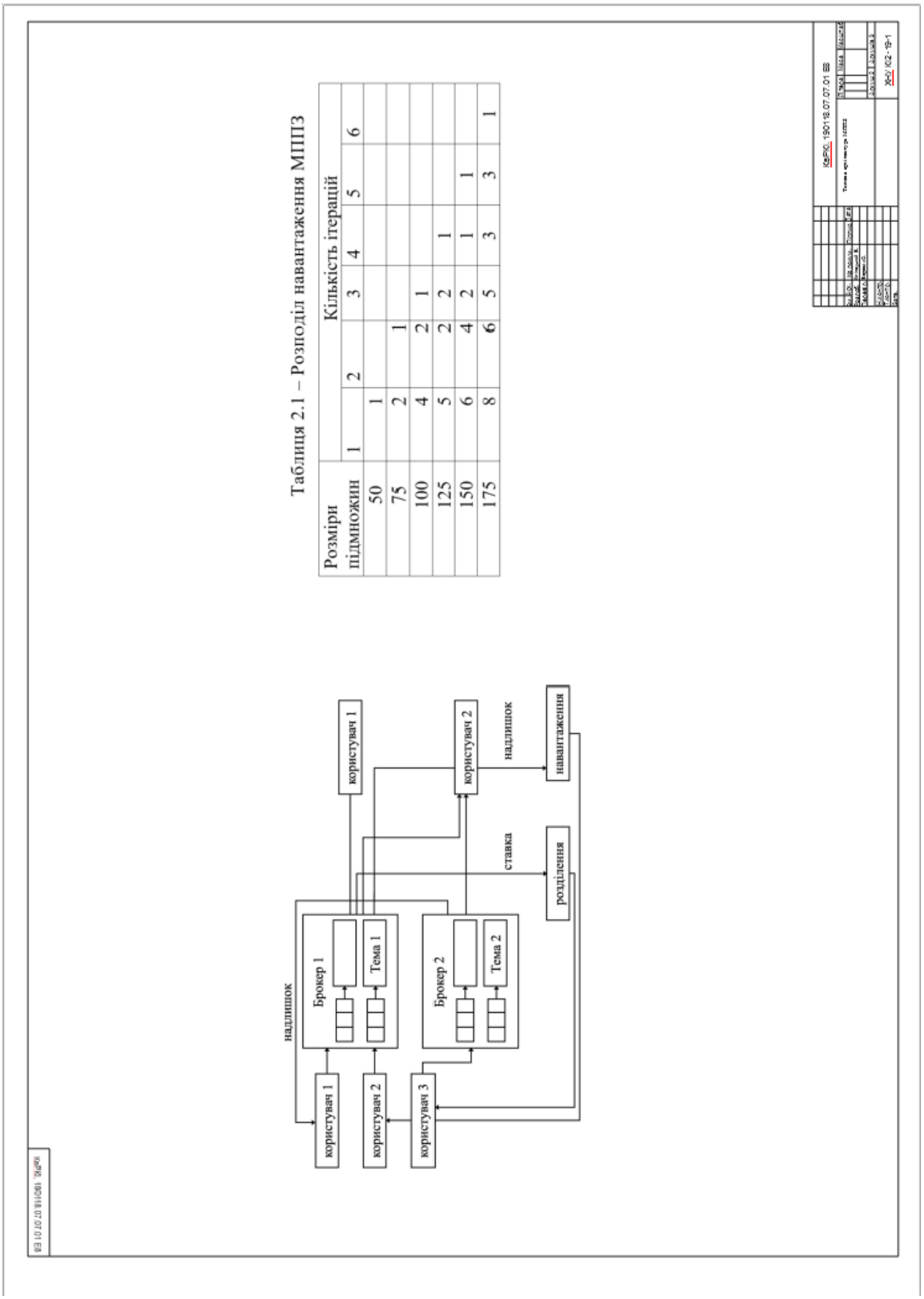
Копія креслення «Комунікація на рівні платформи та інфраструктури»



Додаток Б

(обов’язковий)

Копія креслення «Типова архітектура МППЗ»



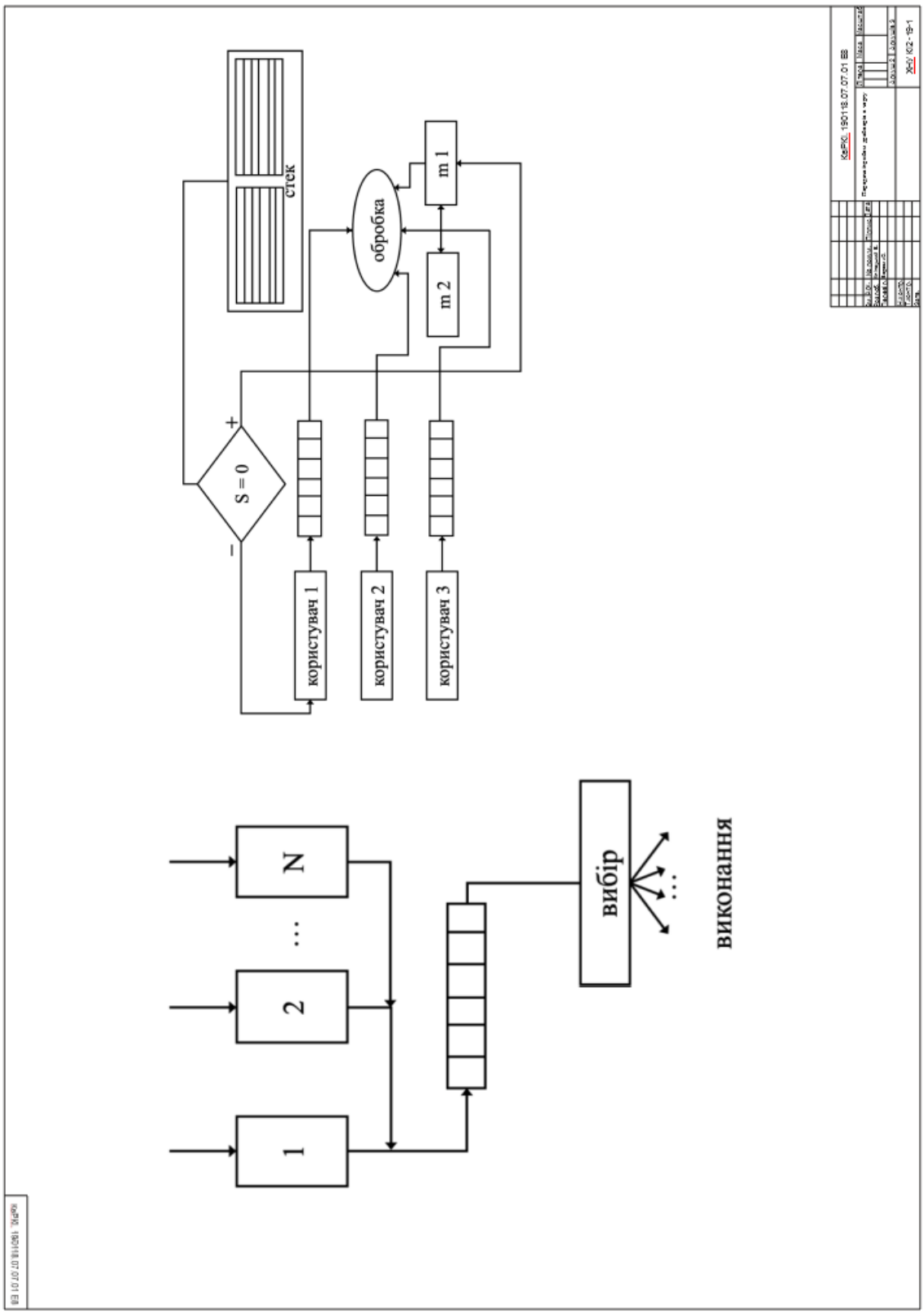
Таблиця 2.1 – Розподіл навантаження МППЗ

Розміри підмножин	Кількість ітерацій						
	1	2	3	4	5	6	
50		1					
75		2	1				
100		4	2	1			
125		5	2	2	1		
150		6	4	2	1	1	
175		8	6	5	3	3	1

Додаток В

(обов’язковий)

Копія креслення «Передача/прийом драйверів в чергу»



Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Кліпацький Владислав Андрійович

ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

31 травня 2023 року

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1015649184

Дата перевірки:
19.06.2023 19:49:29 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
19.06.2023 19:51:00 EEST

ID користувача:
100005591

Назва документа: **Кліпацький_Проміжне програмне забезпечення для комунікації між віртуальними машина...**

Кількість сторінок: **72** Кількість слів: **15857** Кількість символів: **123519** Розмір файлу: **822.00 KB** ID файлу: **1015295007**

0.85%
Схожість

Найбільша схожість: **0.52%** з джерелом з Бібліотеки (ID файлу: **1015041352**)

0.67% Джерела з Інтернету	35	Сторінка 74
0.69% Джерела з Бібліотеки	101	Сторінка 74

0% Цитат

Цитати	5	Сторінка 75
Посилання	1	Сторінка 75

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 4.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 11%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 117102 Название: БКР Проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ Добавлено в БД: 2023-06-19 Авторы: В. А. Кліпацький Руководители: О. В. Боровик Консультанты: Оппоненты:	Символы	Лексемы	Символы	Лексемы
	118186	864	5802 (5%)	62 (7%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ

Автор: *Кліпацький Владислав Андрійович*

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Боровик Олег Васильович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-30 джерелами на один фрагмент речення;

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 0,85% і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Олег БОРОВИК

Гарант ОП _____ Сергій ЛИСЕНКО

Завідувач кафедри КІІС _____ Тетяна ГОВОРУЩЕНКО

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Владислав КЛІПАЦЬКИЙ

Тема: Проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 68

1. Короткий зміст роботи та прийнятих рішень У роботі розроблено проміжне програмне забезпечення для комунікації між віртуальними машинами хмарних середовищ

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі розглянуто механізми та способи комунікування в реальному часі на двох рівнях між віртуальними машинами, а також ітеративний розподіл робочого навантаження. У другому розділі здійснено розробку системи контролю трафіку, що враховує віртуалізацію, яка вирішує ці обмеження за допомогою нової архітектури мережевого вводу-виводу з пріоритетними та обмеженими потоками ядра. У третьому розділі запропоновано МППЗ, яке реалізовано з обмеженим діапазоном динаміки навантаження, а саме зі змінами навантаження між видавцями в межах певної теми. У Висновках підведено підсумки виконаної роботи.

4. Позитивні сторони роботи: немає.

5. Негативні сторони роботи: функціонування розробленого проміжного програмного забезпечення не підтверджено експериментально.

6. Оцінка графічного оформлення та пояснювальної записки роботи: __-____

7. Відгук про роботу в цілому: __Робота виконана на задовільному рівні.

8. Інші зауваження: ___-

9. Оцінка дипломної роботи:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи вважаю, що робота заслуговує оцінки «задовільно» 3,25 (D)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) __Мартинюк Валерій Володимирович, д.т.н., професор, завідувач кафедри АКІТР ХНУ

"_20_" __червня__2023р.