Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель опису розподілених систем з використанням мереж Петрі
(тема)

Виконав:

студент II курсу, групи СПм-20-2

Масленіков Д.Є.
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Лебедєв О.Г.
(посада, прізвище, ініціали)

Допускається до захисту

В.о. зав. кафедри ЕОМ _____ Волк М.О.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Масленікову Денису Євгеновичу
(прізвище, ім’я, по батькові)

1. Тема роботи Модель опису розподілених систем з використанням мереж Петрі

затверджена наказом по університету від “ 28 ” березня 2022 р. № 413 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи
розподілені системи
мережі Петрі
сервісні системи
C#

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз предметної області
Модель опису розподіленої сервісної системи
Практична реалізація модифікованої мережі Петрі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 18 слайдів

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	28.03.2022–10.04.2022	
2	Дослідження апарату мереж Петрі	11.04.2022–16.04.2022	
3	Розробка моделі кольорової мережі Петрі	17.04.2022–26.04.2022	
4	Моделювання та перевірка роботи моделі	27.04.2022–02.05.2022	
5	Отримання та аналіз результатів	03.05.2022–06.05.2022	
6	Оформлення пояснювальної записки	07.05.2022–13.05.2022	

Дата видачі завдання 28 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Лебедєв О.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 58 с., 22 рис., 1 дод., 6 джерел.

АРХІТЕКТУРА, СЕРВІСНА СИСТЕМА, МЕРЕЖА ПЕТРІ, ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Метою кваліфікаційної роботи є побудова моделі, яка дозволяє розробникам програмного забезпечення описувати та аналізувати розподілені сервісні системи.

У ході виконання кваліфікаційної роботи розглянуті сервісні системи які орієнтовані на безперервну роботу протягом тривалого часу, піддаються постійним змінам. Модифікація відбувається як на основі впровадження нової функціональності, так і з-за постійного виявлення виникаючих складностей. Причиною багатьох проблем є неповна інформація про систему - відсутність можливості повноцінної деталізації частини проекту на ранньому проектуванні. Ці проблеми яскраво виявляються в процесі розробки розподілених сервісних систем, де особлива важлива можливість побудови гнучкої архітектури, що змінюється в ході рішень багатофакторних завдань розробки програмних продуктів.

Узгодження модифікацій у ході життєвого циклу сервісного рішення безпосередньо пов'язане з побудовою моделі системи та можливістю імітаційного моделювання найважливіших процесів, що відбуваються в ході її роботи. Це необхідно для аналізу проблемних місць архітектури розглянутого рішення та передбачення поведінки програмного продукту, що розробляється.

ABSTRACT

Master’s thesis: 58 pages, 22 figures, 1 appendices, 6 sources.

ARCHITECTURE, SERVICE SYSTEM, PETRI NETWORK, SIMULATION MODELING, SOFTWARE.

The major goal of this thesis is a model of construction, which allows software developers to describe and analyze distributed service systems.

In order to considered service systems which are focused on continuous work for a long time, are subject to constant changes. Modification occurs both on the basis of the introduction of new functionality and due to the constant identification of emerging difficulties. The reason for many problems is incomplete information about the system - the lack of full detail of the project in the early design. These problems are clearly manifested in the development of distributed service systems, where the ability to build a flexible architecture is especially important, which changes during the solution of multifactor problems of software development. Coordination of modifications during the life cycle of the service solution is directly related to the construction of the system model and the possibility of simulation of the most important processes occurring during its operation. This is necessary to analyze the problem areas of the architecture of the solution and predict the behavior of the software product being developed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

P2P – однорангова мережа (англ., Peer-2-Peer)

SoaML – сервісно-орієнтована архітектура мова моделювання (англ., Service-oriented architecture Modeling Language)

КМП – кольорова мережа Петрі

ММП – модифікація мережі Петрі

ПЗ – програмне забезпечення

ПВВМ – програмована використовувана вентиляційна матриць

ЦП – центральний процесор

ВСТУП

Системи, орієнтовані на безперервну роботу тривалий час, зазнають постійних змін. Модифікація відбувається як на основі впровадження нової функціональності, так і через перманентне виявлення складностей, що виникають. Причиною багатьох проблем є неповнота інформації про систему відсутність можливості повноцінної деталізації частин проекту на ранній стадії проектування. Ці проблеми яскраво виявляються в процесі розробки розподілених сервісних систем, де особливо важлива можливість побудови гнучкої архітектури, що змінюється, в ході вирішення багатофакторного завдання розробки програмних продуктів.

У роботі розглянуті методи моделювання розподілених сервісних систем з погляду побудови архітектури; аналізу протоколів взаємодії компонентів систем; споживаних ними ресурсів; чисельного моделювання розглянутої розподіленої системи та інтеграції одержуваних у ході роботи системи даних у розглянуту модель.

Узгодження модифікацій у ході життєвого циклу сервісного рішення безпосередньо пов'язане з побудовою моделі системи та можливістю імітаційного моделювання найважливіших процесів, що відбуваються в ході її роботи. Це необхідно для аналізу проблемних місць архітектури розглянутого рішення та передбачення поведінки програмного продукту, що розробляється. В рамках даної роботи розглянуто деякі питання моделювання сервісних систем для аналізу обчислень, що виробляються на вузлах, що становлять єдину мережу, об'єднану комунікаційним брокером або Peer-2-Peer (P2P) системою, яка передає інформацію безпосередньо між вузлами. Також розглядається робота сервісних систем, побудованих з урахуванням суперкомп'ютерних кластерів. Досліджуються процеси, що виконуються безпосередньо на локальних вузлах та у форматі взаємодії віртуальних хмарних ресурсів.

Метою кваліфікаційної роботи є побудова моделі, яка дозволяє розробникам програмного забезпечення описувати та аналізувати розподілені сервісні системи.

Для досягнення поставленої мети необхідно вирішення низки основних завдань:

- побудувати модель, що дозволяє аналізувати взаємодії сервісів у розподіленому обчислювальному середовищі;

- описати методику оцінки архітектури програмних комплексів, що дозволяє враховувати конкуренцію за ресурси та логіку управління в умовах неповної інформації;

- створити практичну реалізацію для аналізу сервісних систем на основі представленої моделі у вигляді запропонованої методики.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Існуючі методики моделювання дозволяють проводити системний аналіз програмного забезпечення (ПЗ), що розробляється, верифікувати моделі, що змінюються в часі, на предмет небажаної поведінки, і створювати на базі отриманих моделей програмний продукт. Проте, під час вирішення завдання моделювання сервісної системи найчастіше виставляються жорсткі рамки на повноту опису моделі (у разі формальної верифікації алгоритмів роботи програми); послідовності дій та їх математичних характеристик (у разі роботи з математичними моделями такими як мережі Петрі та кінцеві автомати); можливості моделювання ходу системних взаємодій (у разі використання графічних нотацій, таких як мови Unified Modeling Language (UML) та Service-oriented architecture Modeling Language (SoaML)).

Слід також зазначити, що деякі формальні методи моделювання та верифікації задіяні на спеціальних етапах циклу розробки та не передбачають розширення та доповнення моделі під час розробки ПЗ. Це ускладнює можливість поступового накопичення знань та ітеративного розширення картини поведінки системи.

Розвиток комп'ютерних технологій призводить до появи безлічі різних підходів до вирішення задач та аналізу отриманих результатів. Архітектура застосунків розшаровується, з'являються різні стандарти та посібники з методів побудови та використання систем. Часто такі інструкції вимагають додаткових коментарів і розглядають вузький пласт рішення, що розробляється, не звертаючи уваги на велику картину того, що відбувається в системі.

Текстові рекомендації в більшості випадків не можуть повністю описати події, що спостерігаються, що відбуваються в функціонуючій системі. Засоби налагодження, що надаються окремими компонентами системи, дозволяють аналізувати мікрорівень, на якому працює кожен із них

і можуть приховати системну причину виникнення тих чи інших проблем.

Важливим аспектом, що впливає на розробку сервісних систем, є обладнання, на якому вони виконуються. Розвиток апаратних компонентів і боротьба за вихід до обчислень зі швидкістю екзафлопс для реальних наукових додатків, а не тільки для синтетичних тестів, призводить до значної зміни підходу до побудови високонавантажених систем.

Створювані рішення надають дедалі більше гнучкості програмним комплексам, дозволяючи виробляти динамічне управління різними компонентами системи та під час виконання змінювати структуру роботи з апаратним комплексом потреб додатків. З появою програмного управління пристроєм мережі стало можливо адаптувати роботу сервісів, що взаємодіють, в міру зміни активності і типів взаємодії, здійснюваних користувачами. Що, разом із поширенням співпроцесорів, здатних здійснювати попередню обробку що входить у кожен вузол системи трафіку, призводить до підвищення кількості умов, які впливають перебіг роботи програмного рішення.

Побудова моделі роботи такої підсистеми неможлива без контексту всієї системи, що вимагає комплексного моделювання створюваної системи. Не менш важливим для розробки сервісних систем є накопичення бібліотек програмних кодів. Так, поява стабільних версій різних програмних компонентів, на основі яких можливе вирішення конкретних прикладних завдань, надає можливість використовувати компонентне програмування розробки сервісних систем.

У свою чергу, різноманітність конфігурацій апаратних комплексів та показників роботи на них конкретних компонентів дозволяє вибирати програмне забезпечення, що найбільш успішно вписується у створюване рішення як з точки зору продуктивності, так і ціни комплектації необхідного обладнання. Це актуалізує завдання моделювання роботи програмних комплексів систем задовго до початку розробки. Поряд із сказаним, нові підходи до організації інфраструктури обчислень розширюють пул

застосовуваних підходів до розробки програмного забезпечення. Сучасні хмарні системи дозволяють розробникам програм переносити свої послуги в площину сервісів.

Користувач втрачає можливість прямого доступу до програмної логіки створюваних продуктів, що підвищує безпеку, проте вимагає створення динамічно масштабованої захищеної системи, здатної ефективно відповідати на надіслані до неї запити. Паралельне виконання завдань у додатках такого роду вимагає балансування навантаження, що розповсюджується в системі, що неможливо без формування уявлення про процеси, що відбуваються в ній. Сервісні програми дозволяють розробникам виробляти швидкі ітерації, що розширюють функціональність рішення, що створюється. При цьому розбиття сервісної системи на набір мікросервісів, що взаємодіють, наводить до появи проблем у двох площинах:

- аппаратна архітектура сучасних комп'ютерів не дозволяє здійснювати збалансований конкурентний доступ до ресурсів, викликаючи «голодання» як на рівні високошвидкісної пам'яті ядерного центрального процесора, так і з точки зору роботи з сопроцесорами. Протокол, що описує взаємодію окремих сервісів між собою, у разі відсутності розрахунків завантаження мережі може привести до переповнення каналів зв'язку та розбалансування системи;

- логічна взаємодія ізольованих мікросервісів, що здійснюється за допомогою засобів передачі повідомлень, може приводити до логічних тупикам і взаємозалежним блокуванням, а це вимагає аналізу системи логічного взаємодії компонентів.

Багато проблем, які вирішуються в рамках хмарних сервісних систем, вперше вирішувалися в суперкомп’ютерних центрах. Сьогодні в них спостерігається перехід від моделі вичисленого, логіки яких сфокусовано виключно на центральному процесорі (ЦП) та його властивостей, до розподілених вичислень, діючих властивостей процесорів, що містять спеціалізовані набори інструкцій і требуючих складної координації при

роботі з додатком. Так звані графічні процесори (GPU, GPU) здатні видавати значний приріст у продуктивності при роботі з операціями з плаваючою точкою відповідно до ЦП, а сопроцесори на основі програмованих використовуваних вентиляційних матриць (ПВВМ, FPGA) дозволяють збільшувати швидкість виконання спеціальних інструкцій і програми на мові OpenCL.

Суперкомп'ютер Tianhe-2 є одним із лідерів списку «TOP500» самих виробничих комп'ютерів у світі. Він складається з 16 тисяч узлов, що включають у себе по два ЦПУ з двома ядрами кожного та по трьох процесорів Xeon Phi 31S1P (по 57 ядер на ускорювач, частота 1,1 ГГц). Основною задачею цього суперкомп'ютера є рішення наукових задач. Для запуску задач у нього (як і в багатьох суперкомп’ютерах) використовується система SLURM, яка працює за принципом оптимізації часу, затраченого конкретним додатком на кластері. Це означає, що користувачі всякий раз створюють окремі повноцінні програми, поєднують логіку по роботі з вхідними даними, їх обробку та візуалізацію результатів.

Зазначимо, що SLURM не надає опцій для контейнерізації запущених додатків, що може збільшити продуктивність роботи додатків для виключення із системи всіх компонентів, крім необхідних кінцевих додатків. Розробка груп вчених кожної програми для суперкомп’ютера, як правило, виробляється «з нуля», або використовується готовий продукт, такий як ANSYS, що дозволяє виконати частину своїх задач на кластері.

Розробка додатків з нуля трудозатратна і віднімає багато часу, тоді як закупка спеціальної ліцензії для роботи програми на кластере збільшує навантаження на бюджет наукового дослідження. Виповнення додатків в контейнеризованому середовищі надає широкі можливості масштабування Приклади, будь то сервіси, постійно підтримують некоторое стан системи, або ж наоборот, розово виконують поставлену задачу, всякий розрізняючи певний обсяг ресурсів можуть працювати як у рамках хмарної системи, так і суперкомп'ютерних центров обработки данных.

Розмежовані системи, що надають постійну виділену обчислювальну потужність, поступово витесняються віртуальними хмарними системами, дозволяючи масштабувати використовувані потужності динамічної.

Облачні системи сьогодні переважно використовуються для рішень комерційних прикладних задач, однак ресурси таких систем все частіше находять і наукове застосування. Продумані хмарні рішення дозволяють вирішувати дві важливі задачі: розширювати та контролювати ресурсну базу, застосовувану програму по мірі його роботи, надають широкий набір спеціалізованих, стандартизованих вичислювальних узлов, що мають силові та слабкі сторони: узли з сопроцесорами, та спеціалізовані системи для зберігання даних. Це дозволяє створювати гнучкі сервісні системи, що містяться з різних взаємодіючих між собою додатків, розподілених за різними вичислительными ресурсами, які не прекращають роботу, що надходить у постійну готовність обробляти нові дані. При цьому динамічні виділення гетерогенні ресурси, в рамках яких відбувається робота сервісних систем, вимагають розробки спеціалізованих засобів контролю та взаємодії.

Управління такими сервісними системами встановлюється на кластерні менеджери, вирішуючи три основні задачі: підтримка працездатності компонентів сервісної системи та її максимальну завантаження, агрегація аналітичної інформації про стан додатків і завантажувальних узлов, а також розширення та згортання доступних систем ресурсів за мірою необхідності. Ярким прикладом такого менеджера є система з відкритим вихідним кодом Apache Mesos.

Програми, що виконуються в рамках кластерних менеджерів таких як, наприклад, Mesos запускаються контейнеризовано, а аналітика із завантаження вузлів, що збирається кластерними менеджерами, дозволяє запускати не ресурсомісткі програми на завантажених вузлах, масштабуючи ресурси, що виділяються динамічно.

Зараз в рамках такої системи можна запускати як традиційні програми, розподілені за допомогою MPI, так і програми обробки Великих Даних, такі

як Apache Spark. Їхня взаємодія один з одним здійснюється через відкриті мережеві інтерфейси. Однак постачальники хмарних рішень дозволяють не тільки масштабувати рішення в рамках одного центру обробки даних, але й географічно розподілити рішення задач і, при необхідності, продублювати збережену інформацію, убезпечивши створюване рішення від непередбачених збоїв різного рівня.

Географічний розподіл та різнорідність ресурсів істотно ускладнюють розробку створюваних рішень, надаючи гнучкість та безпеку. Для прямого моделювання таких програм розподілених по мережі існує ряд комплексів моделювання. Більшість таких систем фокусується на безпосередньому виконанні сервісів на базі віртуальних машин та контейнерів з моделюванням мережевих подій між ними на різному рівні деталізації. Проте, тоді як такі продукти фокусуються на моделюванні мережевих проблем взаємодії сервісів, вони дозволяють розглядати деталі алгоритмів роботи самих додатків. Таке радикальне ускладнення процесу розробки програмного забезпечення поєднується з розшаруванням обчислювальних ресурсів, консервацією граничних швидкостей обчислення одного потоку інструкцій та лавиноподібним зростанням кількості інформації, яку необхідно обробляти на вирішення актуальних завдань.

Перехід до динамічно масштабованих ресурсів де-факто означає переведення більшості рішень у формат сервісів, що виконуються в рамках хмарних обчислювальних потужностей, стійких до зростання навантаження. Це призводить до висновку про те, що незабаром суперкомп'ютерні рішення також почнуть експлуатувати підхід розділених на модулі сервісних систем для вирішення актуальних завдань.

Основними відмінними рисами сервісного рішення є здатність взаємодіяти з іншими додатками за заздалегідь узгодженими правилами – протоколом та орієнтацією на тривалу експлуатацію без зупинки додатка.

Сьогодні сервісні системи приймають безліч різних форм, тому відразу обмовимося, що в рамках даної роботи ми не будемо окремо виділяти сервіси

світу інтернету речей і не розглядатимемо обчислення, що виконуються на скупченнях мобільних пристроїв, а також моделювання поведінки специфічних апаратних функцій і передбачення їхньої поведінки.

Предметом уваги в даній роботі є мережі пристроїв, пов'язаних між собою в рамках розподіленої обчислювальної архітектури, в якій раптове від'єднання пристрою розглядається як виняткова ситуація, а не як норма, природна для безпровідних ad-hoc мереж.

Таким чином, для ефективної декомпозиції розподілених сервісних систем необхідно детально розглянути:

- поширені методи взаємодії у розподілених сервісних системах;
- особливості та проблеми існуючих підходів формального моделювання розподілених сервісних взаємодій.

В результаті такого аналізу стане можливим побудова нових методів декомпозиції розподілених сервісних систем, у тому числі з використанням мереж Петрі.

2 МОДЕЛЬ ОПИСУ РОЗПОДІЛЕНОЇ СЕРВІСНОЇ СИСТЕМИ

Мережі Петрі надають широкі можливості та безліч видів спеціалізацій для аналізу різних систем. Однак, розробка та аналіз розробок, що містять різнорідні процеси, що відбуваються незалежно, призводять до помітного ускладнення читання та аналізу одержуваної моделі. Для вирішення цієї проблеми було запропоновано кілька незалежних підходів, що надають можливість розмальовки міток та угруповання вузлів у межах мереж Петрі [2–5]. Зазначимо, що для коректного описи розподіленої сервісної системи потрібно врахувати особливості предметної області.

Важливо мати можливість аналізувати методи вирішення проблем з використанням сервісів у розподіленому обчислювальному середовищі, та необхідно створити апарат моделювання, який дозволить комплексно описувати складні процеси та робити якісні висновки про поведінку аналізованих систем [6].

2.1 Можливості кольорових мереж Петрі

Існує багато робіт, присвячених тому, як мережі Петрі можуть бути використані для опису технологічних процесів. Концепція кольорів часто вдається до спрощення таких описів.

Колір – це особливий вид мітки, який можна однозначно співвіднести з конкретним значенням раніше відомого набору. Моделі CSPN, GCSPN зберігають можливість перемикання з моделі, що використовує кольори на вихідну. Важливо підкреслити, що формалізм мереж Петрі передбачає видалення міток при переході по одній за раз.

Миттєвий перехід спрацьовує у той момент, коли він стає активним, тоді як синхронізований перехід спрацьовує через позитивний час.

2.2 Проблеми традиційних кольорових мереж Петрі

При роботі з традиційними кольоровими мережами Петрі дослідники стикаються із двома основними проблемами опису моделей сервісних систем:

– створення переходу в залежності від загальної кількості міток неможливе без створення спеціальних схематичних рішень. Таким чином, цілі числа не можуть бути використані із коробки;

– якщо перехід визначено як синхронізований та запускається тільки через певний час δ, його складно поєднати з іншими переходами, щоб отримати заздалегідь визначені часові характеристики. Таким чином, угруповання та комбінація елементів потребують додаткової підтримки з точки зору додаткових маркувань та місць.

2.3 Модифікована кольорова мережа Петрі

Нам потрібно створити набір розширень, який дозволив би користувачам МП створювати:

- цілі числа;
- складові типи;
- комбінації операцій з фіксованими тимчасовими характеристиками;
- композиційні групи для визначення та повторного використання шаблон.

Таким чином, ми отримаємо середовище моделювання, яке надає користувачам наступний функціонал:

- створення своїх комбінацій типів;
- створення переходів, які працюють із такими типами;
- об'єднання місць та переходів у групи та шаблони для багаторазового використання.

Давайте переформулюємо наш підхід до можливості зміни стану $p(s', s,$

T *). Такий підхід узагальнює спосіб зміни станів на підмножині T та стану s загалом. Це ускладнює формулювання мережі з погляду переходів та місць. Хотілося б визначити набір, що описує функції ймовірності спрацьовування переходу Y = $\{\gamma 1, \gamma 2, \ldots \gamma k\}$ для кожного переходу в $T = \{t1, t2, \ldots tk\}$.

Таким чином, γ сама по собі вільна від стану конфігурації системи s і оцінюється коли перехід дозволено через $I(t)$ та $L(t)$. Це обмеження знижує потужність моделювання ММП, але забезпечує чіткість визначення мережі з практичної точки зору моделювання системи.

Сервісні системи складаються з непрямих шарів. Процеси в таких шарах можуть мати різні часові масштаби під час виконання аналогічних операцій. Щоб вирішити таку проблему, ми робимо всі пов'язані з часом транзакції залежними від конфігурації розмітки мережі s замість абстрактного системного часу.

Ми робимо це, додаючи підсистему переходу-місця в мережу та викликаючи всі синхронізовані переходи, що залежать від неї. Лічильник часу підключено так, що кожен перехід у системі може спрацьовувати лише при включеному переході часу. Коли тимчасовий перехід увімкнено, він називатиметься одиницею часу.

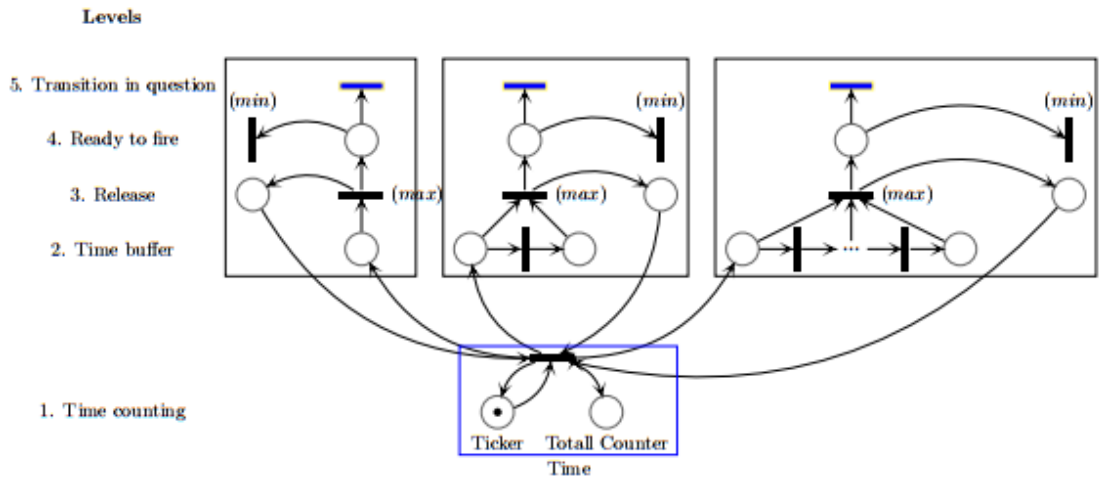

Рисунок 2.1 – Тимчасова структура, вбудована у конфігурацію стану

Як видно з рисунку 2.1 транзакції, може знадобитися будь-яка ціла кількість часу, перш ніж вона буде активована з точки зору обмеження time. На рисунку 2.1 зображено фактичну транзакцію, яка має розпочатися після цілого часу m, у вигляді синього прямокутника.

Сервіс-орієнтовані програми повинні дозволяти склади переходів, які можуть бути виконані за один крок за часом – як функції програмування, складені з наборів операцій. Таким чином, нам довелося створити блокуюче розширення, яке може бачити, чи дана транзакція може почати своє виконання і коли вона закінчила його.

Місце блокування з однією позначкою через перехід пов'язане з робочим місцем. Цей перехід також пов'язаний з усіма активаторами всіх необхідних переходів.

Якщо перехід не увімкнено, позначки зібрані, і перехід не спрацював. Інакше логіка переходу спрацьовує. Тепер ми можемо розглядати комбінацію переходів як окрему мережу, що працює як один негайний перехід. Після того, як усі необхідні операції закінчили виконання та готові повернути результати у вихідні місця, ми очищаємо «виконані» місця, забезпечені однією відміткою, і знімаємо блокування. Усі блокування дозволяють включити «роботу» не частіше ніж один раз на такт.

Таким чином, наведено (рисунок 2.2) розширення логіки блокування, яке дозволяє комбінувати переходи. На рисунку 2.2 вхідні дані композиції переходів зображені синім прямокутником, а вихідні – помаранчевими.

Узагальнена стратегія блокування дозволяє створювати функціональні транзакції, які складаються з наборів операційних транзакцій. Іншими словами, комбінація структури блокування, доступна для всіх таких груп транзакцій, які, по суті, зображають функції, у поєднанні з глобальним годинником часу дозволяють нам визначати складні функції відповідно до формалізму КМП.

Такі функції можуть бути визначені як $f(t, r, p, I) -> 0$, де t - необхідна ціла кількість тимчасових кроків, r передає пріоритет виконання функції

порівняно з іншими, *I* зображує вхідні покази можливість виконання, коли всі вимоги включені, а *O* відображає параметри вихідних переходів. Зверніть увагу, що будь-який такий функціональний перехід – це просто КМП-граф, деякі вузли якого визначені як Inputs, а деякі визначені як Outputs.

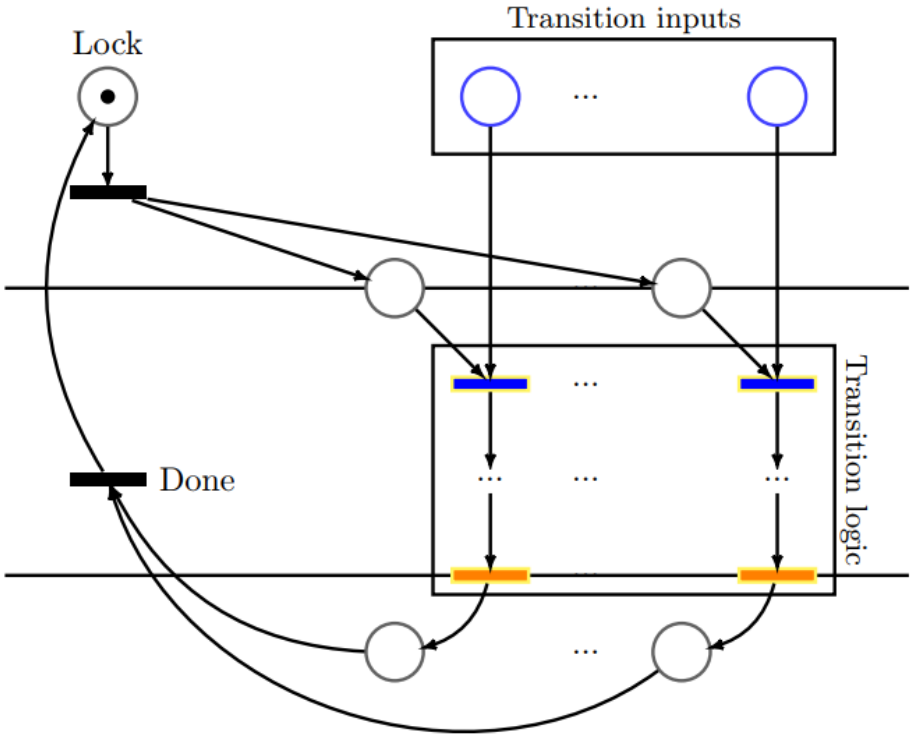

Рисунок 2.2 – Структура блокування переходу

Важливо мати можливість об'єднувати стільки переходів в один, скільки необхідно, щоб їхня комбінація могла спрацьовувати в передбачуваний час (за один або кілька системних тиків). Таким чином, функціональні перетворення можна комбінувати, припускаючи складні варіанти поведінки. Коли ми можемо бути впевнені, що наш функціональний перехід виконано за один тимчасовий крок, ми можемо почати розмірковувати про такий перехід, як графік КМП, відокремлений від його оточення.

Важливо, що всі необхідні вхідні дані такого підграфа повинні бути передані йому - заблоковані або використані, щоб інші переходи не могли їх

змінювати паралельно. Таким чином, ми можемо мати справу з "all" вхідними елементами розміщення як формою вимог функціональний перехід.

Рисунок 2.3 – Функціональні переходи для копіювання та переміщення вводу

Приклади простих функціональних переходів представлені на рисунку 2.3: "скопіювати всі вхідні дані з одного місця до іншого" і "перемістити всі елементи з одного місця до іншого".

Щоб реплікувати значення між двома вузлами, функціональний перехід копія, який зберігає всі вихідні елементи і клонує їх у місце виведення, зображений на рисунку 2.3. Аналогічно функціональний перехід "переміщення" дозволяє вилучити елементи-мітки з одного місця та помістити їх у інше.

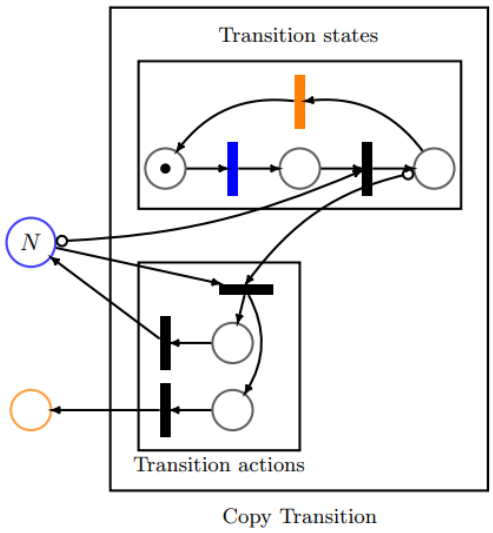

Рисунок 2.4 – Реалізація копіювання

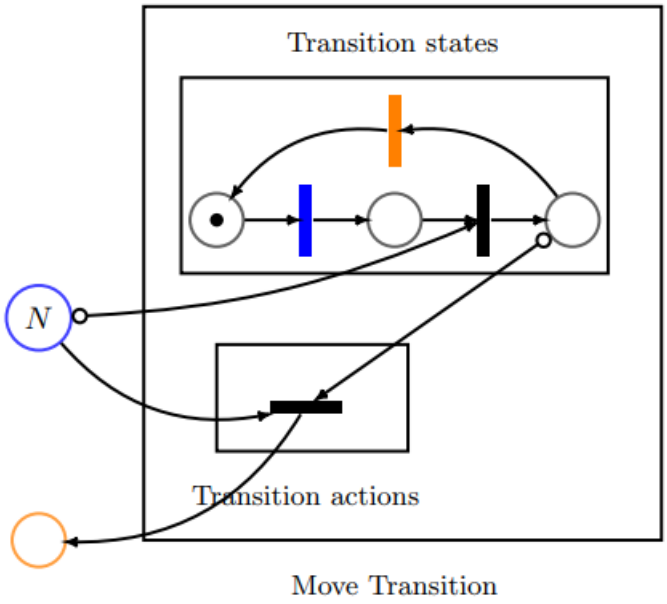

Рисунок 2.5 – Реалізація переміщення

Деталі графіка реалізації ММП наведені рисунку. 2.4. Тут ми показуємо:

- місце введення позначено синім кружком (всі його вхідні дані мають бути заморожені доти, як цей перехід спрацює);
- вхідний перехід по синьому прямокутнику, який включається, коли включено час, ймовірність та загальні блокування;
- місце виведення для всіх вхідних елементів у вигляді оранжевого кола;
- перехід, який виводить дані у вигляді помаранчевого прямокутника, дозволяє розблокувати систему після завершення її виконання.

Ход реалізації аналогічний рисунку 2.5.

Вкрай важливо мати можливість вибору варіанта залежно від усіх предметів у місці. Отже, нам потрібний функціональний перехід для перемикання.

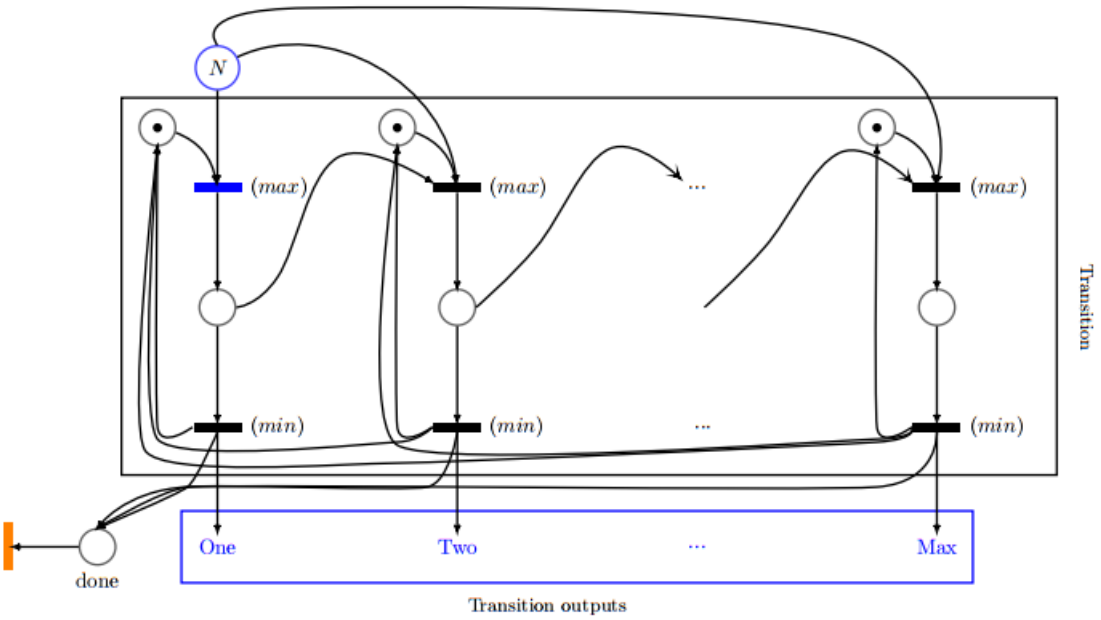

Рисунок 2.6 – Структура умов перемикання

Іншими словами, важливо мати можливість перемикатися в залежності від входу. Приклад такого перемикання представлено рисунку 2.6. Ми використовуємо пріоритети (у дужках поруч із переходами), щоб встановити пріоритет активації переходу та зняти блокування, коли перемикач вибрав значення.

Зверніть увагу, що цей підхід умови перемикання може використовуватися для реалізації всіх видів поведінки, включаючи переходи заборони для перевірки порожнечі. Основним недоліком такого графа є те, що для передачі з формалізмом МП потрібен фіксований максимальний розмір введення N. Тим не менш, оскільки це розширення, така ємність може бути встановлена і зафіксовано до виконання мережі ММП.

Оскільки ми можемо копіювати частини МП-графіків і комбінувати їх, ми можемо створювати функції, складені з наборів операцій.

Приклад можливості комбінування функціональної транзакції показано на рисунку 2.7.

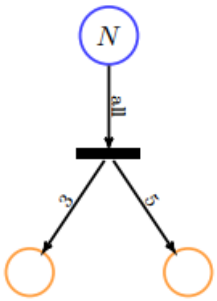

Рисунок 2.7 – Підмережа, що працює на вході та виводить значення const залежно від нього

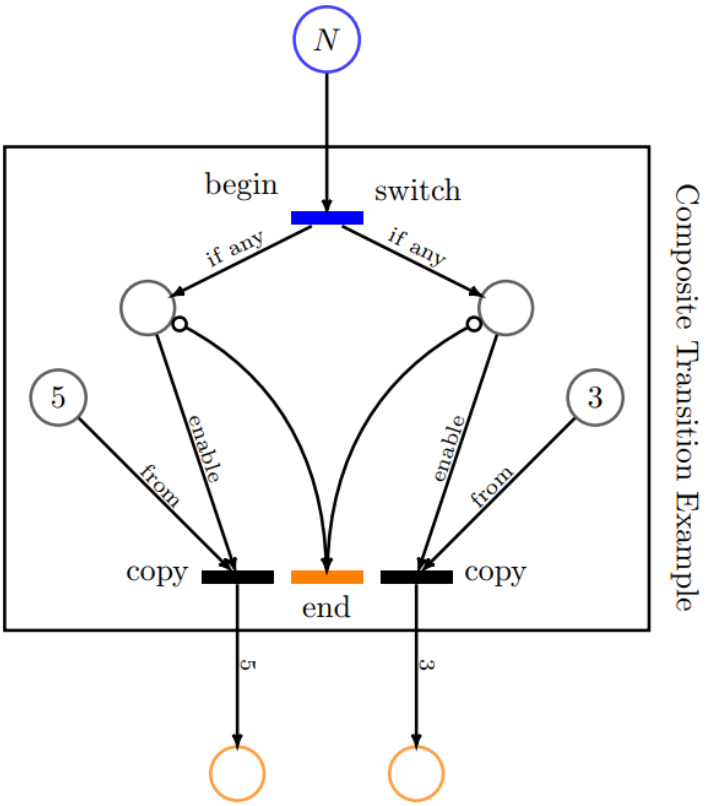

Рисунок 2.8 – Підмережа, що працює на вході та виході, постійне значення в залежності від варіанта її реалізації

Реалізацію докладно показано на рисунку 2.8: регістр перемикання, побудований на базі переходу «копіювання». Якщо набір F складений з функціональних переходів fi, таких що для будь-якого fi всі виходи переходу досяжні, а набір F складений так, що будь-який з його виходів може бути

досягнутий, то і всі виходи *F* досяжні.

Крім цілих чисел потрібно мати можливість виконувати безліч різних математичних операцій. У цій кваліфікаційній роботі представлено базовий набір цілих операцій, таких як {+, −, >, ==, <, *, /, %} на рисунку 2.9.

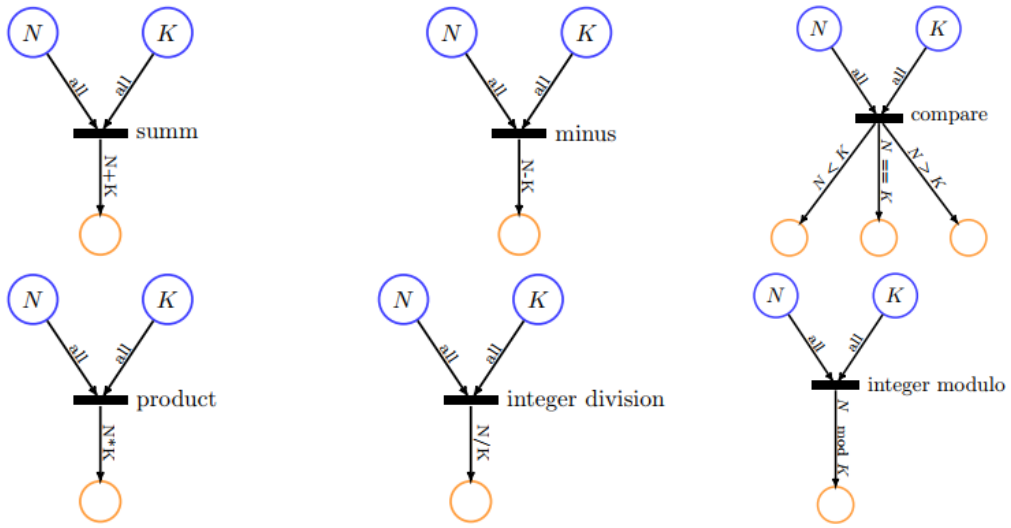

Рисунок 2.9 – Операції +, -, >, ==, <, *, /, %

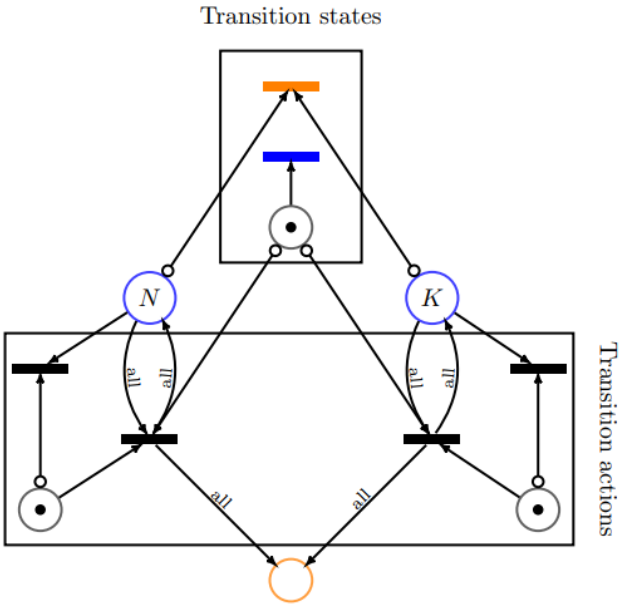

Рисунок 2.10 – Підмережа доповнення

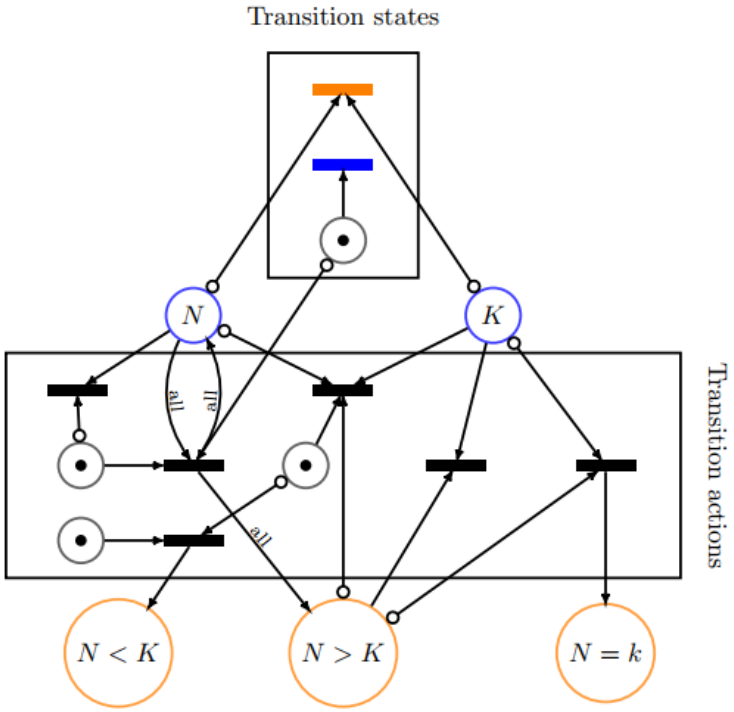

Рисунок 2.11 – Підмережа порівняння

Порівняння дозволяє нам взяти два вузли зі значеннями N і K і отримати будь-який з трьох результатів, а саме: $K == N$ або $K > N$ або $K < N$. Як показано на рисунку 2.11, ми виснажуємо вихідні значення місць, тому іноді операція копіювання може продовжуватися, як показано тут у деталях реалізації 2.11.

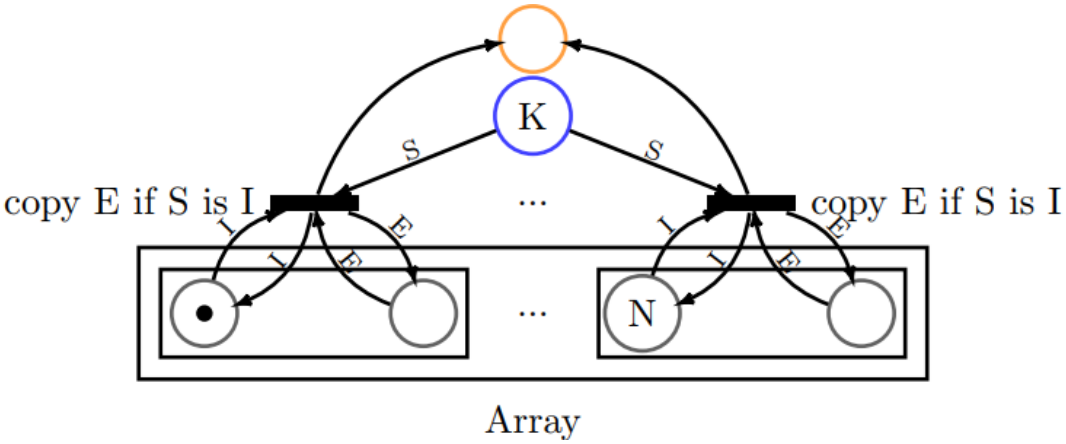

Рисунок 2.12 – Отримання елементу масиву по індексу

Складні типи, такі як масиви з фіксованою довжиною та операції над ними, можуть бути реалізовані у вигляді набору функціональних транзакцій поверх групи місць, як показано на рисунку 2.12. Тут показано операцію читання, інші реалізовуються аналогічно.

2.3.1 Розширення вкладених типів

Для опису розширення вкладених типів потрібний опис того, де знаходиться весь масив, ціле число або деяка інша комбінація місць, заданих за допомогою функціональних переходів поверх них. Таким чином, введено позначення для цього: type-place bonding. За аналогією з тим, як це здійснюється в програмуванні, в моделі ММП є поділ логіки потоку та управління пам'яттю. Користувач ММП керує абстракцією вищого рівня місць, які опосередковано пов'язані з екземплярами типів:

- для кожного місця, яке буде обробляти складні типи в мережі ММП, ми маємо унікальний цілісний адресний номер - тому ми називаємо його holder place;
- кожен тип складається зі своїх місць даних (поверх яких визначено функціональні переходи) та місця, де зберігається його адреса, його location Id;
- перед виконанням моделювання визначено максимально допустиму кількість екземплярів типу;
- коли транзакція переносить екземпляр типу з одного місця в інше, вона змінює ідентифікатор розташування екземпляра.

Таким чином, користувач ММП працює з графом екземплярів типів, прив'язаних до місць (аналогічно покажчикам). Коли операція про функціональний перехід активована, вона змінює або створює нові випадки не тільки шляхом завдання даних, але також шляхом зміни ідентифікатора розташування типу.

2.3.2 Структуровані типи

Пропонується додавання колбору до вузлів, використовуючи КМП як основу ММП, можна визначити структуру складного типу як набір кольорів. Так, наприклад, для типу даних:

Лістинг 2.1 – Приклад структури

```
struct data {
int number;
string text;
}
```

Отримаємо: $data = \{int, \{int1, ..., intn\}\}$ де n - максимальна довжина масиву, набір червоного кольору буде позначати тип цілого числа, набір синього кольору буде вказувати на елементи масиву, набір зеленого кольору роздільник між різними типами даних. Тут кольори дозволять здійснювати операції перетворення типів КМП: розділення та сумісності міток, у той час як розширення ММП дозволять перехідні функції, глобальні операції часу та блокування, прив'язку типу-місця.

2.3.3 Модулі

Будемо називати групу вузлів з виділеними "публічними" місцями та переходами шаблоном або модулем.

Типи, що переносяться, над ними забезпечують низькорівневу абстракцію для загального логічного опису та типів RPC. Тим не менш, коли йдеться про сервіс-орієнтовану мережу, дуже важливо також мати шаблони взаємодії, які можуть описувати взаємодії служб один з одним. Таким чином, далі представлено концепцію module.

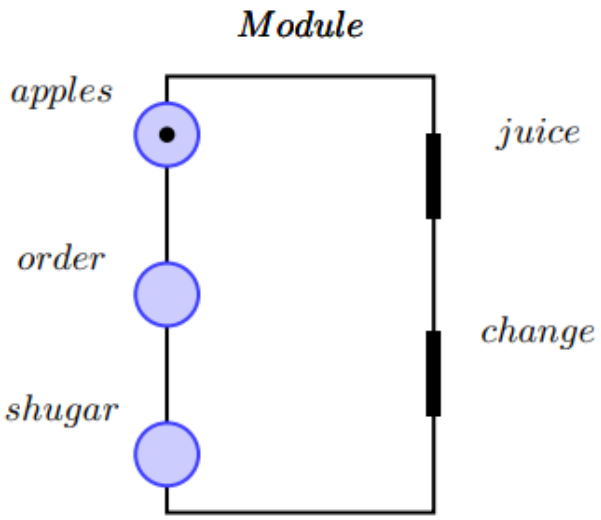

Рисунок 2.13 – Модуль, зображений у згорнутому вигляді - показані лише зовнішні елементи

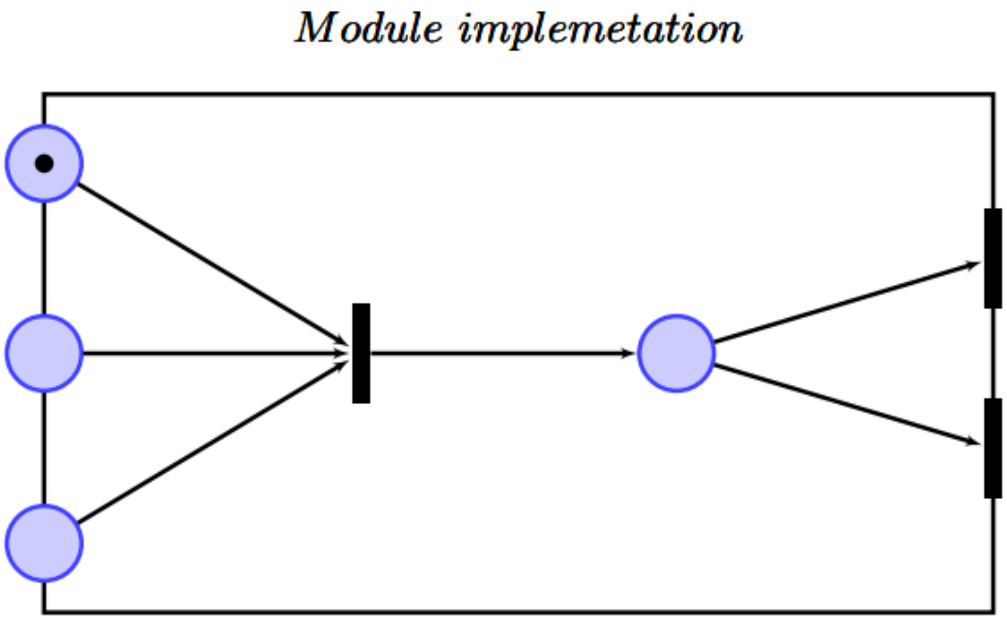

Рисунок 2.14 – Модуль, зображений у розгорнутому вигляді

Модуль - це підграф ММП, який складається з місць та переходів і може бути легко відтворено. Має загальнодоступні переходи та місця. Час усередині нього можна масштабувати шляхом множення часу всіх переходів константу. Приклад модуля можна побачити рисунку 2.14.

Проведемо виклик місця підграфа модулів ММП, підключених до модуля публічних переходів, і до місць цього модуля. Публічні місця - related. Нехай еволюція конфігурації набору місць S - це набір пар, що зіставляють мітки, які будуть знаходитися в цих місцях si,t, ймовірностям pi,t у різні моменти часу t. Розподіл еволюції конфігурації функціонального переходу (CED) $O(s)$ −> S означає, що для будь-якої заздалегідь відомої конфігурації s пов'язаних місць для даного модуля ми знаємо, які позначки будуть поміщені в його зв'язані місця через кількість кроків t.

Для формального моделювання взаємодії сервісних систем дуже важливою є можливість опису протоколів взаємодії вузлів. Це вимагає підтримки набору, здавалося б, найпростіших типів даних: цілі числа, числа з плаваючою комою, перерахування, рядки. Проте існуючі реалізації мереж Петрі, орієнтовані на математичну спільність, працюють із погляду перенесення одиниць інформаційних міток з позицій у переходи однієї за раз.

Цей принцип у своїх формальних визначеннях також спрямовує існуючі різновиди кольорових мереж Петрі, таких як CSPN та GCSPN. Щоб вирішити проблему опису сервісних взаємодій, розроблено нову модель сервіс-орієнтованої моделі мережі Петрі на основі стохастичних мереж Петрі, описаної в цьому розділі. Основною відмінністю нашого підходу є припущення про відомі максимальні розміри типів даних, що відображаються в моделі. Крім того, ми використовуємо метод для визначення часу у вигляді пари позначок положення, зазначених усередині системи.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МОДИФІКОВАНОЇ МЕРЕЖІ ПЕТРІ

Для декомпозиції розподілених сервісних систем за допомогою запропонованої моделі мовою C# було розроблено бібліотеку та комплекс прикладів моделювання. Загальна архітектура отриманого рішення представлена на рисунку 3.1.

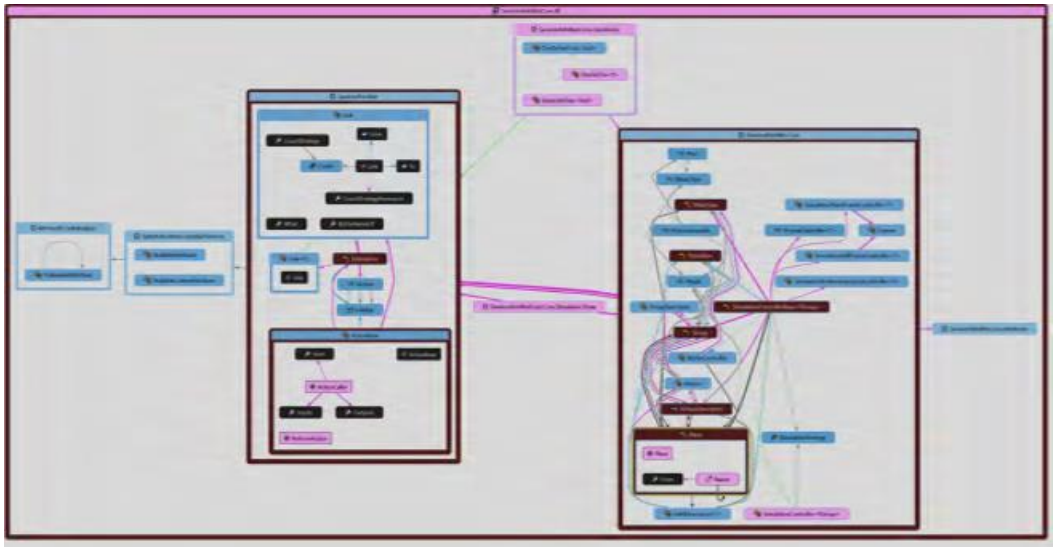

Рисунок 3.1 – Архітектура

На мові C# була реалізована бібліотека, її тести та приклади використання. Усі компоненти математичної моделі ММП знайшли реалізацію у архітектурі, представленої на рисунку 3.1.

Для роботи з великою кількістю різнорідних об'єктів в ММП застосовується організація даних, побудована на базі алгоритмів застосовуваних у завданнях зберігання пам'яті віртуальних машин:

- велика купа з об'єктами фіксованого розміру;
- до об'єктів надається доступ за вказівником;
- для створеної симуляції виділяється заданий циклічний буфер об'єктів заздалегідь задається розміру.

Розширення мереж Петрі – мітка будь-якого типу містить метадані про:

- ідентифікаційний номер;
- ім'я типу;
- статус об'єкта (новий, оновлений, виключений).

Для реалізації поставленого завдання пропонується використовувати представлену модель ММП, що розширює мережу Петрі. Наведемо приклад моделювання з допомогою ММП C#.

Для побудови розподілених систем у рамках запропонованої моделі слід зробити декомпозицію системи на п'ять основних шарів:

- шаблони взаємодії, представлені у системі.
- організація ресурсів, які має система.
- інфраструктура взаємодії вузлів системи.
- правила взаємодії додатків системи.
- розглянутий кейс роботи системи.

3.1 Компоненти моделювання

Основні компоненти практичного моделювання: групи, місця, переходи, типи міток та шаблони.

Група містить усі: вкладені групи, місця, переходи, типи позначок. За аналогією з програмою може бути запущена, змодельована покроково та зупинена.

Шаблони використовуються для спрощення створення груп, динамічної вставки місць та переходів. Шаблон надає креслення організації вузлів групи.

Типи міток ($MarkTypes$) можуть представляти будь-яку пасивну структуру даних (POD). Користувач визначає спадкоємців типу MarkTypes як типи збереження даних.

Переходи працюють із типами міток. Вони дозволяють генерувати, використовувати, складати та розбирати на складові зразки типів міток –

конкретні мітки. За аналогією до логіки виконання коду – перехід визначає, що брати, звідки та куди поміщати результати.

Дії описують правила того, що відбувається з типами міток під час активації переходів. Щоб функціонувати, кожен перехід має прив'язану йому дію. Аналогічно сервісу – дія реалізує логіку.

Лістинг 3.1 – Формування звіту про характеристики виробу (файл Entity.as)

```
using System;
using System.IO;
using System.Linq;
using ServicesPetriNet.Core;
using ServicesPetriNetCore.Core.Simulation.Draw ;
namespace ServicesPetriNet {
public class SimpleDemoProgram {
public class Mark: MarkType {
public int value;
}
public class Add:ActionBase {
public static Mark Action (Mark fromA , Mark fromB ) {
return new Mark {
value = fromA.value + fromB.value
};}}
public class Sample : Group {
public Place A, B, C;
private Transition Summ;
public Sample () {
Summ .Action <Add >()
.In <Mark >(A)
In <Mark >(B)
.Out <Mark >(C);
Marks = Extensions.At(A, MarkType. Create <Mark
>(5) )
.At(B, MarkType . Create <Mark >(6) );
}}
public static void Main () {
var simulation = new SimulationController <Sample >() ;
var s = simulation. DebugGraphToDot ();
Console . Write (s);
File. WriteAllText ("./ simple . dot " , s);
simulation . SimulationStep ();
var result = simulation . TopGroup.C. GetMarks (). First
() as Mark;
Assert. AreEqual (5 + 6, result . value );
Console. Write (" Simulation completed!");
Console . ReadLine ();
} } }
```

Місця містять екземпляри типів міток. Переходи беруть $Mark\ Types$ з $Places$ і поміщають нові мітки в них. Місця функціонують за аналогією з контейнерами, в яких зберігаються дані та забезпечується доступ до них на видалення, додавання та перевірку наявності. Мережі Петрі працюють із орієнтованими дводольними графами, які здебільшого з місць і переходів. Приклад мінімальної програми моделювання в лістингу 3.1.

Опишемо, що цей приклад робить з погляду коду:

- оголошується $PODMark\ \ TypeMark$. Його екземпляри можуть містити ціле значення;
- оголошується дія додавання Add, яка може працювати з мітками – взяти дві та повернути одну. Зверніть увагу, що тут жоден інтерфейс не обмежує, які аргументи приймає функція $Action$ та що вона повертає;
- оголошується група з трьох місць A, B, C та одного переходу $Summ$;
- у конструкторі групи ми створюємо графік того, як працюватимуть наші переходи;
- з'єднуємо входи: місця A та B з місцем виведення C;
- заявляємо, що наш перехід $Summ$ буде використовувати клас Add для виконання переходу екземплярів $MarkTypear$ - В основному методі $Main$ ми створюємо симуляцію групи;
- в метод DebudDraw для того, щоб зберегти топологію групи у файл $.dotGraphV\ iz$;
- моделюємо один крок;
- перевіряємо результати. Отримуємо граф, показаний на рисунку 3.2:

Для моделювання комплексів програм необхідно розуміти практичний зв'язок реальних компонентів систем з об'єктами, що застосовуються в моделі ММП.

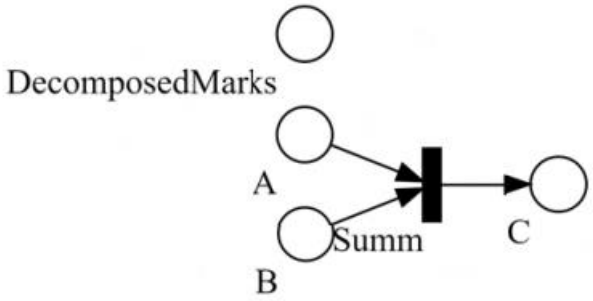

Рисунок 3.2 – Модель мережі для створеної програми

Для реалізації перевірки задоволення залежностей компонентів та послідовного виконання операцій паралельних систем пропонується використовувати комбінації місць та переходів. Для реалізації складної поведінки сервісів, такої як передплата каналів повідомлень, необхідна можливість створення ключів, що дозволяють форсувати деяку поведінку лише один раз.

Для можливості розгляду метаморфози від інформаційного повідомлення, що надсилається одним сервісом іншому, до пакетів даних, що передаються по мережі, модель дозволяє застосувати декомпозицію, реалізувати можливість втрат пакетів при передачі та реконструкцію вихідних даних з набору пакетів.

Для реалізації перевірок необхідна можливість аналізу вмісту місць щодо наявності міток, потрібних для активації кожного конкретного переходу. У сервісних системах велику роль грає можливість розсилки повідомлень як усім, і лише одному сервісу, обраному за певним правилом з набору. Це є причиною використання в описі моделі системи угруповань сервісів, об'єднаних у модулі.

У запропонованій моделі набори міток та переходів між ними можуть бути об'єднані у групи. У межах цих груп місцям надаються ідентифікатори – позначення, що дозволяє здійснювати адресацію у межах груп. Те саме місце може ідентифікуватися одночасно по-різному в різних групах – грати роль. Тим самим здійснюється можливість об'єднання груп для отримання єдиного

графа зв'язків та залежностей системи. Усі групи за замовчуванням розташовуються в єдиній супергрупі.

Для здійснення моделювання роботи сервісної системи дуже важливо поняття часу. Для різних груп компонентів, об'єднаних в єдину сервісну систему, час, що витрачається на операції - переходи системи з одного стану в інший, може змінюватись щодо розмірності часу фізичного світу. У даній моделі ми називатимемо тіком – мірою часу – одне виконання передачі мітки від місця до переходу у випадку, якщо це можливо хоча б для одного переходу, пов'язаного з міткою, виконаного кожним місцем у групі.

Таким чином, час визначається локально для кожної групи. Це означає, що для здійснення передачі повідомлення від одного сервісу до іншого через групу, пов'язану з мережею, один тик між сервісами може дорівнювати тисячам тиків між мережними пристроями, що їх з'єднують. При об'єднанні груп на єдиний граф швидкість руху локального часу для колишніх груп зберігається.

Важливою концепцією сервісних систем є можливість передплати сповіщення та їх своєчасне отримання. Так, для реалізації механізму подій пропонується використання спеціальних місць, які зберігатимуть у собі мітки – повідомлення про події – протягом лише одного тику після події. До таких міток з інших груп можливе підключення виключно дуги умов.

У аналізованої системі перехід є групою, що складається з двох окремих блоків – перевірки умови та обробки даних. Блок перевірки умови видає подію, що сигналізує про можливість переходу, у той час як блок обробки даних містить три події: повідомлення про отримання необхідних міток, повідомлення про процес їх обробки, повідомлення про закінчення виконання переходу, яке виконується після розсилки міток. Обробка переходом отриманої мітки виконується функцією переходу, яку накладаються обмеження вхідних даних. Вона працює в рамках контексту глобальних змінних, поточного часу групи, в якій вона знаходиться, а також аргументів переходу у вигляді міток та сигналів дуг умов.

Місця традиційних мереж Петрі розширюються наданням ймовірностей активації дуг, які ведуть переходам. В рамках одного тику системи на кожному конкретному місці застосовується наступний алгоритм роботи: дуги, що йдуть від місця, перевіряються на можливість виклику; після чого переходи, пов'язані з цими дугами, перевіряються на можливість активації; так відбувається для кожної дуги, що йде від місця, поки не знайдеться дуга з доступним до виконання переходом або всі дуги не будуть випробувані один раз. Якщо кілька місць пов'язані з одним переходом, перед перевіркою переходу перевіряються ймовірності виклику дуг усім місцях, що з цим переходом.

Для роботи з масивами місць у групах використовується шаблон паралельного програмування stencil†, що дозволяє задати кожному елементу масиву місць у групі логіку переходів і зв'язків. Групи дозволяють задавати використання масивів міток як конкретних чисел і елементів, і залишаючи місця визначення конкретного переліку елементів ззовні. Такі групи вважаються неповними.

Приклад неповної групи в рамках сервісних систем є шаблон передплатник-респондент, де всім заздалегідь не відомим передплатникам надсилається повідомлення. Конкретна кількість передплатників задається під час роботи системи. Важливим моментом є об'єднання шарів моделі в один - на цьому моменті здійснюється перевірка моделі на відсутність у ній неповних груп. Це дає гарантію того, що в рамках моделі кожної конкретної системи завжди точно відома кількість міток та переходів.

При використанні груп, що містять різні шляхи, що дозволяють мітці перейти від одного місця до іншого, передбачена можливість вибору шляхи для мітки з урахуванням обраної для кожної групи евристичної функції. Це дозволяє моделювати системи, що містять у собі кілька мережних комутаторів, що поєднують одні й ті ж вузли, наприклад, InfiniBand та Ethernet.

Низка важливих практичних відмінностей отриманої моделі полягає в

наступному:

запропонована мережа є кольоровою у сенсі, розширеному для практичного застосування – мітки містять пари, що складаються з «типу-ключа» - маркера кольору, якому зіставляється конкретне значення. Таким чином, перехід, розглядаючи один колір мітки, може приймати різні рішення виходячи з конкретних значень її ключів;

на відміну від ієрархічних мереж Петрі, модель, що розглядається, оперує концепцією груп, які при агрегації повної моделі «розвертаються», об'єднуючись в єдиний граф мережі;

порівняно з дуалістичними мережами Петрі (dPNs), у запропонованій моделі приділяється особлива увага можливості множинних шляхів між місцями, дозволяється виділення цільової функції – евристики – при виборі шляху, який необхідно пройти мітці між різними групами.

Методика описує:

- об'єкти, що підлягають моделюванню;
- межі можливостей моделювання;
- питання, що ставляться під час вивчення поведінки при різних параметрах моделювання: з погляду масштабування, обробки помилок і збільшення обсягу даних.

Це дозволяє отримати числові характеристики для оцінки працездатності та вузьких місць створюваної системи. Методика застосовна різних етапах життєвого циклу системи. Вона дає можливість сформувати допуски за продуктивністю компонентів і здійснити пошук точок перегину при масштабуванні.

3.2 Моделювання законів

Коректність роботи представленої моделі виявляється у відповідності до чисельних показників, одержуваних при виконанні моделювання з використанням ММП зі значеннями аналітичних формул для законів Амдала

та Густафсона. Як можна побачити із рисунку 3.3 значення, отримані під час моделювання (точки), відповідають значенням аналітичної формули закону Амдала.

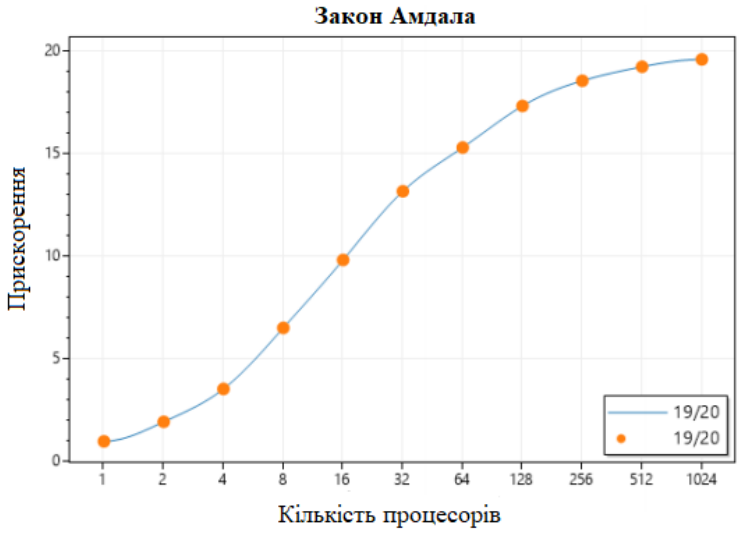

Рисунок 3.3 – Моделювання закону Амдала

Також на рисунку 3.4 показано як значення, що отримуються в ході моделювання (точки) відповідають значенням аналітичної формули закону Густафсона.

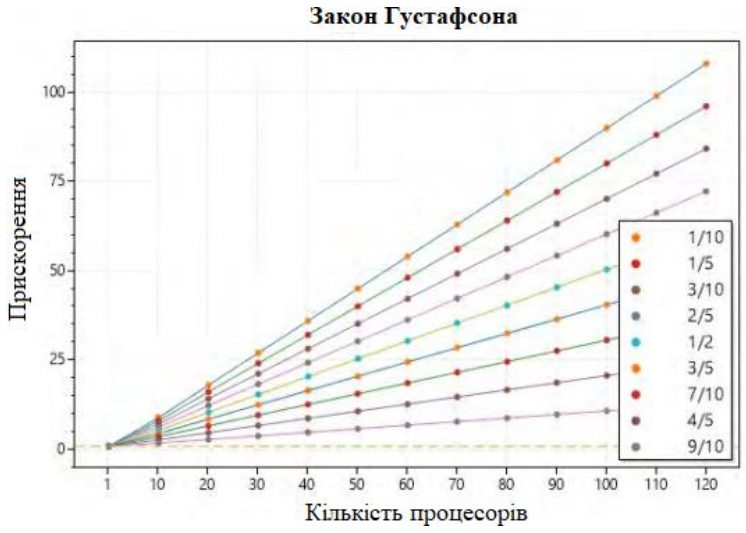

Рисунок 3.4 – Моделювання закону Густафсона

3.3 Верифікація моделей

Для верифікації роботи ММП моделей додатків була розроблена система, що дозволяє здійснювати запуск та здійснювати моніторинг роботи розподілених сервісних програм. Пряме виконання розподілених застосункуів здається перспективним підходом для аналізу розв'язків задач із використанням сервісів у розподіленому обчислювальному середовищі. Створено розробку, яка є самостійною платформою для виконання розподілених програм. Перед введенням в експлуатацію програми потребують ретельного тестування. Якщо програма є розподіленою, то крім коректності роботи кожного з її компонентів необхідно враховувати їх мережеві взаємодії.

Робота відбувається за таким алгоритмом:

- за допомогою API мовою Python створюється «Лабораторія» – опис моделі мережі, що включає список вузлів мережі, їх розподіл по підмережах і параметри запуску. Інформація, що описує лабораторію, зберігається в JSON-файлі;

- "лабораторія" запускається, створюючи мережеві вузли на базі Docker контейнерів та мережевих зв'язків між ними;

- відображення емульованої мережі доступне розробнику у веб-інтерфейсі для проведення та спостереження за перебігом експериментів.

Мережу можна уявити у вигляді дерева, вузли якого – роутери та клієнти. Так виглядатиме мережа для значень змінних висоти 3 та кількості листя графа, що дорівнює 3 (рисунок 3.5). Побудова такої мережі виконується за таким принципом: граф розширюється за допомогою кількох ітерацій додавання підмереж для дочірніх вузлів, наприкінці кожної з яких масив зовнішніх вузлів оновлюється. Для всіх вузлів, крім зовнішніх, настроюється робота протоколу динамічної маршрутизації. Це дозволить трафіку проходити через дерево. Кожному дочірньому вузлу графа ставиться у відповідність Docker-образ, який буде використаний для створення вузла

клієнта. Ще один крок, який необхідно зробити для коректної роботи віртуальної мережі, – вказати адресу сервера для кожного клієнта, щоб вони могли взаємодіяти. Після збереження конфігурації збудованої мережі можна спостерігати за роботою програми за допомогою графічного інтерфейсу.

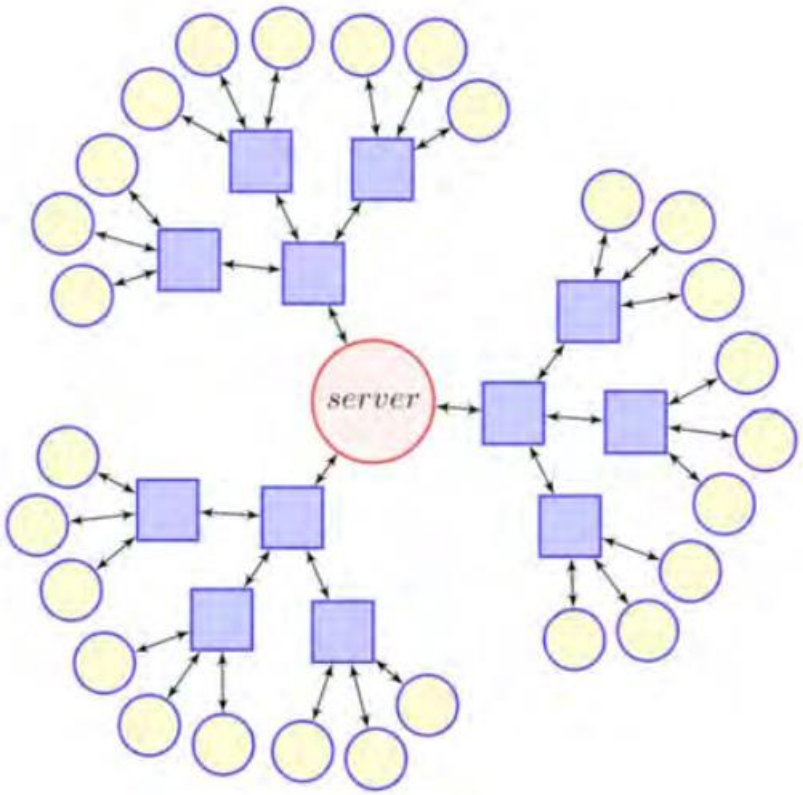

Рисунок 3.5 – Модель мережі

При запуску моделі з'являється можливість задавати неполадки мережі вручну та спостерігати за тим, як програма реагує на них. Щоб настроїти стан мережі, необхідно перейти до будь-якого вузла та встановити параметри перешкод за допомогою вікна, що відкрилося (рисунок 3.6), а потім натиснути кнопку tcset. Наприклад, щоб роутер втрачав 50% пакетів, які йому приходять, потрібно встановити відповідне значення поля loss. Поставивши неполадки для кількох вузлів, можна спостерігати за результатом на графі. Колір вузла залежить від того, який статус у повідомлення, що надійшло. Зелений колір сигналізує про нормальну роботу,

жовтий – про неполадки, червоний – про критичні проблеми, а фіолетовий – про несподівані винятки.

Рисунок 3.6 – Моделювання клієнт-серверної архітектури

Розроблена система тестування дозволяє:

- створювати реалістичні моделі IP-мереж великого розміру;
- розгортати поверх них розподілені програми на базі Dockerобразів;
- керувати якістю роботи окремих ділянок моделюваної мережі;
- візуалізувати стан працюючого в моделі розподіленого додатка в реальному часі. Іншими словами, продукт може бути використаний для порівняння роботи кількох розподілених додатків в однакових умовах, перевірки стійкості програми щодо нестабільної роботи мережі, вивчення впливу структури мережі на його роботу або для перевірки наявності уразливостей мережного план.

Значення, отримані в моделі, побудованій на базі створеного інструменту моделювання МП C#, були зіставлені на значення, отримані на базі ММП для топології FatTree. Модель в ММП містила три основні шари

вкладеності:

- рівень лінків-комунікації;
- рівень вузлів;
- рівень програм.

Приклад візуалізації шарів системи представлений рисунку 3.7.

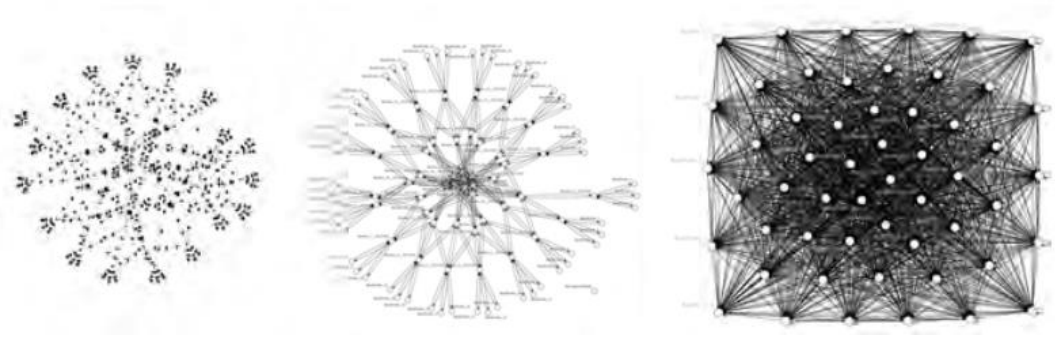

Рисунок 3.7 – Рівні архітектури застосунку

Було здійснено пряме виконання системи з аналогічним кількістю вузлів, повноцінними образами, що моделюють вузли та зв'язку, OpenMPI застосунком, що виконується на вузлах. Отримані результати в залежності від кількості зв'язків, що не працюють, і втрачених пакетів представлені на рисунку 3.8.

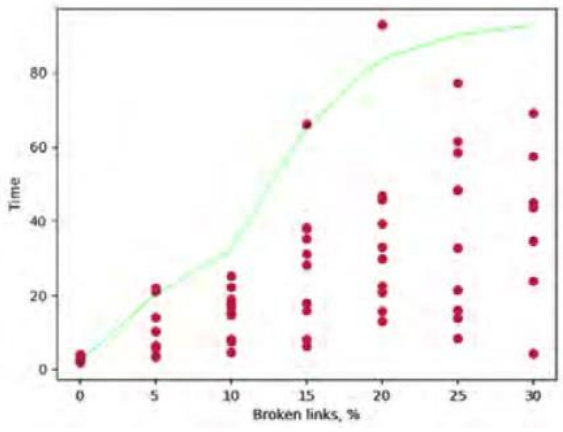

Рисунок 3.8 – Результати моделювання

Наведено опис практичних аспектів застосування моделі та сформовано методику опису об'єктів, що підлягають моделюванню, межі можливостей моделювання, питання, які ставляться у ході вивчення поведінки за різних параметрів моделювання: з погляду масштабування, обробки помилок та зростання обсягу даних. Таким чином, представлена модель ММП та її реалізація ММП C# дозволяє отримувати числові характеристики для оцінки працездатності та вузьких місць створюваних систем.

ВИСНОВКИ

Проведена розробка математичного та програмного забезпечення обчислювальних машин для створення та супроводу розподілених сервісних систем. Представлена дискретна стохастична модель, позначена як ММП, яка поєднує у собі якості кольорових, ієрархічних та узагальнених мереж Петрі.

Модель є рядом розширень над базовим апаратом стохастичних мереж Петрі, що дозволяє прямого переходу від моделі ММП до базової моделі стохастичних мереж Петрі. У моделі було введено ряд додаткових термінів для опису угруповання елементів ієрархії компонентів сервісних систем.

Розроблена модель має низку нових властивостей для опису складних обчислювальних систем на базі мереж Петрі. Так, наприклад, повідомлення в сервісних системах здатні розбиватися на пакети, частина яких втрачається і відновлюється в ході передачі, а на стороні одержувача пакети повинні збиратися назад у цілісне повідомлення. Для реалізації цього завдання необхідна підтримка важливої, нової для мереж Петрі властивості: декомпозиції та злиття елементів даних-міток. Для цього, у свою чергу, потрібна можливість ідентифікації міток, представлена в запропонованій моделі ММП.

Для коректної роботи зі складними типами даних модель дозволяє розглядати різні рівні логіки взаємодії компонентів систем та виробляти моделювання потенційних проблем як програмного, так і апаратного рівнів. Було створено спосіб організації об'єктної моделі з урахуванням мереж Петрі для моделювання програмних систем, заснований на розподілі графа логіки операцій над елементами та місцями їх зберігання. Представлений метод дозволяє вирішувати поставлені завдання, зберігаючи зворотну сумісність із моделлю стохастичних мереж Петрі. На основі отриманого рішення можна здійснювати представлення складних шаблонів взаємодії сервісів.

У ході вирішення завдання створення практичної реалізації для

апробації представлених підходів розроблено бібліотеку моделювання ServicesPetriNet (ММП C#) з відкритим вихідним кодом, що дозволила:

- реалізувати програмний опис моделей ММП мереж Петрі;
- верифікувати та відтворювати результати;
- здійснювати візуалізацію та налагодження створюваних моделей;
- протестувати роботу із законами Амдала та Густафсона;
- побудувати модель архітектури FatTree;
- створити бібліотеку моделей шаблонів мережевої взаємодії.

Достовірність моделі підтверджується відповідністю поведінки збудованих дискретних моделей аналітичним показниками для законів Амдаля та Густавсона. У роботі поведено приклад випробування системи на архітектурі MPI програми, що працює на кластері з архітектурою FatTree та зміни її поведінки при варіації помилок мережного устаткування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Масленіков Д.Є., Лебедєв О.Г., Дяченко В.О. Модель та метод опису розподілених систем // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Тези доповідей дванадцятої міжнародної науково-технічної конференції 27-28 квітня 2022 р., т.1. Баку-Харків-Жиліна. 2022 р. С.73.

2. Jensen, K. Coloured petri nets / K. Jensen // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1987. Vol. 254. P. 248–299.

3. Jensen, K. Coloured Petri Nets: Modelling and Validation of Concurrent Systems / K. Jensen, L.M. Kristensen. Springer Berlin Heidelberg, 2009. P. 384.

4. Jensen, K. Coloured petri nets and the invariant-method / K. Jensen // Theoretical Computer Science. 1981. Vol. 14, N. 3. P. 317–336.

5. David, R. Petri nets for modeling of dynamic systems. A survey / R. David, H. Alla // Automatica. 1994. Vol. 30, N. 2. P. 175–202.

6. Gischer, J. Shuffle Languages, Petri Nets, and Context-Sensitive Grammars / J. Gischer // Communications of the ACM. 1981. Vol. 24, N. 9. P. 597–605.