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)

Моделі паралельного та розподіленого моделювання в хмарних системах
(тема)

Виконав:
студент II курсу, групи СПм-22 - 4
Самойлов І.А.
(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: проф. Волк М.О.
(посада, прізвище, ініціали)

Допускається до захисту

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

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Самойлову Івану Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі паралельного та розподіленого моделювання в хмарних системах

затверджена наказом по університету від "01" квітня 2024 р. № 257ст

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

3. Вхідні дані до роботи

1. Технології забезпечення паралельного та розподіленого моделювання
2. Мікросервіси та контейнеризація.
3. Специфікації системи дискретних подій (DEVS)

4. Перелік питань, що потрібно опрацювати в роботі

1 Аналіз предметної області

2 Розробка моделей паралельної та розподіленої архітектури

3 Розгортання хмарного середовища

4 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати – 12 арк. ф. А4

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	01.04.24 – 04.04.24	
2	Розробка моделей	05.04.24 – 15.04.24	
3	Реалізація алгоритмів	16.04.24 – 28.04.24	
4	Розробка структури програмних засобів	29.04.24 – 15.05.24	
5	Розробка програмних модулів	16.05.24 – 25.05.24	
6	Оформлення матеріалів кваліфікаційної роботи	26.05.24 – 05.06.24	
7	Подання кваліфікаційної роботи керівникові та	07.06.24 – 12.06.24	
	попередній захист		
8	Подання кваліфікаційної роботи на	13.06.24 – 15.06.24	

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Волк М.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 61 с., 14 рис., 1 дод., 33 джерела.

ХМАРНІ СИСТЕМИ, РОЗПОДІЛЕНЕ МОДЕЛЮВАННЯ, МОДЕЛЬ, РОЗПОДІЛЕННЯ РЕСУРСІВ, ВИСОКОПРОДУКТИВНІ ОБЧИСЛЕННЯ

Хмарна інфраструктура забезпечує швидке надання ресурсів для обчислень на вимогу користувача. Середовище хмарного моделювання сьогодні широко використовується для імітації складних систем і забезпечує віддалений доступ до потужних обчислювальних ресурсів. Одною з проблем є масштабованість обчислювальних ресурсів для моделювання та імітації (M&S), яку можна вирішити за допомогою еластичності розгортання хмари на вимогу користувача. Однак, реалізуючи високу продуктивність, хмарна структура M&S, дотримуючись цих еластичних принципів, не є тривіальним завданням розпаралелювання за існуючими архітектурними рішеннями.

Дійсно, як паралельні, так і розподілені обчислення M&S розвивалися різними шляхами. Паралельні рішення завжди були зосереджені на спеціальних рішеннях, тоді як загальні підходи, з іншого боку, призвели до визначення стандартів на розподілені фреймворки, прикладом яких є архітектура високого рівня (HLA), або вплинули на використання розподілених технологій, як інтерфейс передачі повідомлень (MPI). Кваліфікаційна робота представляє уніфіковану паралельну та розподілену M&S архітектуру з достатньою гнучкістю для розгортання паралельних і розподілених симуляцій у хмарі без зміни базової моделі вихідного коду і досягнення важливих прискорень порівняно з послідовним моделюванням, особливо при паралельній реалізації. Вона заснована на формальної специфікації системи подій (DEVS) і використовує xDEVS M&S і тест DEVStone з тестуванням на десяти обчислювальних вузлах.

ABSTRACT

Master’s thesis: 61 pages, 14 figures, 1 appendice, 33 sources.

CLOUD SYSTEMS, DISTRIBUTED SIMULATION, MODEL, RESOURCE ALLOCATION, HIGH PERFORMANCE COMPUTING

Cloud infrastructure provides fast provision of resources for computing on demand of the user. The cloud simulation environment is widely used today to simulate complex systems and provides remote access to powerful computing resources. One challenge is the scalability of computing resources for modeling and simulation (M&S), which can be solved by the elasticity of cloud deployment on user demand. However, while realizing high performance, the M&S cloud structure, following these elastic principles, is not a trivial task to parallelize according to existing architectural solutions.

Indeed, both parallel and distributed M&S computing have evolved along different paths. Parallel solutions have always focused on ad hoc solutions, while generic approaches, on the other hand, have led to the definition of standards for distributed frameworks, exemplified by High Level Architecture (HLA), or have influenced the use of distributed technologies such as Message Passing Interface (MPI) . The certification work presents a unified parallel and distributed M&S architecture with sufficient flexibility to deploy parallel and distributed simulations in the cloud without changing the basic model of the source code and achieving important speedups compared to sequential simulations, especially in parallel implementation. It is based on the Event System Formal Specification (DEVS) and uses M&S's xDEVS and the DEVStone benchmark with testing on ten compute nodes.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВМ – віртуальна машина

ОС – операційна система

ПЗ – програмне забезпечення

AKS – Azure Kubernetes Service

API – Application Programming Interfase

DIS – Distributed Interactive Simulation

DEVS – Discrete Event System Specification

IaaS – Infrastructure as a Service

EKS – Elastic Kubernetes Service

ID – identity

FCFS – First-Come First-Served

FPGA – Field-Programmable Gate Array

GKE – Google Kubernetes Engine

HAL – Hardware Abstraction Layer

HLA – High Level Architecture

HPC – High Performance Computing

HPF – Highest Priority First

M&S – Modelling and Simulation

SaaS – Simulation as a Service

VM – Virtual Machine

VPN – Virtual Private Network

ВСТУП

Паралельне та розподілене моделювання — це два різних напрямка, які виникли у 1970-х та 1980-х роках відповідно від двох різних дослідницьких спільнот [1]. Спільнота Parallel Simulation була зосереджена на прискоренні моделювання шляхом використання ресурсів високопродуктивних обчислень (HPC). Відповідно, паралельне моделювання визначається як розпаралелювання моделювання між різними обчислювальними вузлами. Коли є значна географічна розділення між обчислювальними вузлами, паралельне моделювання перетворюється на розподілене моделювання. Хоча рішення паралельних обчислень є неявно розподіленим, зворотне не завжди вірно. Комунікація розподіленого моделювання (незалежно від аспекту паралельного моделювання) значною мірою зосереджена на з'єднання часткових симуляцій через локальні або глобальні мережі. Наразі ці два напрями продовжують зберігати ту саму тенденцію: паралельне моделювання працює переважно над тісно пов’язаними апаратними об’єктами, тоді як distributed simulation все ще працює на слабо зв’язаних компонентах через глобальні мережі на основі стандартів (наприклад, розподілене інтерактивне моделювання (DIS), архітектура високого рівня (HLA) тощо).

Бажання створити як паралельні, так і розподілені M&S стикається з новими викликами через складність нових додатків і еволюцію основного апаратного забезпечення [2]. З прикладної точки зору, імітація систем з постійно зростаючою складністю, як-от Інтернет речей, потребує величезних обчислювальних ресурсів [3].

З іншого боку, з апаратної точки зору, нові парадигми, такі як Cloud Computing, дозволяють забезпечити велику обчислювальну потужність. Інфраструктури Google або Amazon надають досліднику доступ до ресурсів для виконання моделювання [4]. Однак технології хмарних обчислень

вимагають спеціального супроводу, а додатки M&S повинні розвиватися, щоб адаптуватися до хмарних архітектур [5].

Паралельне та розподілене моделювання в хмарі є новою областю досліджень, керується економічною перевагою моделювання та масштабуванням на вимогу обчислювальних ресурсів, не зазнаючи витрат на придбання та експлуатацію високопродуктивних обчислювальних платформ. Проблеми, які перешкоджали прийняттю цих паралельних і розподілених технологій моделювання в минулому [6]. Відповідно до визначення Національного інституту стандартів і технологій США (NIST), Хмарні обчислення – це модель для забезпечення повсюдного, зручного мережевого доступу на вимогу до спільного пулу конфігурованих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ даних, програм та служб), які можна швидко надати та вивільнити з мінімальними зусиллями з управління або взаємодії з постачальником послуг[7]. Враховуючи неоднорідність, впровадження відповідної обчислювальної інфраструктури моделювання є дуже складним завданням.

Попередні механізми M&S були розроблені в основному для вирішення глобальних проблем як прозорість, моделювання як послуга, вартість і продуктивність [8]. Рішення, представлене в цієї роботі, використовує категоріальний поділ моделювання та симуляції в будь-якій архітектурі M&S і зосереджується на одній моделі, яка виконується через паралельне або розподілене виконання, тобто задається стандартна модель, яка повинна моделюватися послідовно, паралельно або розподілено, не змінюючи жодного рядка вихідного коду моделі.

Для досягнення цієї мети:

- модель має бути визначена відповідно до стандартних специфікацій;

- механізм моделювання та модель повинні бути відокремлені;

- технологія моделювання має бути достатньо стійкою, щоб легко вирішити проблему диверсифікації обчислень, яку пропонує хмара: віртуалізація, контейнеризація та ін.

Існує кілька стандартів M&S, які допомагають впоратися з першою задачею. Серед них ми вибрали Специфікацію системи дискретних подій (DEVS) [9], оскільки вона забезпечує не лише глобальну структуру для визначення моделей, але й стандартні механізми для проведення процесу моделювання, який категорично відокремлений від моделі, що відповідає другої задачі.

Хоча паралельне рішення завжди легко розгорнути, для розподіленої програми в процесі моделювання викликають багато технічних труднощів під час розгортання розподіленого симулятора. Для кластера - набір віртуальних машин або контейнерів є кращим механізмом розподілу. Наше рішення використовує просту розподілену архітектуру, шаблон клієнт/сервер із використанням стандартних сокетів. Було розроблено уніфіковану паралельну та розподілену архітектуру моделювання, яка керується подіями, де послідовне моделювання можна масштабувати до паралельного та розподіленого моделювання, з надзвичайно простим механізмом розгортання, навіть у хмарному середовищі.

Основний внесок нашого дослідження можна підсумувати таким чином: пропонується використання уніфікованої паралельної та розподіленої архітектури моделювання, що заснована на стандарті DEVS та реалізована засобами xDEVS Tool та API [10].

Після реалізації моделі її можна моделювати на послідовних, паралельних або розподілених платформах, не змінюючи жодного рядка вихідного код моделі.

Розроблено стандартну схему розгортання моделювання, де за допомогою XML-файлів, що визначають схему розподілу, симуляцію можна розгортати на паралельних або розподілених платформах.

Мультимодальне розгортання є дуже стійким, тобто його можна виконати за централізованими або хмарними ресурсами, включаючи фізичні (реальні) або віртуальні машини, або контейнери з використанням запропонованого методу та структури.

Стандартний тест DEVStone представлено для оцінки паралельних і розподілених моделей на основі DEVS моделювання.

Оцінка пропонованої архітектури проводиться враховуючи традиційні показники продуктивності, також розподіл ресурсів враховує вартість рішення.

Кваліфікаційна робота організована таким чином. Аналіз пов’язаних робіт наведено у розділі 1. Розділ 2 представляє моделі, що лежать в основі роботи, детальний огляд паралельної та розподіленої архітектури, реалізованої в xDEVS. Розділ 3 показує варіанти паралельного та розподіленого розгортання та експериментальні дослідження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Паралельне та розподілене моделювання

Можна знайти безліч паралельних і розподілених архітектур моделювання в літературі. Паралельна симуляційна парадигма зазвичай використовує рішення на основі технологій багатопотокового програмування [11] [12], хоча більш конкретні рішення можна знайти в останнє десятиліття з використанням графічних процесорів (GPUs) [13] або навіть програмованих вентильних матриць (FPGA) [14].

З іншого боку, розвідку парадигми розподіленого моделювання сприяли не тільки розподілені технології, такі як інтерфейс передачі повідомлень (MPI) [15], але й загальні та надійні стандарти розподіленого моделювання, такі як DIS [16] або HLA [17].

Що стосується формалізму DEVS M&S, який використовується в цій роботі, багато симуляторів були розроблені та опубліковані протягом останніх двадцяти років.

Деякі з них були спеціально розроблені для обробки паралельних або розподілених симуляцій.

Що стосується паралельних реалізацій DEVS, ми можемо знайти роботи Лю[18], Nutaro [19] або Lanuza [20] серед інших. Ці розробки засновані на оптимістичних методах за концепцією логічних процесів, алгоритмах планування часу або інших. Однак ці підходи не забезпечують стандартних інтерфейси для полегшення оцінки ефективності та порівняння за допомогою паралельних або розподілених тестів. Розробка двигуна xDEVS M&S бере свій початок з роботи про уніфікований процес DEVS [21] і протягом багатьох років був розширений, щоб додати в DEVS різні доменно-спеціальні мови (DSL) [22].

Попередні підходи [23] розпаралелювали стандартне моделювання

DEVS-циклів, які викликають функції переходу та виведення, зберігаючи оригінальну специфікацію DEVS і всі її властивості. Кодова база наразі підтримується в [10].

Стосовно розподілених реалізацій DEVS, деякі фреймворки, наприклад DEVS/SOA [24] або CD++ [25], які наразі не підтримуються, базувалися на концепція Simulation as a Service (SaaS), тоді як інші, такі як DEVS [26], які є більш гнучкими, вимагають від користувача знати про тонкощі розповсюдження моделювання.

1.2 Технології забезпечення паралельного та розподіленого моделювання

Фреймворк, що розробляється, повинен мати можливість виконувати моделювання та оптимізаційні дослідження у розподілених і децентралізованих середовищах. З цією метою ми вибрали розподілену архітектуру на основі контейнерів, яка базується на мікросервісах через її потужність, технічну та конфігураційну простоту [27]. У цьому розділі ми описуємо технічні технології, що задіяні в роботі, які виконують розподілене моделювання на основі мікросервісів, парадигми контейнеризації та стандарт DEVS.

1.2.1 Мікросервіси та парадигми контейнеризації

Традиційно системи розроблялися за монолітною архітектурою, де вся функція системи базується на одній програмі. Ця монолітна модель часто призводить до тісно пов’язаних систем із сильно взаємопов’язаними та взаємозалежними компонентами.

Навпаки, архітектури мікросервісів набирають популярності в останні кілька років. У цих архітектурах різні система, що моделюється, розкладається на окремі моделі, які спілкуються одна з іншою переважно

через асинхронні механізми (повідомлення), що керуються подіями. А стандартний протокол зв’язку та набір чітко визначених API, незалежний від будь-яких постачальників, продуктів або технологій, використовуються для зв’язку між мікросервісами.

Як зазначають Міттал і Мартін [28], будь-яка архітектура на основі мікросервісів має вирішувати дві фундаментальні проблеми: управління розподіленими даними (для зберігання стану мікросервісів локально) і спільна обробка подій (для полегшення обміну інформацією між мікросервісами). Ця інформація з локальних даних, а обробка подій зберігається всередині мікросервісів і використовується разом для виконання їх внутрішньої бізнес-логіки. Ця альтернативна методологія призводить до

а) розробки більш стійких систем, коли система продовжує свою роботу, навіть якщо окремі компоненти виходять з ладу,

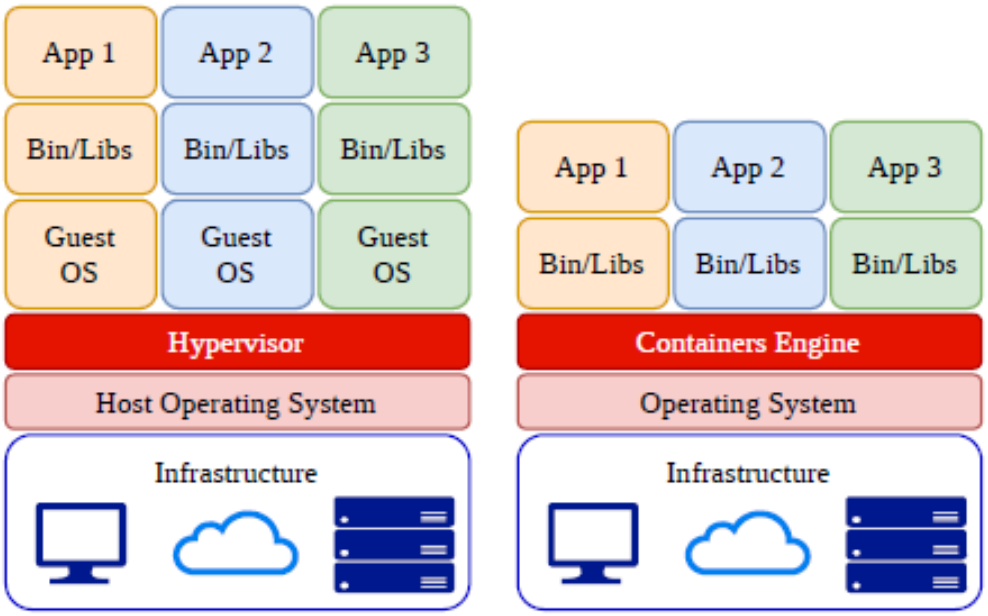
б) кращому використанню ресурсів, оскільки це дозволяє масштабувати конкретні компоненти на основі попиту,

в) чіткої незалежності компонентів системи, які можна розробити та перевірити окремо.

Для реалізації та розгортання систем на основі мікросервісів прийнято використовувати контейнерну архітектуру. Контейнер — це легкий, ефективний і стандартний спосіб переміщення програм між середовищами та незалежної роботи. Це загортає частину програмного забезпечення в повну файлову систему, яка містить усе необхідні для запуску (крім спільної операційної системи на сервері).

Цей підхід сприяє переносимості систем, оскільки вони можуть бути легко розгорнуті в багатьох операційних системах і апаратних архітектурах, і дозволяє прискорити управління циклами розробки, тестування та виробництва. Вони також становлять менше накладних витрат порівняно з традиційними середовищами віртуальних машин, оскільки вони не включають операції розгортання системи. Як наслідок, у багатьох випадках традиційна віртуалізація, присутня в перших рішеннях хмарних обчислень,

переходить до контейнерної архітектури. Рисунок 1.1 ілюструє відмінності між цими двома підходами, зокрема, рисунок 1.1 а) показує, як віртуальні машини зберігають всю операційну систему (ОС), бібліотеки, бінарні файли та програми, що вимагає величезного простору пам'яті в головній машині. На рисунку 1.1 b) показано, як контейнер складається з бібліотек, необхідних бінарних файлів та програми; і як усі контейнери мають спільний доступ до ядра ОС.



а) б)

Рисунок 1.1 – Графічне представлення а) віртуальної машини та б) архітектури контейнера

При управлінні великими контейнерними системами управління контейнерами приходить істотно. Воно відповідає за автоматизацію розгортання, управління, масштабування, мережу та доступність контейнерів. З часом з'явилися різні інструменти, які інкапсулюють їх і дозволяють застосовувати їх у різних двигунах контейнерів. Деякі популярні приклади з цих інструментів управління контейнерами є Kubernetes і Docker Swarm.

Крім того, багато хмарних служб пропонують платформи інфраструктури як послуги (IaaS) на цих інструментах, що дозволяє розробникам розгортати складні сценарії на основі контейнерів. Серед них Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS) і Google Kubernetes Engine (GKE).

1.2.2 Специфікації системи дискретних подій (DEVS)

DEVS — це загальний формалізм для моделювання систем дискретних подій на основі математичної теорії множин [9]. Ми можемо розрізняти Classic DEVS і паралельний DEVS. Паралельний DEVS був представлений як версія Classic DEVS. Рухаючись вперед, будь-яка згадка про DEVS означає Parallel DEVS. Поняття паралелізму у формалізмі DEVS існує як на рівні моделювання, так і на розподіленні. Це збіг, який відбувається одночасно в певний момент і як стандарти DEVS обробляють цей збіг подій у своїй стандартній моделі та, зрештою, реалізують її в середовищі моделювання, зберігаючи це злиття. Стандарт DEVS не стосується аспекту продуктивності паралельних обчислень для прискорення тощо. Виконання середовищ DEVS і симулятори компонентів у багатоядерній архітектурі є однією з тем дослідження в цій роботі.

Стандарт DEVS включає два типи моделей: атомарні та пов’язані моделі.

Обидві моделі мають інтерфейс, що складається з введення (X) і виходу (Y) для спілкуватися з іншими моделями. В атомарних моделях кожен стан (Q) у моделі є пов’язаним з функцією випередження часу, що визначає тривалість, протягом якої стан залишається незмінним. Одного разу призначений час відповідає внутрішній функції переходу ($\delta_{внутр}: Q \leftarrow Q$) і запускає внутрішній перехід, що призводить до локальної зміни стану ($\delta_{внутр}(s) = s'$). На у цей момент результати виконання моделі реалізуються через вихідні порти моделі шляхом активації функції виведення (λ). Вхідні

зовнішні події (події, що отримано від інших моделей) збираються у вхідні порти. Зовнішня функція розташування ($\delta_{доп}$:$Q \times X \leftarrow Q$) визначає, як реагувати на ці вхідні дані за допомогою поточного стану (S), час, що минув з останньої події (e) і вхідне значення (X) ($\delta_{внутр}((s, e), x) = s'$). Паралельний DEVS представляє конфлюентну функцію ($\delta_{внутр}((s, ta(s)), x) = s'$), яка визначає наступний стан у випадках зіткнення між зовнішніми та внутрішніми подіями.

З'єднана модель має чотири додаткових набори: дочерні компоненти C, зовнішній вхід EIC, зовнішній вихід EOC, і внутрішня інтерфейс IC.

Поєднані моделі представляють агрегацію/композицію двох або більше атомів і пов'язаних моделей, які з'єднані явними зв'язками, що робить DEVS замкнутими. Замкнутість у зв'язку дозволяє використовувати мережі систем як компоненти у більших об'єднаних системах, що призводить до ієрархічної модульної конструкції.

Загалом, цей формалізм забезпечує основу для інформаційного моделювання декілька переваг для аналізу та проектування складних систем: повнота, можливість перевірки, розширюваність та зручність обслуговування.

2 МОДЕЛІ ПАРАЛЕЛЬНОЇ ТА РОЗПОДІЛЕНОЇ АРХІТЕКТУРИ

Після опису системи відповідно до теорії DEVS її можна легко реалізувати за допомогою одного з багатьох механізмів DEVS M&S. Усі вони пропонують зручний для програміста API для визначення нових моделей за допомогою мови високого рівня, але лише деякі з них забезпечують зручний API для паралельного та розподіленого виконання симуляції, зокрема у хмарному середовищі.

Серед них xDEVS [10, 29, 28] нещодавно включив хорошу альтернативу розпаралелювати та розповсюджувати процес моделювання в хмарі, слідуючи мікросерверній архітектурі і контейнеризації, згадані в попередньому підрозділі. У цьому підрозділі подано короткий вступ до xDEVS, а потім і до паралельних та розподілених архітектур, які будуть використані у кваліфікаційній роботі.

2.1 xDEVS

xDEVS — це кросплатформенний симулятор дискретно-подійних систем, який забезпечує універсальний інтерфейс прикладного програмування DEVS (API) як на рівні моделі, так і на рівні моделювання. API реалізований у трьох широко використовуваних об’єктно-орієнтованих мовах програмування: C++, Java та Python. Репозиторій доступний через проект API за адресою [10], де проект має три основні гілки (з назвами c++, java та python). Ця структура дозволяє стандартизувати та виконати моделі DEVS. На основі формалізму DEVS, він має чітке розділення між моделюванням і імітацією. Діаграма класів, що показує взаємозв’язок між цими рівнями моделювання та симуляції, показано на рисунку 2.1.

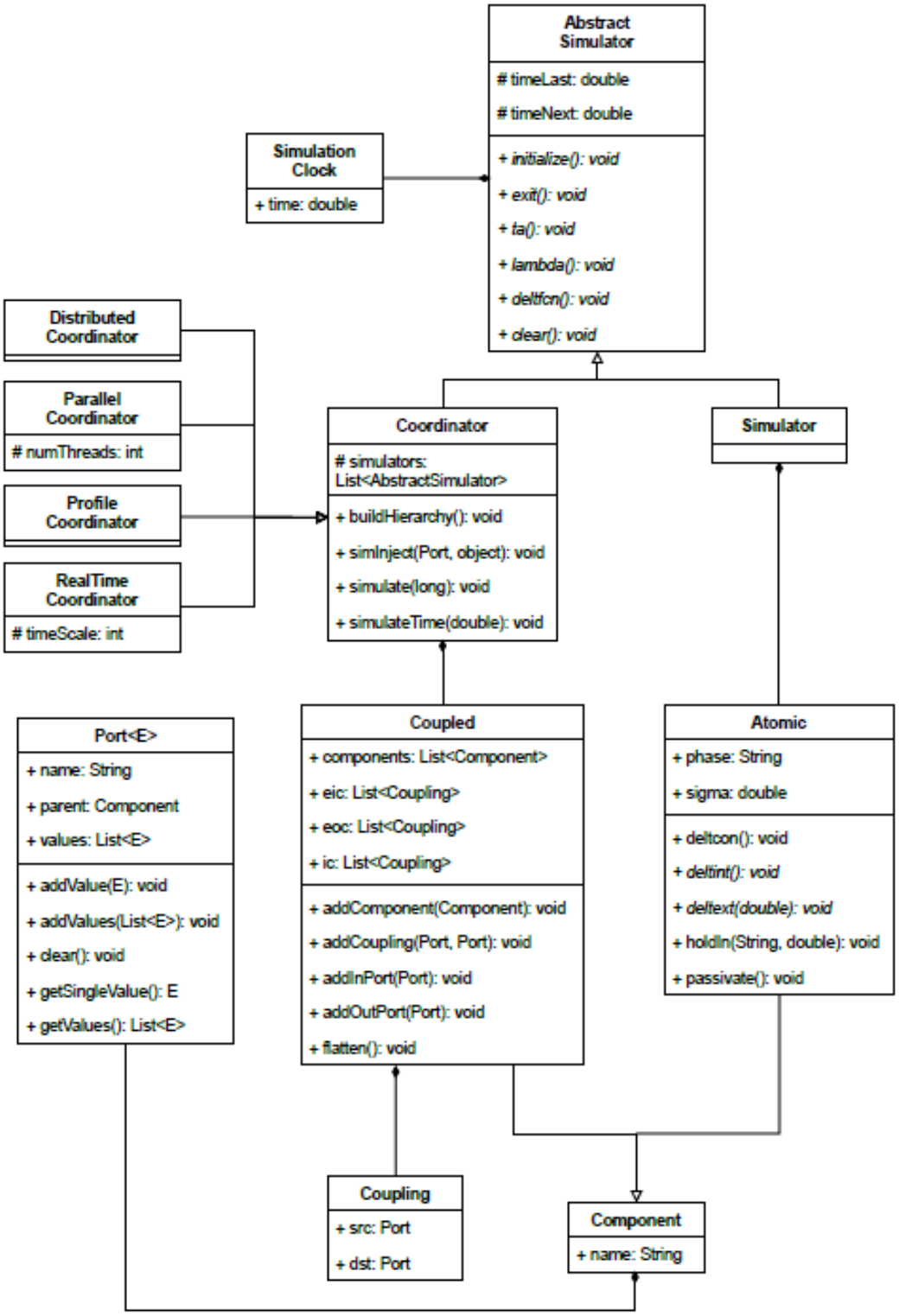


Рисунок 2.1 – Діаграма класів архітектури xDEVS

Моделі DEVS у xDEVS створюються за допомогою двох основних компонентів. Атомні компоненти визначають поведінку системи в парі компонентів, створюючи ієрархію моделі. Обидва вони мають порти, які представляють інформаційні точки входу/виходу. Зв'язати два компоненти

моделі можна, вибравши джерело і призначивши порти. Інформацію з інтерфейсів пари елементів, об’єднують, вказуючи порти, які потрібно з’єднати.

Рівень моделювання базується на концепції абстрактного симулятора. Дотримуючись цієї концепції, ми поділяємо сутності моделювання на Симулятори та Координатори. Кожен Симулятор пов'язаний з атомарним компонентом. Кожен Координатор приєднується до певної пов’язаної моделі та синхронізує їх дочірні Симулятори та Координатори. Це призводить до еквівалентної ієрархії описаний для шару моделювання.

Відповідно, Coordinator API займається виконанням пов’язаної моделі DEVS протягом певного часу. У цій роботі ми представляємо як паралельний, так і розподілені координатори, відповідно CoordinatorParallel та CoordinatorDistributed, які розроблені, щоб дозволити моделювати в централізованому або розподіленому паралельних середовищах.

2.2 Паралельна архітектура

Паралельний координатор xDEVS виконує послідовний координатор за допомогою кількох одночасних потоків і підходить для багатоядерних машин із спільною підсистемою пам'яті. Паралельний координатор xDEVS формується кількома потоками пул. Кожен дочірній координатор, як правило симулятор, додається до одного з пулів потоків.

Лістинг 2.1 показує уривок коду паралельного координатора з одним потоком в пуле. Як видно при побудові ієрархії, пара завдань, екземплярів TaskDeltFcn та TaskLambda, створюються для кожного симулятора: одне завдання для запуску функції переходу та інша для запуску функції виводу відповідно.

За замовчуванням коренева пов’язана модель зрівнюється в паралельному та розподіленому моделюванні. Зведена модель DEVS — це модель, яка зведена до однорівневої пов’язаної моделі, що містить лише

атомарні моделі в результаті алгоритму зведення, який зберігає взаємозв'язки. Як наслідок, кореневий координатор керує лише симуляторами (для атомарних компонентів), а не ієрархічними координаторами. Цю поведінку може змінити менеджер.

Лістинг 2.1 – Паралельний координатор xDEVS

```
public class CoordinatorParallel extends Coordinator {
     protected int numberOfThreads;
     protected LinkedList<TaskLambda> lambdaTasks = new
     LinkedList<>();
     protected LinkedList<TaskDeltFcn> deltfcnTasks = new
     LinkedList<>();
     protected ExecutorService executor;
     public CoordinatorParallel(SimulationClock clock, Coupled
model,int'numberOfThreads) {
          super(clock, model, true);
          this.numberOfThreads = numberOfThreads;
          executor =
     Executors.newFixedThreadPool(numberOfThreads);
     }
     public void buildHierarchy() {
     super.buildHierarchy();
     simulators.forEach((simulator) -> {
     lambdaTasks.add(new TaskLambda(simulator));
     }
     );
     simulators.forEach((simulator) -> {
     deltfcnTasks.add(new TaskDeltFcn(simulator));
     }
     );
     }
     public void lambda() {
     executor.invokeAll(lambdaTasks);
     propagateOutput();
     }
     public void deltfcn() {
     propagateInput();
     executor.invokeAll(deltfcnTasks);
     tL = clock.getTime();
     tN = tL + ta();
     }
}
```

Цикл моделювання DEVS в основному складається з виконання в усіх симуляторах зазначає наступне:

а) функція випередження часу;

б) вихідна функція;

в) функція переходу.

Функція випередження часу викликає кожен симулятор для наступної події часу, тому вона не розпаралелюється через низьку складність. Функції виведення та переходу, навпаки, може вимагати більше процесорного часу. Таким чином, ці два завдання цілком виконані паралельно у пулі потоків. Як показано в лістингу 2.1, як вихідна, так і транс-функція розташування запускають відповідні дочірні функції паралельно (через invokeAll виклик), з кількістю потоків, визначених користувачем (в атрибуті numberOfThreads).

Менеджер може додати більше пулів потоків, створивши новий паралельний координатор з двома або більше виконавців сервісів пулу потоків. Потім і лямбда-функцію, і функцію переходу необхідно змінити відповідно до цієї схеми N потоків (Лістинг 2.2).

Лістинг 2.2 – Схема для N пулів

```
// ...
public void lambda() {
      executor1.invokeAll(lamdaTasks1);
      executor2.invokeAll(lamdaTasks2);
      // ...
      executorN.invokeAll(lamdaTasksN);
}
```

Варто зазначити, що різні пули виконуються послідовно, хоча кожен із них внутрішньо працює паралельно. Проте, маючи великий пул (з багатьма потоками) для складних моделей і невеликий пул (з кількома потоками) для легших моделей можуть привести до незбалансованого навантаження.

Для цілей цієї дослідницької роботи ми створили простий і конкретний клас, який завантажує XML-файл, що визначає пул розподілу для кожного атомарної моделі. Таким чином, клас створює стільки різних пулів потоків, скільки визначено у файлі XML разом із кількістю потоків для кожного пулу.

Розпаралелювання повністю сумісно з DEVS, оскільки воно використовує алгоритм моделювання DEVS, визначений у [9]. Таким чином, ми можемо запевнити, що результати паралельного моделювання будуть еквівалентними, а насправді ідентичними до результатів, отриманих за допомогою послідовного моделювання. Власне те саме модель може бути змодельована за допомогою як послідовного Координатора, та і паралельного Координатора, не змінюючи жодного рядка у коді моделі.

2.3 Розподілена архітектура

Далі ми надаємо деталі дизайну та реалізації механізму розподіленого моделювання xDEVS. Його новизна та перевага полягає в спрощенні підходів, розроблених протягом останнього десятиліття, щоб полегшити розгортання розподіленого моделювання на основі DEVS, незалежно від неоднорідності хмарного рішення, яке використовується.

За основу покладено виконання розподіленого моделювання xDEVS на основі мікросервісів за допомогою класичної моделі Experimental Frame - Processor (EF-P) [28].

Ця модель, представлена на рисунку 2.2, містить два компоненти: пов’язану модель експериментального кадру (EF) і атомарну модель процесора (P). Як зазначалось вище, координатори та симулятори використовуються для визначення структури симуляції. Кожна модель (або атомарний компонент) пов’язана з компонентом симулятора. У випадку пов’язаної моделі вона пов’язана з координатором компонента. Для того, щоб імітувати це в хмарі, ієрархічна модель автоматично розміщується за допомогою xDEVS2, видаливши всі проміжні зв’язки моделі, щоб отримати однорівневу пов’язану модель, яка складається з 3 атомарних моделей.

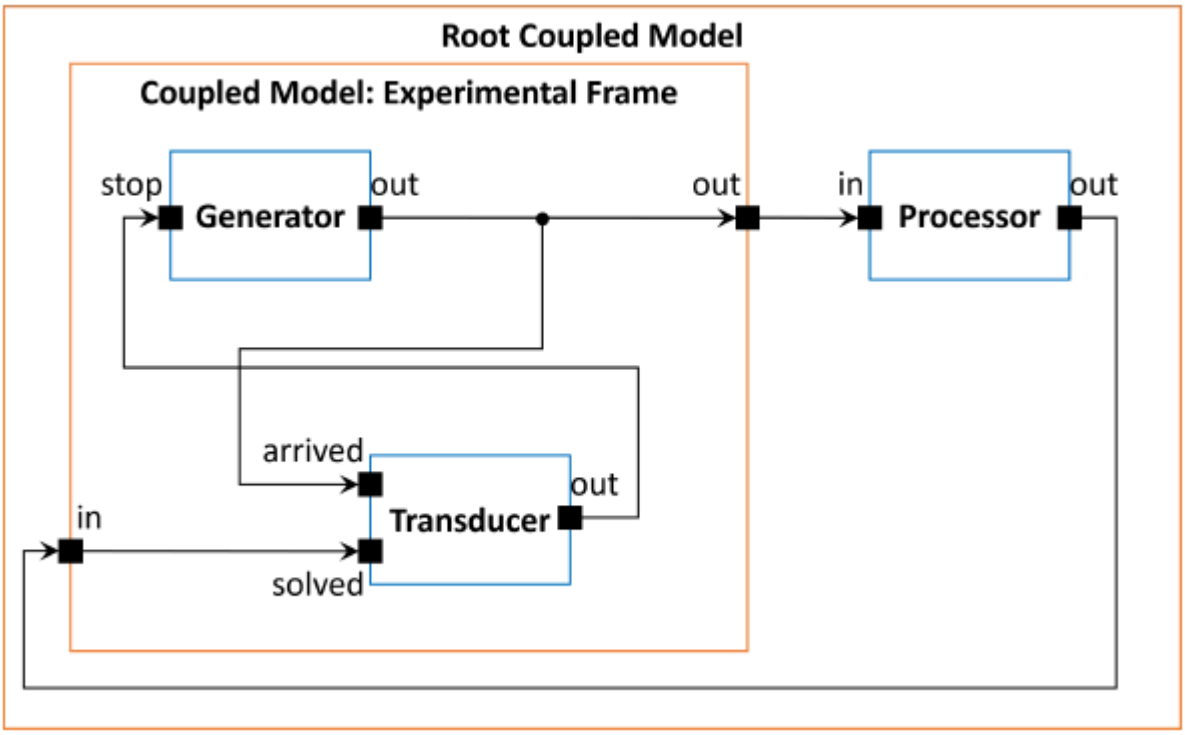


Рисунок 2.2 – Ієрархічна пов’язана модель DEVS

Еквівалентну суміщену пов’язану модель DEVS зображено на рисунку 2.3.

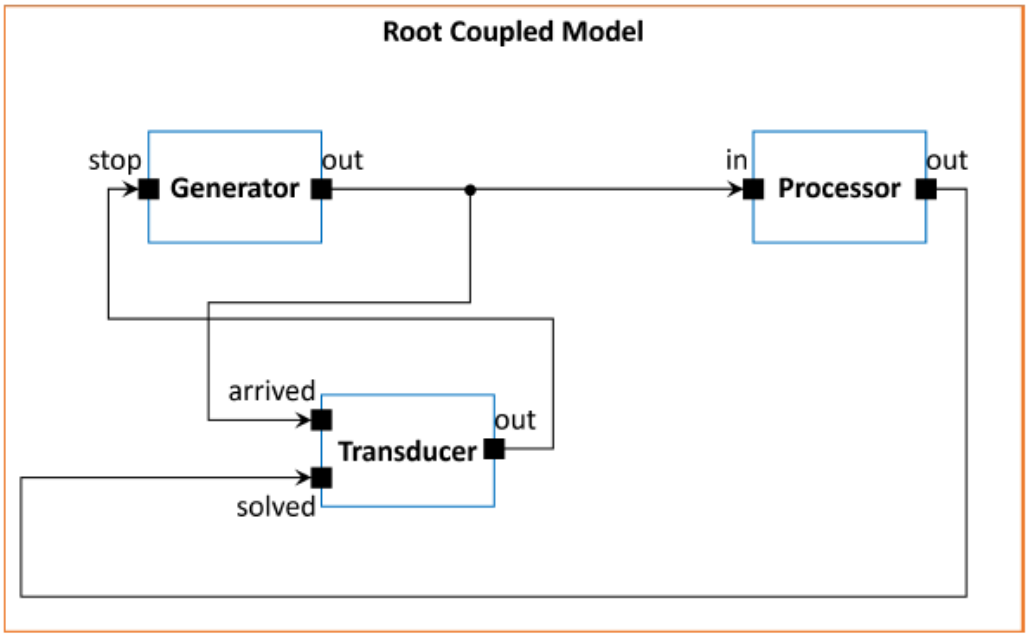


Рисунок 2.3 – Еквівалентна суміщена пов’язана модель DEVS

Використання файлу конфігурації дозволяє розподілене моделювання можна запустити, ввівши об’єкти моделювання що-небудь еквівалентне наступним викликам (Лістинг 2.3)

Лістинг 2.3 – Виконання розподіленої симуляції

```
$ # Simulation Entity 2, generator
$ java -cp xdevs.core.simulation.SimulatorDistributed gpt.xml
generator
$ # Simulation Entity 3, processor
$ java -cp xdevs.core.simulation.SimulatorDistributed gpt.xml
processor
$ # Simulation Entity 4, transducer
$ java -cp xdevs.core.simulation.SimulatorDistributed gpt.xml
transducer
$ # Simulation Entity 1, run the Coordinator
$ java -cp xdevs.core.simulation.CoordinatorDistributed gpt.xml
```

Незалежно від розгортання хмари, розподілене моделювання можна розглядати як набір незалежних процесів, пов’язаних між собою за допомогою виконання мікросервіса. У файлі конфігурації перелічено атомарні моделі, а також IP-адресу та порт для кожної з них, який модель прослуховує, з еквівалентною структурою до паралельного файлу конфігурації. Під сутністю ми розуміємо комп’ютер, віртуальну машину, контейнер тощо, будь-який віртуальний або фізичний пристрій, здатний імітувати модель xDEVS. Рисунок 2.4 ілюструє процес.

Після запуску координатора він викликає команду через сокети, які виконують функцію виведення як мікросервіс. Кожен компонент симулятора слухає цю команду та запускає функцію виведення (лямбда) їх відповідних атомних моделей. По-друге, координатор викликає команду для розповсюдження вихідних даних, надісланих і виконаних усіма симуляторами компонентів. Щоб уникнути подальших накладних витрат, пов’язаних із мережевим зв’язком, значення розповсюдження виконується безпосередньо між симуляторами компонентів без координатора, який діє як

реле між ними.

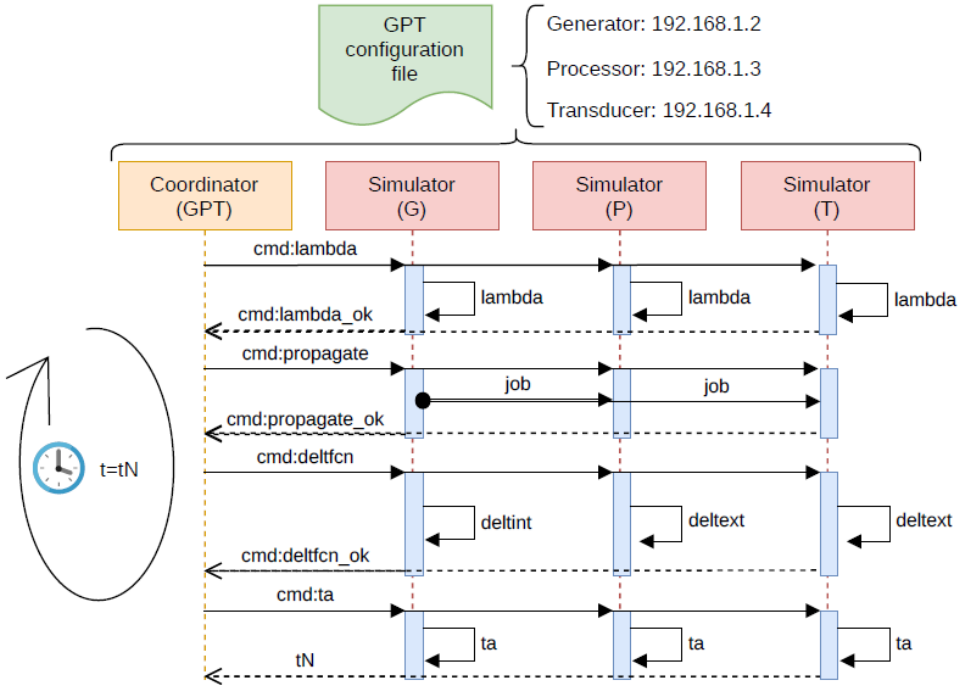


Рисунок 2.4 – Діаграма послідовності розподіленого моделювання xDEVS на основі протоколу абстрактного моделювання DEVS

Після вихідного розповсюдження, запитується виконання функції переходу, і кожен симулятор компонента виконує функцію передачі (внутрішню, зовнішню або перевизначену) залежно від поточного часу моделювання, стану та зовнішнього оцінювання, якщо функція управління була зовнішня, та передає вхідне повідомлення на вхідні порти.

Нарешті наступна подія (tN на рисунку 2.4) вимагає знову запустити цикл моделювання DEVS. Це виконується до того, коли досягнуто заданої кількості ітерацій DEVS, або всі моделі входять у пасивний стан.

Як видно, алгоритм розподіленого моделювання базується на фундаментальному протоколі абстрактного моделювання DEVS, представленому в [9]. Модель завжди однакова в послідовному, паралельному та розподіленому виконанні, а отже, поточна архітектура xDEVS об’єднує паралельне моделювання та моделювання розподілу стандарта Parallel DEVS у реалізації xDEVS.

2.4 Архітектура програмного забезпечення

Архітектура програмного забезпечення (рисунок 2.5) базується на традиційної розподіленої архітектурі, в котрої кожний з компонентів клієнт/сервер може прослуховувати, приймати та відповідати на повідомлення незалежно та паралельно. Симулятор і координатор реалізовані за допомогою класів CoordinatorDistributed і SimulatorDistributed відповідно. Кдас повідомлення Message - призначений для обробки команд, що надсилаються між координатором і симуляторами (див. рисунок 2.4). Інформація передається через порти (за допомогою сокетів). Нарешті, клас DistributedTask був розроблений для виконання всіх завдань координатора паралельно.

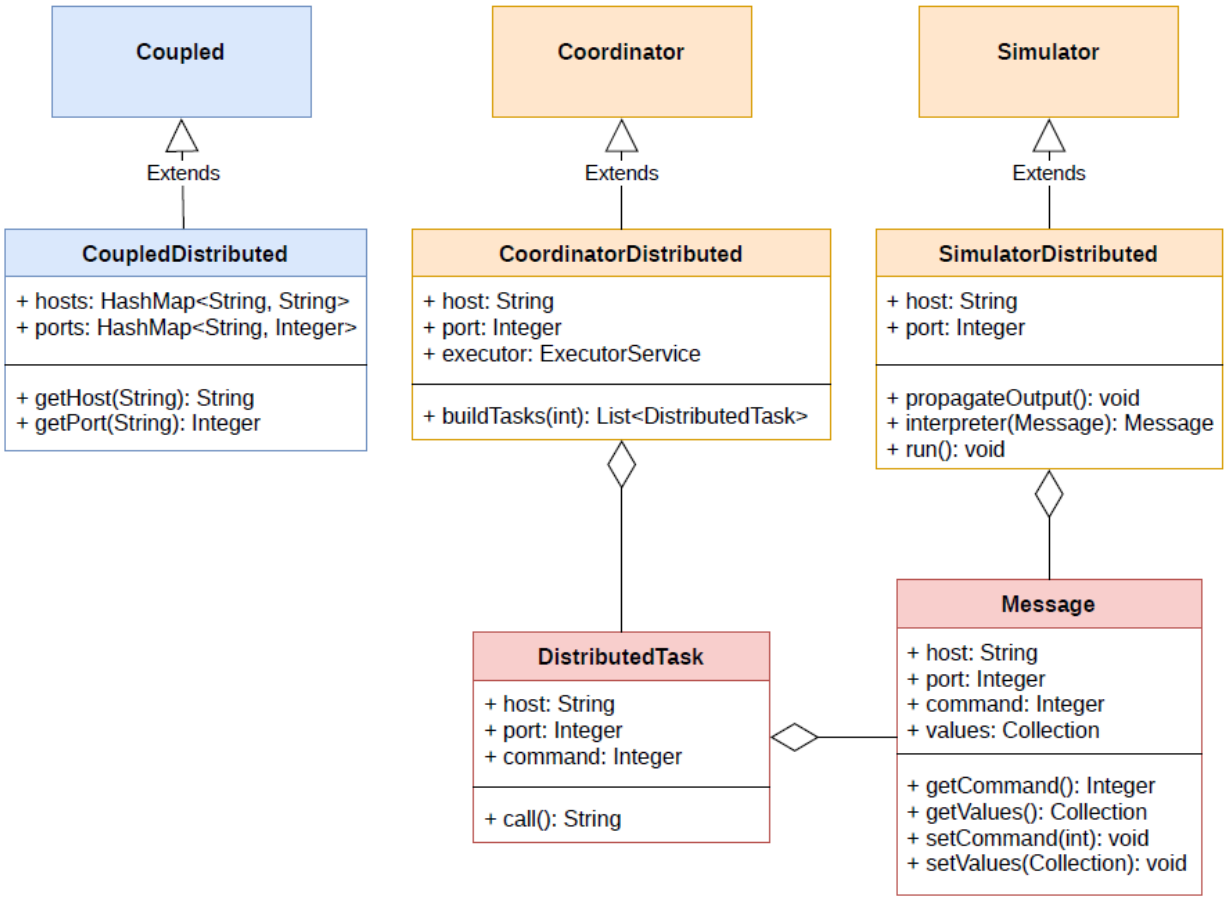


Рисунок 2.5 – Діаграма класів для підтримки розподіленого моделювання в xDEVS

Механізм розподіленого моделювання не потребує додаткових бібліотек або середовищ виконання (фрейворків) та його розгортання можна легко автоматизувати, як описано в наступному розділі.

3 РОЗГОРТАННЯ ТА ДОСЛІДЖЕННЯ

3.1 Розгортання паралельних та розподілених моделей

Як паралельне, так і розподілене моделювання можна розгорнути в будь-якій комп’ютерній системі зі спільною пам'яттю. У випадку розподіленого моделювання, кожна атомарна модель виконується всередині симулятора відповідного компонента як ізольований процес, тоді як процес-координатор позначає початок і кінець моделювання, як описано на рисунку 2.4. Зв'язок між цими симуляторами виконується за допомогою мережевих вузлів. В результаті будь-яка розподілена архітектура можлива. На рисунку 3.1 показано кілька прикладів.

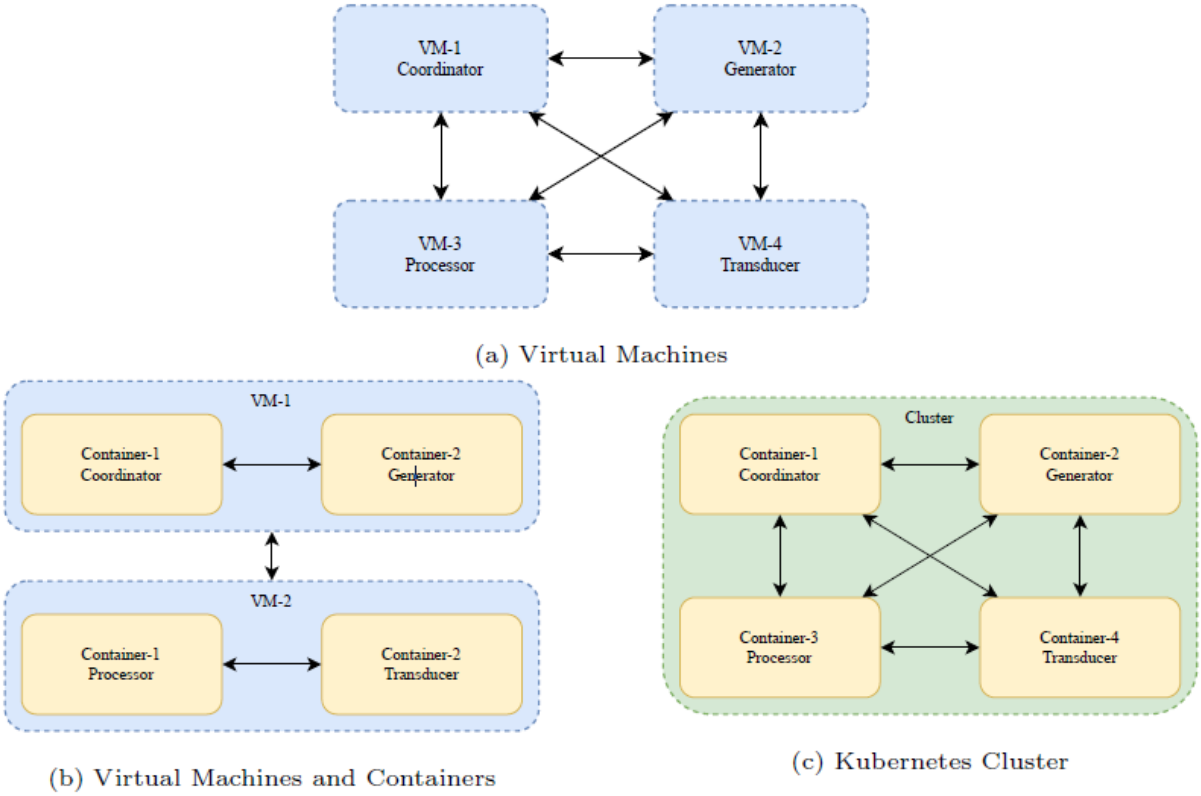


Рисунок 3.1 – Можливі архітектури для розгортання розподіленого моделювання

Наприклад, рисунок 3.1(a) ілюструє просте розгортання GPT з використанням лише віртуальних машин, більш традиційного підходу. Рисунок 3.1(b), з іншого боку, ілюструє той самий розподіл, але з контейнерами всередині віртуальних машин. Нарешті, прикладі на рисунку 3.1(c) показано підхід, що використовується в цій роботі, де знаходиться набір контейнерів, який кере керується кластером kubernetes. Ці архітектури можливо та реально розгорнути в хмарі за допомогою послуг, що надаються постачальниками інфраструктури добре відомі на сьогоднішній день: серед інших Google, Amazon і Microsoft Azure, чиї послуги схожі або принаймні використовують стандартні інструменти віртуалізації як Docker і Kubernetes. Для цілей цього дослідження ми вибрали служби Google Cloud Platform, зокрема, які ми використовували для паралельного моделювання однієї віртуальної машини, а для розподіленого моделювання кластер контейнерів, автоматично розгорнутих через Google Kubernetes Engine (GKE).

На рисунку 3.2 показано кроки, які необхідно виконати, щоб виконати паралельний або розподілене моделювання. Цей процес є похідним від наших попередніх механізмів розгортання DEVS/SOA [24].

На першому етапі XML-опис зведеної версії оригіналу. Модель створена за допомогою xDEVS. Цей текстовий файл містить усі атомарні моделі і відносини зв’язку, отримані після зміни зв’язків пов’язаних моделей, які видаляються за замовчуванням для полегшення розгортання [23] і повторного зменшення накладних витрат на моделювання. Цей текстовий файл, на додаток до традиційних DEVS атрибутів (імена компонентів, імена портів, підключення тощо), також містить адреса хоста, яка ідентифікує об’єкт симуляції (який буде розгорнутий у середовищі виконання), і кінцеву точку зв’язку (іменований порт) для випадку розгортання розподіленого моделювання та ім’я пулу потоків для розгортання паралельного моделювання. Оскільки відбувається генерація текстового файлу автоматизований, він генерує одне ім’я хоста та кінцеву точку або один пул потоків.

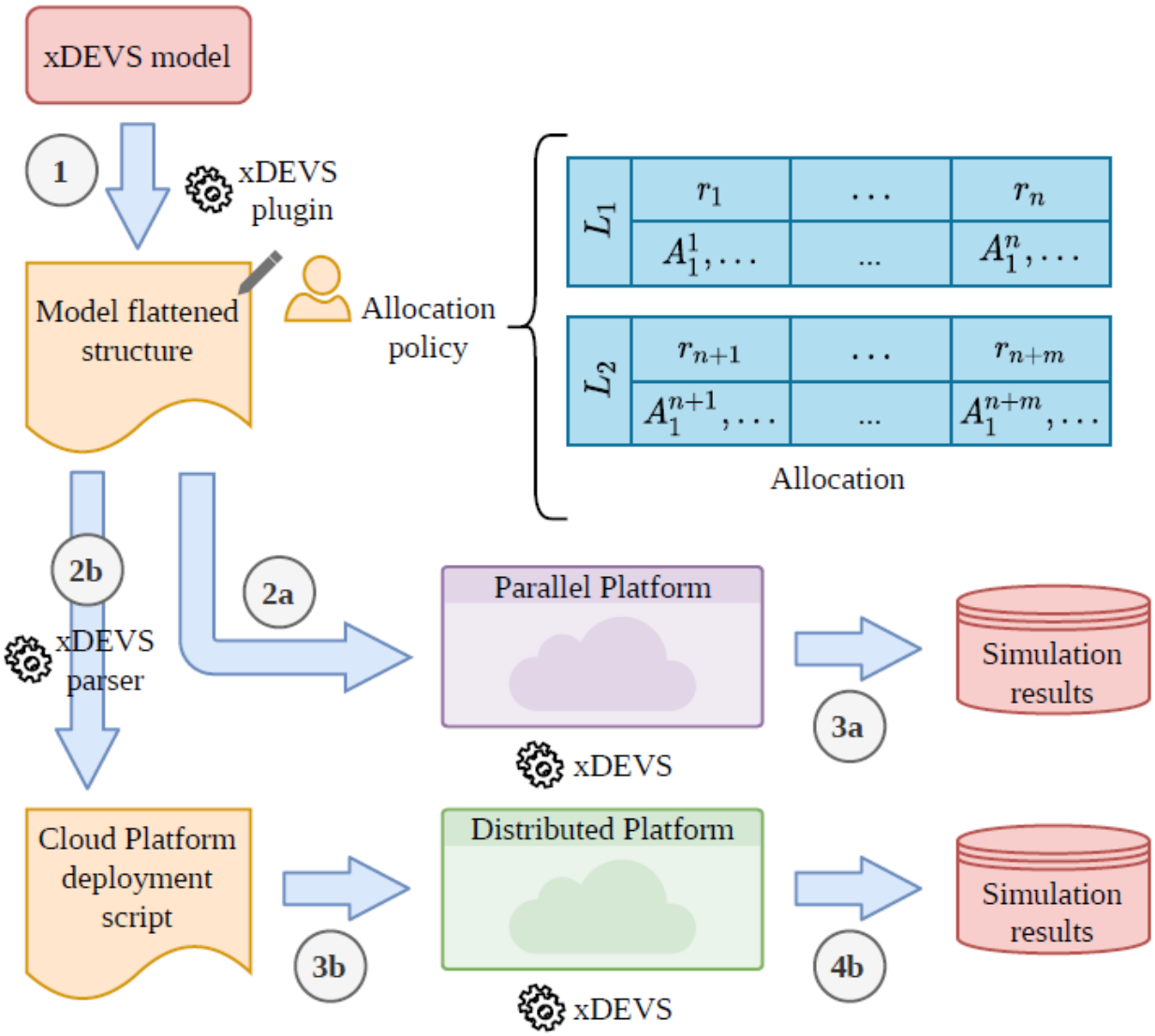


Рисунок 3.2 – Схема розгортання хмари

Однак цей файл можна редагувати, щоб змінити типову поведінку. Хоча всі атомарні моделі можна віднести до одного контейнера або до пулу потоків, цей параметр ще не працює, оскільки модель DEVS може містити сотні атомних моделей із величезною різноманітністю обчислювальної ваги в термінах циклів ЦП. Таким чином, редагування початкового текстового файлу, як показано на рисунку 3.2, дозволяє згрупувати кілька атомарних моделей на один набір контейнерів (розподілене розгортання) або пул потоків (паралельне розгортання). На рисунку 3.2 показано 2-рівневу політику розподілу, яку ми використовували. Цей розподіл розподіляє атомарні моделі ($A_i^j \in A$) над двома наборами контейнерів або двома пулами

потоків. Моделі з високими обчислювальними вимогами розміщуються на рівні 1 (*L1*), від r_1 до r_n. Решта атомних моделей розподіляються по рівню 2 (*L2*), від від r_{n+1} до r_{n+m}. Тут r_i являє собою обчислювальний ресурс. r_i є контейнером у випадку розподіленого моделювання за схемою, наведеною на рисунку 3.1(c), або одним потоком у випадку паралельного моделювання. У будь-якому випадку можна виділити одну або кілька атомарних моделей в кожному ресурсі. Загалом, $n > m$ згідно крупнозернистого розподілу. Така політика розподілу:

- використовує знання управляючої програми про те, які атомні моделі споживають більше ресурсів процесора;

- уникає фази профілювання, яка потребує обчислень.

Другий етап, після завершення політики розподілу, залежить від типу моделювання. У паралельному випадку модель просто імітується за допомогою класу CoordinatorParallel (крок 2а на рисунку 3.2), створюючи вказані пули потоків, і отримані результати моделювання. У розподіленому випадку синтаксичний аналізатор читає XML-файл і генерує специфічний для архітектури сценарій у вигляді файлу розгортання YAML (крок 2б на рисунку 3.2). Цей файл описує структуру розгортання розподіленого моделювання, включаючи конфігурації модулів, їхні внутрішні контейнери та порти, відкриті в цих контейнерах для зв’язку різних моделей через мережу. У цьому випадку ми вказуємо один контейнер. Дизайн цього аналізатора простий, складається з читання XML-файлу та створення YAML-файлу, і його можна адаптувати до інших постачальників послуг.

На третій фазі модулі, зазначені у файлі YAML, створюються у вибраній хмарній платформі, і модель розгортається, як описано в політиках розподілу. На цьому кроці атомарні моделі розподіляються по контейнерам, створюючи відповідні процеси моделювання відповідно до протоколу моделювання DEVS. Таким чином, кожен контейнер виконує один або більше розподілених симуляторів xDEVS, кожен зі своєю відповідною

моделлю атома. Крім того, один конкретний контейнер запускає розподілений кореневий координатор xDEVS.

Як правило, після завершення моделювання, результати зберігаються розподіленим способом, оскільки кожна атомна модель може мати інший механізм для збереження своїх даних. Рекомендація для уніфікації цих даних полягає в тому, щоб мати різні атомарні перетворювачі моделі [9], збираючи відповідну інформацію та зберігаючи її у відповідних сховища.

Основна відмінність між паралельним і розподіленим моделюванням полягає в тому, що у випадку паралельного моделювання два або більше пулів потоків виконуються послідовно послідовно, один за одним, хоча кожен пул потоків, звичайно, є паралельним.

У разі розподіленого моделювання кожен симулятор є незалежним повноцінним процесом, який потребує багато виділеної пам’яті, але моделювання внутрішньо паралельно, незалежно від кількості використаних контейнерів.

3.2 Оцінювання варіантів розподілу моделей

У цьому розділі ми оцінюємо як паралельні, так і розподілені координатори механізму моделювання xDEVS. Це виконується за допомогою тесту DEVStone. DEVS контрольний показник використовується із затримкою в перехідних функціях для вимірювання продуктивності механізмів моделювання дискретних подій. Враховується аспект затримки, який необхідний для оцінки впливу виконання моделі на навантаження ЦП.

Спочатку ми опишемо тест DEVStone і те, як вводиться затримка. Далі ми виконали аналіз розподілу синтетичної затримки, вибравши модель DEVStone для призначення різних ваг затримки набору атомарних моделей. Одного разу призначаються затримки, після чого ми переходимо до аналізу

паралельного та розподіленого моделювання та надаємо результати порівняння.

3.2.1 Тест DEVStone

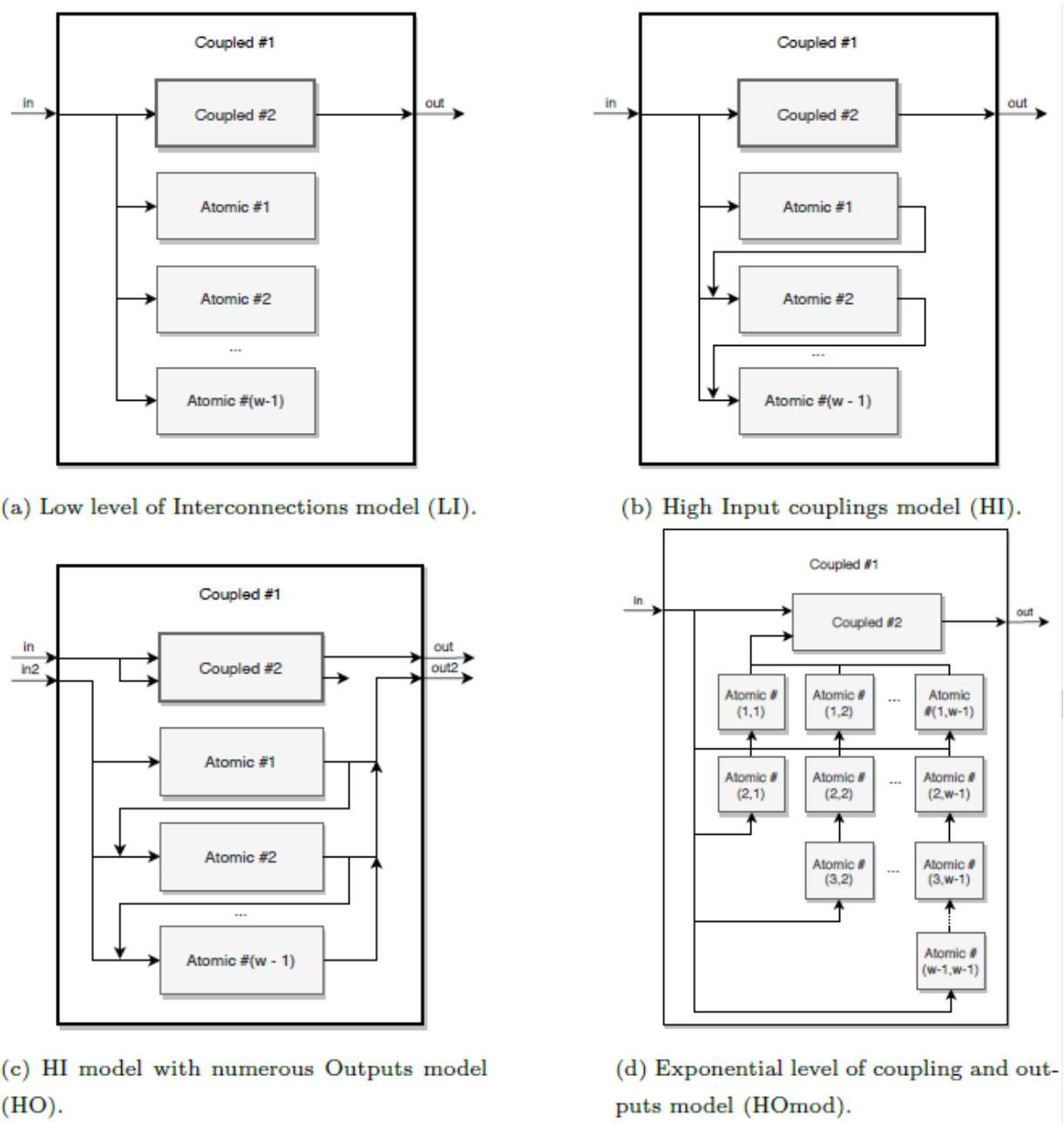
DEVStone [30] — це синтетичний тест, призначений для автоматизації оцінювання використання підходів моделювання на основі DEVS. Це дозволяє створювати різні типи моделей, кожна з яких спеціалізується на вимірюванні конкретних аспектів моделювання. Цей еталонний показник став популярним протягом багатьох років широко використовується в літературі для оцінки та порівняння ефективності різних різні симулятори DEVS [29, 31].

DEVStone описує кілька синтетичних моделей, в яких можна змінювати їх розмір і складність. З цією метою створено рекурсивну структуру з можливістю налаштування глибини, де всі рівні містять еквівалентні компоненти та взаємозв’язки. Налаштування моделей здійснюється за допомогою чотирьох параметрів: (i) ширина, що впливає на кількість компонентів (1 зв'язаний і ширина − 1 атомна модель) на шар, (ii) глибина, що визначає число вкладених поєднаних моделей, (iii)внутрішня затримка переходуі (iv)зовнішня затримка на транспортування та розташування. Відповідно до специфікацій DEVStone, ці два часи затримки витрачаються на виконання Dhrystones [32], щоб підтримувати роботу ЦП. В цій роботі ми обчислюємо цю затримку як час процесора, тобто Dhrys-цикл контрольного тесту виконує ітерації доки споживається час ЦП (а не час годинника), і він менш ніж Δ_{int} функції внутрішнього переходу, або Δ_{ext} функції зовнішнього переходу. Важливо вимірювати час ЦП, тому що інакше ЦП може запускати сотні одночасних функцій переходів, що споживають відповідні Δ_{int} і Δ_{ext} затримки, і не змушує зберігати кожну функцію переходу в ЦП протягом зазначеного часу.

Поведінка моделі DEVStone визначається її розподілом атомних моделей DEVStone (рисунок 3.3):

- LI (Низький рівень взаємозв'язків) моделі - найпростіші моделі, с низький рівень зв’язків (a);

- HI (High Input couplings) моделі подібні до моделей LI, але збільшується кількість внутрішніх зв'язків (8б);



(a) Low level of Interconnections model (LI).

(b) High Input couplings model (HI).

(c) HI model with numerous Outputs model (HO).

(d) Exponential level of coupling and outputs model (HOmod).

Рисунок 3.2 – Внутрішня структура DEVStone-моделі

- HO (моделі з численними виходами) моделі є різновидом моделі HI, у яких усі атомарні компоненти в кожному з'єднаному модулі під'єднані до з'єднаного вихідного порту. Варто відзначити, що ці моделі мають непідключені порти, які можуть служити для виявлення несправності в імітаторах при очищенні значень портів без зв'язків (в);

- HOmod моделі відтворюють експоненціальний рівень зв'язку та виходів (d).

Аналіз публікацій, які вивчають продуктивність моделювання DEVS движків через DEVStone, ми можемо говорити про те, що набір HO пропонує хороший баланс між використанням процесора та пам'яті [29, 33]. У результаті ми використовуємо набір HO моделей DEVStone, щоб оцінити продуктивність наших паралельних і DEVS механізми розподіленого моделювання. У HO моделі найглибшого зв'язку формується за допомогою однієї атомній моделі.

3.2.2 Проведення експериментів

Розмір моделі HO та затримки переходу повинні бути враховані, щоб досягти компромісу між кількістю атомних моделей і часом моделювання.

Щоб знайти ці значення, ми виконали профілювання різних моделей HO. У цьому розділі показано отримані результати та обґрунтовано вибір однієї моделі HO. Усе профілювання було виконано на віртуальній машині з 4 процесорами Intel(R) Xeon(R) на 2,8 ГГц і 32 ГіБ оперативної пам'яті на основі серії конфігурацій N2 Google Cloud і операційної системи Debian GNU/Linux з OpenJDK 11. Це мінімальний вузол, який може працювати для розподіленого моделювання, тому він фіксується як базова машина для послідовного, паралельного і розподіленого експериментів.

Спочатку розглянимо моделі, в яких ширина дорівнює глибині ($w=d$). Ми протестували шість різних розмірів w=10,11, . . . ,15, що надає 82, 101,

122, 145, 170 і 197 атомних моделей відповідно наступне (1). Далі ми визначили зовнішню затримку переходу, що дорівнює внутрішній затримці переходу в кожній атомній моделі.

На рисунку 3.3 зображено час моделювання шести моделей різних розмірів для семи різних конфігурацій затримок переходу. Щоб дозволити повторення конфігурації параметрів у паралельному та розподіленому експериментах, ми вибрали випадкове початкове значення. Незалежно від розміру моделі, найповільніше моделювання відповідало Δ_i=3, а потім Δ_i=χ2 або Δ_i=2 і Δ_i=N(0,3), Δ_i=N(0,2), Δ_i=1, і Δ_i=N(0,1).

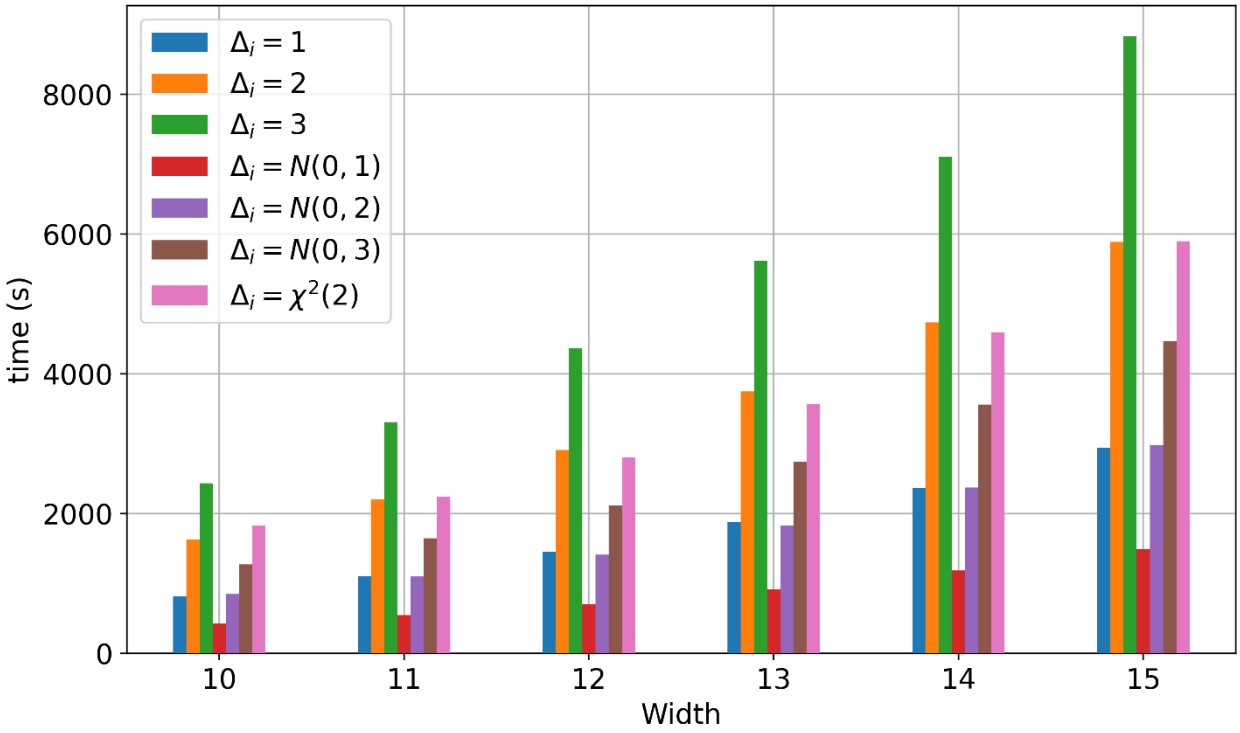
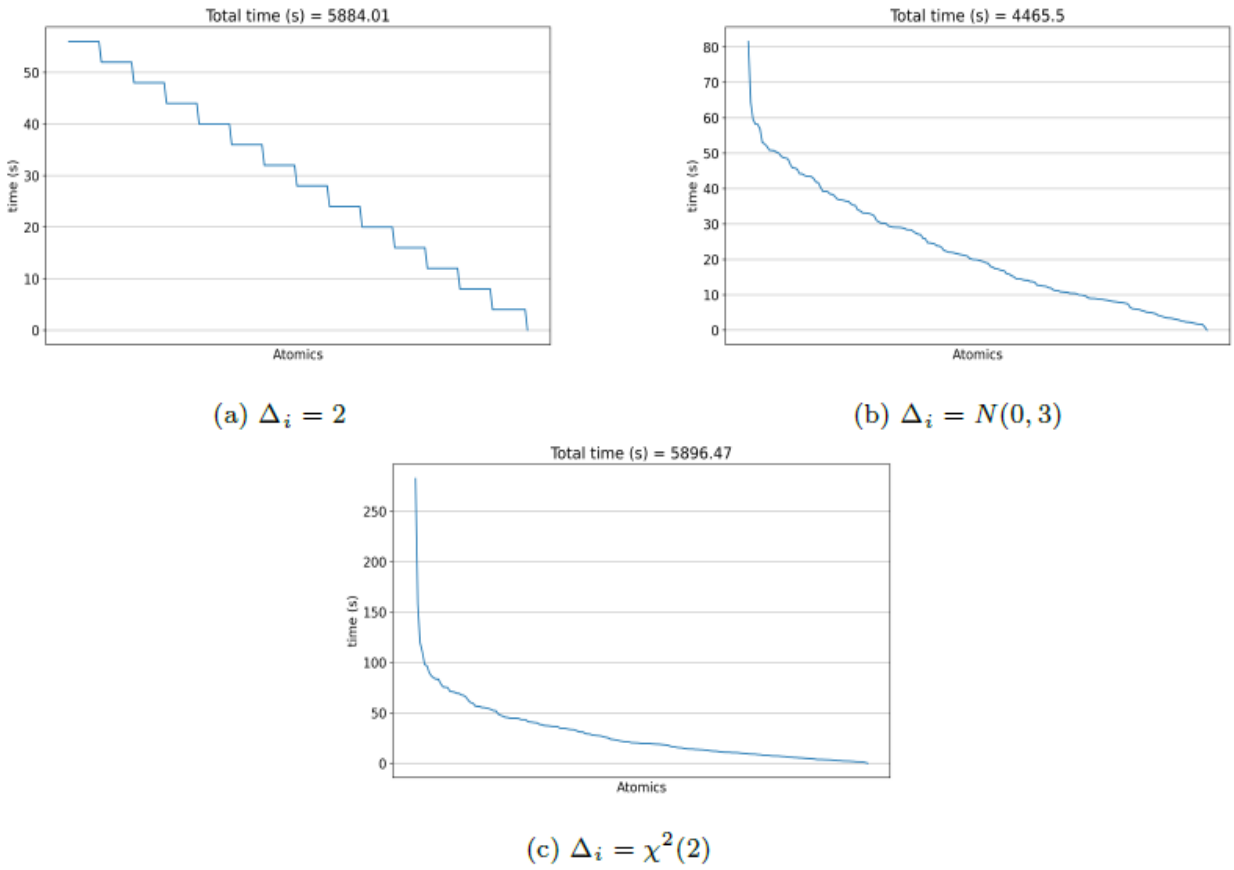


Рисунок 3.3 – Час моделювання

Моделі HO з Δ_i=3 і w=15 було змодельоване в 8826.06 секунд.

З іншого боку, моделі HO з Δ_i=H(0,1) iw=10 було змодельоване в 424.58 секунд. В результаті виходять моделі HO з шириною рівною 15, що відповідає розміру моделі (197 атомних моделей) для виконання різних паралельних і розподілених моделей зі значною кількістю пулів потоків і контейнерів.

Щоб вибрати один із семи розподілів для затримок, ми перевірили час моделювання, який споживає кожна атомна модель. Ми перевірили Δ_i=2, Δ_i=H(0,3) і Δ_i=χ2(2), оскільки вони пропонують (а) Δ_i=2, (б) Δ_i=H(0,3) та (с) Δ_i=χ2(2) (рисунок 3.4) з еквівалентний часом моделювання (діапазон 4000-6000 секунд) і представляють три класи розподілу (постійні, рівномірні та ксі-квадрат).



(a) $\Delta_i = 2$ (b) $\Delta_i = N(0,3)$

(c) $\Delta_i = \chi^2(2)$

Рисунок 3.4 – Час моделювання, який споживають атомні моделі з використанням трьох різних розподілів і постійним розміру (w=15)

Фігура 3.4 ілюструє результати. Для простоти ми не позначили атомні моделі, але впорядкували їх від вищого до нижчого часу моделювання.

Дотримуючись схеми нумерації, атомний компонент позначено як A_i^j, де *j*-а пов'язана модель, яка належить (з 1-ю кореневою моделлю HO та

пов’язаною моделлю i та d=w=15), $i \in 1...w-1$ = 14 номер атомного компонента в j-му ланцюгу моделей пов’язаних компонентів.

Сталий розподіл (Δi=2) дає добре відому поведінку. Оскільки кожна атомна модель A_i отримує події в каскаді, всього i зовнішній і внутрішній, виконуються функції переходів (2×i переходи). Таким чином, час моделювання споживається функціями переходів і становить приблизно $2 \times i \times \Delta i = 4 \times i$.

Тоді є 14 атомних моделей займає 56 секунд, 13 моделей атомів займає 52 секунди тощо.

Рівномірний розподіл H(0,3) дає мінімальне значення 0, максимальне значення 3 і середнє значення 1.5, які рівномірно розподілені. Таким чином, ми можемо очікувати максимальне споживання 2×14×3 = 84 секунди, близько 81.58 секунд, а потім плавне лінійне падіння, еквівалентне спостережуваному в постійному розподілі. Данні свідчать про ефективність запропонованих рішень.

Розподіл ксі-квадрат дає мінімальне значення 0, статистично максимальне значення 10.60, а середнє значення 2. Є кілька моделей з високим значенням Δ_i, оскільки розподіл дещо незбалансований. Ми можемо очікувати максимальне споживання 2×14×10.60 = 296.80 секунд, знову близько до $t(A_{614})$ = 282.20 секунд, а потім короткий різкий спад, за яким слідує плавний спуск.

Оскільки розподіл хі-квадрат демонструє більшу варіативність, він досить добре охоплює спектр симуляцій, які ми хотіли б проаналізувати. Маючи таку різноманітність часу моделювання, ми можемо краще проаналізувати вплив на кількість потоків або контейнери, розгорнуті для розподіленого моделювання, а також кількість виділені в них розподілені атомні моделі. Це підкреслює гнучкість запропонованих рішень і дозволяєлегко масштабувати рішення, тому їх впровадження доцільно у системах хмарних обчислень.

3.3.3 Паралельне моделювання

Далі ми показуємо результати, отримані в результаті паралельного моделювання.

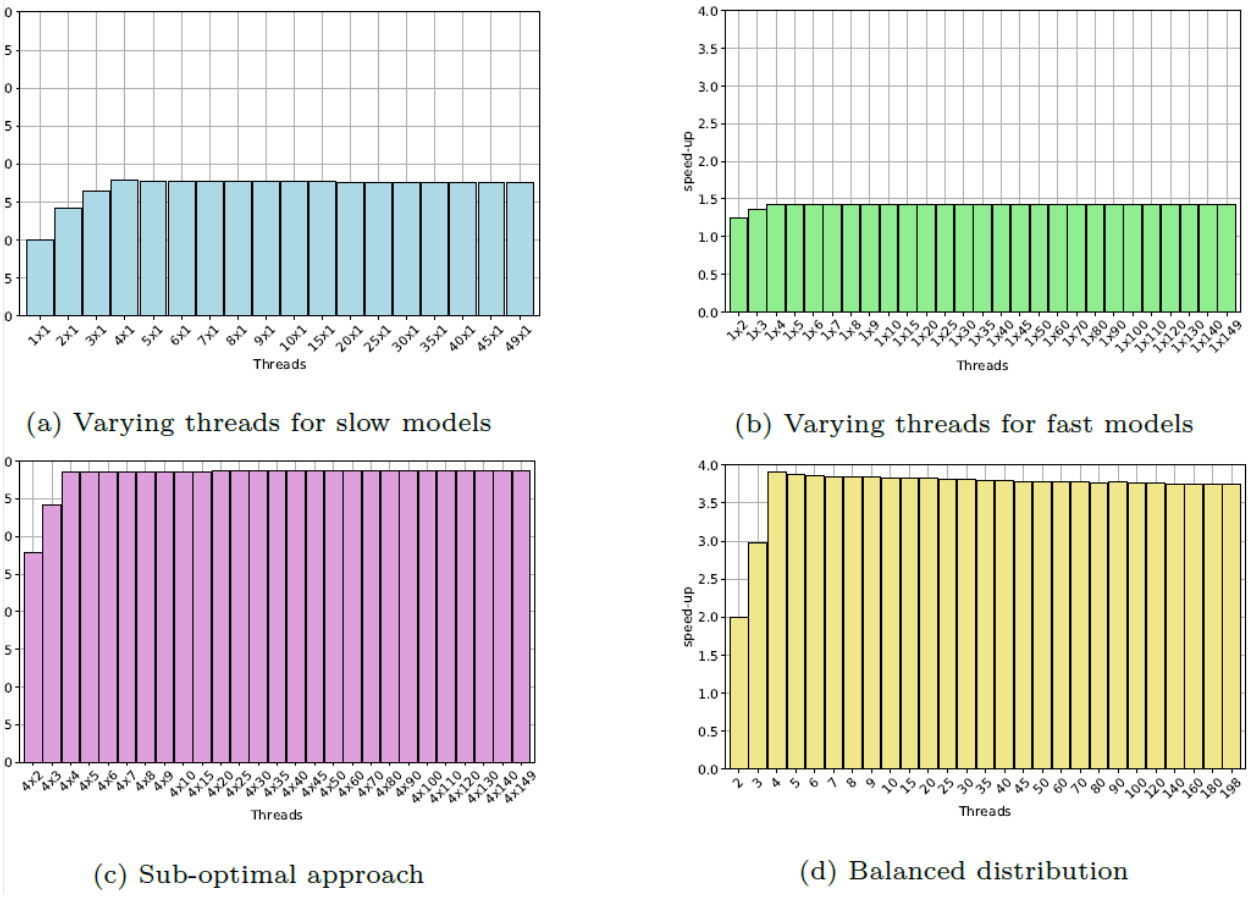
З цією метою ми налаштували кілька експериментів після розгортання показано на рисунку 3.2 з двома пулами потоків і додатково паралельним виконанням з одним пулом потоків. Управління апаратними ресурсами для кожного пул потоків було залишено операційній системі.

Ми використовували розподілене моделювання як еталон для встановлення віртуальних машин. Як наслідок, ми перевірили паралельне моделювання з використанням 4 процесорів Intel(R) Xeon(R) на 2,8 ГГц і 32 ГБ оперативної пам’яті, що є мінімальним вузлом, здатним підтримувати розподілене моделювання, і 32 процесора Intel(R) Xeon(R) @ 2,8 ГГц і 256 ГБ оперативної пам'яті, оскільки ми підключили до восьми вузлів у розподіленому моделюванні з операційною системою Debian GNU/Linux 10, і з OpenJDK 11.

У першій серії експериментів ми використовували два пули потоків. Перший пул було визначено для запуску 25% найповільніших атомних моделей (загалом 49), тоді як інший пул використовувався для розподілу решти (всього 149, в т.ч генератор початкової події). Ідея полягає в тому, щоб довести, що призначення ресурсу для найповільніших моделей (більше потоків, тобто $n > m$ на рисунку 3.2), у цьому випадку отримується покращення продуктивності.

У другому наборі в експериментах ми використовували один пул потоків, де обчислювальне навантаження кожного потоку був збалансований розподілом найважчих моделей у різних потоках. Щоб обчислити прискорення, ми використовували як еталонний час виконання послідовного моделювання, тобто 5896.54 секунди. Для аналізу ми змінювали кількість потоків у кожному пулі наслідки розподілу ресурсів.

З одного боку, рисунок 3.5 ілюструє результати, отримані для 4 vCPU віртуальних машин. Штрихові ярлики мають форму $i \times j$, де i представляє кількість потоків у пулі високого пріоритету (L_1 як на рисунку 3.2), і j представляє кількість потоків у пулі з низьким пріоритетом (L_2 як на рисунку 3.2). рисунку 3.5(a) показано прискорення, коли кількість потоків, якими керує L_1 пул, збільшився. Як видно, максимальне прискорення (1.78×) виходить, коли L_1 пул використовує кількість потоків, що дорівнює кількості ЦП. Рисунок 3.5(b) показує той самий ефект, але змінюється кількість потоків, якими керує L_2 пул. Однак максимальна продуктивність в цьому випадку (1.43×) досягнуто, коли кількість потоків у L_2 пулі дорівнює кількості швидких атомних моделей (149). Це тому, що ці моделі мають низьку обчислювальну потужність вагу, а потім 4 ЦП достатньо для обробки затримок переходу.



(a) Varying threads for slow models

(b) Varying threads for fast models

(c) Sub-optimal approach

(d) Balanced distribution

Рисунок 3.5 – Розподіл ресурсів (потоків) і прискорення для паралельного моделювання на 4 vCPU.

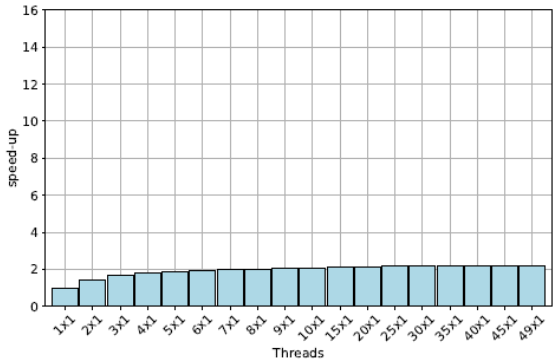
Порівнюючи рисунки (a) і (б), можна помітити, що прискорення досягається, коли більше ресурсів (потоків) надається повільнішим моделям, на 24,48% більше. Ці дві цифри підтверджують нашу $n > m$ гіпотезу, оскільки більше потоків для пулу L_1 покращує прискорення. Після цього ми маємо шукати неоптимальну конфігурацію, фіксуючи кількість оптимальних потоків з пулу L_1, і змінюючи кількість потоків у пулі L_2. Як показано на рисунку 3.5(c), прискорення значно вище (3.88×порівняно з попереднім 1.78×). Нарешті, ми запустили моделювання, використовуючи один пул потоків, змінюючи кількість потоків. Як показано на рисунку 3.5(d), отримане тут прискорення є найкращим (3.91×). Це тому, що коли ми використовували два пули потоків, кожен з них виконується паралельно, але один пул за іншим, послідовно. З одним єдиним пулом потоків, усі функції переходу виконуються паралельно, а потім лінійне покращення прискорення обмежується лише кількістю ЦП та операціями введення/виведення, якщо такі є. Зауважте, що пік прискорення досягається завжди коли кількість потоків дорівнює кількості ЦП. Хоча четвертий випадок, з рівнем пулу одного потоку, досягає найкращого прискорення, ми вважаємо, виходячи з нашого досвіду, що в тих самих симуляціях реального світу надання ресурсів для найповільніших моделей може бути цікаво, особливо коли різниця між часом моделювання найповільнішої та найшвидшої моделей занадто великий.

Як згадувалося вище, розподілене моделювання використовувало до восьми вузлів з 4 vCPU, 32 ГБ пам'яті. Тому ми повторили попередні паралельні експерименти на 8×4 = 32 ЦП Intel(R) Xeon(R) на 2,8 ГГц і 8×32 = 256 ГіБ оперативної пам'яті. На рисунку 3.6 зображено результати моделювання. Вони якісно однакові.

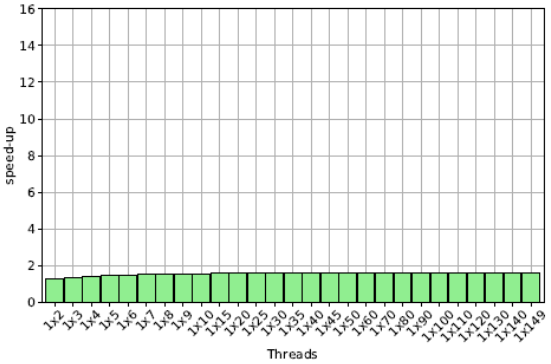
Коли збільшення кількості потоків у пулі L_1 (рисунок 3.6(a)) максимальне, прискорення, 2.19×, було отримано, коли кількість потоків дорівнювала кількість найповільніших моделей: 49, що означає, що 32 ЦП

змогли впоратися з обслуговуванням усіх моделі. Те саме сталося, коли ресурси перейшли до швидкого потоку, тобто максимальне прискорення, 1.63×, було досягнуто з певною кількістю паралельних потоків, що дорівнює кількості найбільш швидких моделей: 149 (рисунок 3.5(b)). Як можна вивести з двох попередніх цифр, коли параметр $n > m$ збільшується, збільшується прискорення.

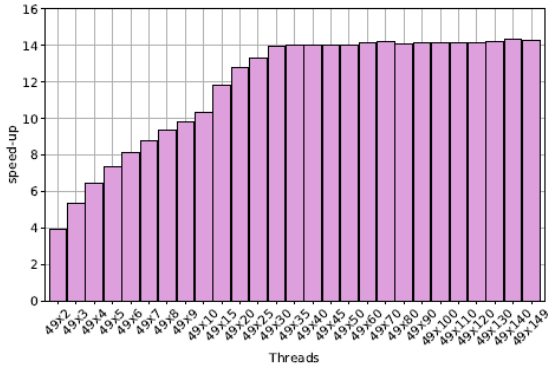
Оптимальний підхід (рисунок 3.6(c)) отримав надзвичайне покращення порівняно з віртуальною машиною з 4 ЦП – 14.33×. Крім того, як показано на рисунку 3.6(d), збалансоване прискорення в цьому випадку набагато краще, ніж у попередні, 15.94×. Однак спостерігається втрата ефективності з 4 vCPU до 32 vCPU (15.94<8·3.91). Це пояснюється зростанням витрат часу на передачу даних, синхронізацію з одночасним зменшення обчислювальної складності окремої задачі.



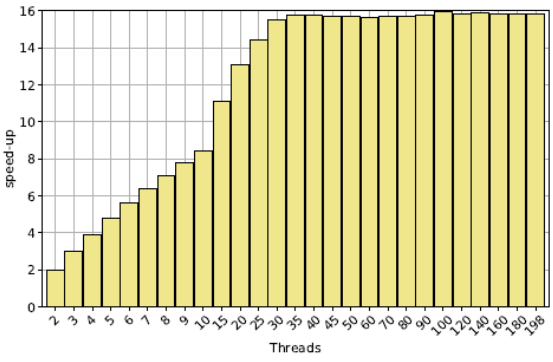
(a) Threads for slow models



(b) Threads for fast models



(c) Sub-optimal approach



(d) Balanced distribution

Рисунок 3.6 – Розподіл ресурсів (потоків) і прискорення для паралельної симуляції на 32 vCPU

3.3.4 Розподілене моделювання

У цьому розділі ми аналізуємо обчислювальну вартість розподіленого моделювання. на основі політики розподілу контейнерів і описаної архітектури раніше.

З цією метою ми застосували стратегію інкрементного контейнера, подібну до тієї, яка використовується в паралельному підході, використовуючи дворівневу чергу (замість потоку) та пули для виділення контейнерів (замість потоків), позначених L_1 і L_2 на рисунку 3.2. Варто нагадати, що будь-яка політика розподілу може бути використана, редагуючи XML-файл, що описує сплощену структуру моделі та контейнери, де розміщено кожен атом. Як зазначалося вище, L_1 було зарезервовано для atomic моделі з високими обчислювальними вимогами (тобто повільніші атомні моделі), тоді як рівень L_1 використовується для виділення решти моделей (тобто швидших атомарних моделей).

У першій серії експериментів ми збільшили кількість контейнерів L_1 виділили один єдиний контейнер в L_2, надаючи більше ресурсів моделям із вищою обчислювальною вартістю. Як тільки ми знайшли оптимальну кількість контейнерів L_1 (тобто де більше немає запасу для покращення продуктивності), ми збільшуємо кількість контейнерів в L_2, як ми робили в паралельному підході, щоб знайти 2-рівневу оптимальну конфігурацію. У другій серії експериментів ми просто використовуємо один рівень, щоб розподілити всі контейнери, збалансувавши розподіл атомних моделей серед них. Для виконання розподіленого моделювання ми використовували кластер GKE з 8 вузлами n2-highmem-4. Ці вузли включають 4 vCPU Intel(R) Xeon(R) на 2,8 ГГц і 32 ГіБ оперативної пам’яті. На рисунку 3.7 показано результати цього аналізу з точки зору прискорення. Як і в паралельному моделюванні, смуги позначені як $i \times j$, де i представляє число контейнерів (капсул), створених у L_1 і j пулів, створених в L_2. Як ми бачимо на синіх смугах на малюнку рисунку 3.7, прискорення збільшується зі збільшенням кількості

блоків, які створено для повільніших моделей в L_1, досягаючи максимального значення 0.71× в 7×1. Однак результати раптово погіршуються, починаючи з 8×1 розподілу заздалегідь. Це пов'язано з кількістю вузлів, присутніх у кластері. Тоді як 7×1 сценарій розподіляє рівно один пакет на вузол, наступні сценарії представляють вузли з декількома модулями. Прискорення менше ніж 1 у цьому випадку. Це тому, що розподілене моделювання відрізняється від паралельного, в якому всі 198 симуляторів виконуються як незалежні процеси віртуальної машини Java (JVM), незалежно від кількості модулів, тобто ресурси пам'яті, необхідні для запуску розподіленого моделювання, значно вище, ніж у паралельному рішенні.

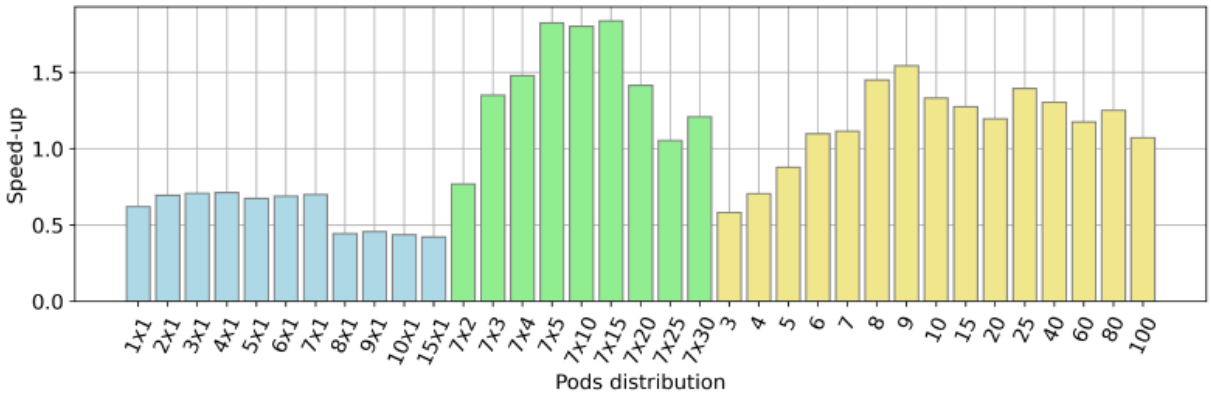


Рисунок 3.7 – Прискорення розподіленого моделювання залежно від кількості контейнерів (L_1× L_2)

Так як в конфігурації 8×1 більше контейнерів, ніж вузлів, також менше ресурсів для виконання 49 найшвидших моделей, різко збільшується час виконання та, як наслідок, зменшується прискорення. В результаті, оптимальна кількість контейнерів в L_1 з одним контейнером L_2 досягається, коли всередині є 7 контейнерів L_1. Крім цієї кількості, збільшення кількості контейнерів не принесе користі в L_1 без збільшення кількості контейнерів L_2.

Після цього кількість контейнерів у L_1 є фіксованою і дорівнює оптимальному значенню, а потім розмір L_1 збільшується, шукаючи оптимальну конфігурацію, як і в паралельному випадку. Це можна побачити

на зелених смугах на рисунку 3.7, які показують, що збільшення контейнерів у L_2 також помітно покращує продуктивність із максимальним значенням прискорення 1.84×7×15. Цей факт не підсилює нашу $n > m$ гіпотезу, оскільки 149 екземплярів віртуальної машини Java (JVM) споживають значні обсяги пам'яті та стають вузьким місцем, надаючи перевагу вищім значенням для m, що також пояснює низьке значення прискорення.

Нарешті, ми використали єдиний рівень контейнера, тобто один єдиний клас для розподілу усіх атомарних моделі та збалансували відповідно до їх затримок. Жовті смуги в на рисунку 3.7 показують, що ця конфігурація може давати до 1.45×. Знов таки швидкість збільшується разом із кількістю пулів, доки вони не стануть приблизно рівними кількісті вузлів.

У розподіленій версії переваги поділу ресурсів на рівнях видно не так чітко, як у паралельній версії, оскільки, як зазначено вище (a) усі атомарні моделі виконуються одночасно, і (b) один із рівнів може діяти як вузьке місце, коли ресурси (переважно пам'ять), зарезервовані для цього рівня є недостатніми. Цю неефективність можна пояснити проблемами розповсюдження, яким є випадок у розподіленому моделюванні. Однак ми підтвердили, що вузьке місце не є зв'язком між вузлами, оскільки при встановленні затримок рівними 0 секундам ($\Delta_i=0$), прискорення, отримані паралельною машиною з 32 ЦП і поширена версія була еквівалентною (21×проти 19×). Майбутня робота включає вивчення механізмів полегшення ваги процесів JVM.

3.4 Порівняння паралельного та розподіленого моделювання

Щоб порівняти нашу паралельну та розподілену архітектури, повинні враховуватися чотири метрики: продуктивність, вартість, вартість/продуктивність і використання обладнання. Найкращу продуктивність досягає Parallel 32 збалансована конфігурація vCPU, майже в 16 разів швидше, ніж послідовна симуляція.

З точки зору вартості, найдешевшим рішенням є, звичайно, паралельний 4 vCPU підхід із місячною вартістю орієнтовно 168 доларів США. Його використання підтримується кластером Kubernetes із 1347 дол. США на місяць. Нарешті, 32 vCPU virtual machine є найдорожчим рішенням з $1414/міс. Вартість/продуктивність однієї збалансованої точки прискорення становить $43, $89 і $732 для віртуальних машин з 4, 32 vCPU і розподілених рішень відповідно. Очевидно, що розподілена інфраструктура є повністю насиченою і не повинена бути спрямована на фактор рентабельності.

Нарешті, щодо основного апаратного забезпечення, розподілені рішення є більш гнучкими, оскільки вони підтримують різнорідні архітектури, та моделювання базується на сокет-розподілену програму, сумісну з будь-яким дистрибутивом обладнання.

Паралельні рішення дійсні лише для систем із спільною пам’яттю та гомогенною архітектурою.

Попереду ще багато роботи в області розподіленого моделювання, очевидно, стосовно управління пам'яттю незалежними розподіленими процесами, що є величезним вузьким місцем, яке необхідно усунути. У будь-якому випадку, щодо можливих складних зв’язків навколо розподіленої системи, структура M&S, представлена в цьому документі, дозволяє нам уніфіковане послідовне, паралельне та розподілене рішення, яке полегшує розгортання будь-якої конфігурації, повністю зосереджуючись на моделі автоматичне розгортання.

ВИСНОВКИ

Імітаційне моделювання - це діяльність із запуску симулятора в обчислювальному середовищі, яке розвивалося з часом. Сьогодні воно складається з різноманітних варіантів, таких як локальна робоча станція, розподілена мережа, багатоядерна, віртуалізована інфраструктура, інфраструктура HPC або контейнеризованане хмарне середовище. Розширена та масштабована архітектура моделювання повинна бути здатна безперебійно працювати в будь-якому обчислювальному середовищі. На жаль, більшість архітектур симуляції не розроблені як розширювані та масштабовані, особливо коли потрібна велика кількість симуляційних прогонів від моделі, розробленої для виконання на локальному робочому комп'ютері, яка не може працювати в інших високопродуктивних середовищах. Добре відомий факт, що послідовні програми, які були розроблені для одного ЦП, не отримують жодних переваг від їх виконання на багатоядерному ЦП. Те саме стосується архітектур симуляції.

Проблема ускладнюється, коли формалізм моделі тісно пов’язаний з архітектурою симуляції, і обидва потрібно переписати для виконання в іншому середовищі виконання обчислень, ніж оригінальне.

Формалізм DEVS категорично розділяє шари моделювання та симуляції так що архітектура моделювання є прозорою для архітектури моделі, і обидві можуть розвиватися горизонтально. За останні 15 років роботи Міттала і Мартіна продемонстрували цей аспект виконання моделей DEVS та інших доменно-спеціальних моделях (DSM) з їх відображеннями DEVS над прозорими архітектурами моделювання. У цієї роботі надано докази, які сприяють їх попередній роботі з механізмом моделювання xDEVS M&S, здатним розгортати симулятор у паралельній багатоядерній архітектурі та в розподіленій мережній архітектурі нерозривним способом. Ми описали об'єднуючу архітектуру поєднання двох координаторів DEVS, які запускають

ту саму модель DEVS як у паралельній, так і в розподіленій архітектурі. Хоча ця базова концепція відрізняється від координаторів для різних платформ розгортання, як було представлено в Zeigler's [9], він потребував деяких удосконалень для їх використання на хмарних платформах.

Ці два координатори були додатково розгорнуті в контейнерному середовищі з підтримкою хмари, що зробило інфраструктуру моделювання дійсно прозорою для моделі. Як паралельна, так і розподілена реалізації DEVS-сумісні.

Це гарантує для послідовного, паралельного та розподіленого моделювання точно такі ж результати.

Була зроблена оцінка продуктивності паралельної системи моделювання для паралельного та розподіленого координаторів розподіленого моделювання за допомогою стенда DEVStone і остаточно отримали 16-кратне прискорення від паралельного координатора, і прискорення в 1,84 рази за допомогою розподіленого координатора для заданої апаратної конфігурації. Для паралельного моделювання ми досягли наступного:

- підтверджено нашу гіпотезу про те, що більше призначення потоків до пулу потоків, який містить моделі з інтенсивним використанням процесора, забезпечує більш високе покращення прискорення;
- пік прискорення досягається, коли кількість потоків у пулі потоків дорівнює кількості ЦП;
- коефіцієнт витрат і вигод набагато вищий порівняно з розподіленим моделюванням сценарій використання продуктивності.

Цей результат демонструє, що паралельне виконання будь-якої моделі DEVS слід надавати перевагу над будь-яким розподіленим виконанням. Цей результат далі розширений до всього випадку розподіленого моделювання, яке ніколи не може зрівнятися з результатами отримані за допомогою паралельних архітектур [33].

Розподілені обчислювальні архітектури з’явилися раніше, ніж багатоядерні паралельні обчислювальні архітектури. Мотивація для розподілених обчислень (до появи Інтернет), який мав з’єднати територіально розподілені об’єкти вирішення складної проблеми тепер поступилося місцем паралельним обчисленням, де ресурси доступні або в HPC, або в хмарному середовищі та прозоро доступні для використання. Відповідно, архітектури M&S (обидві успадковані та майбутні) повинні розвиватися, щоб отримати вигоду від паралельного виконання з підтримкою хмарних обчислювальних архітектури. Дотримання формалізмів, таких як DEVS (і пов’язаних реалізацій xDEVS), які забезпечують міцну основу для складених M&S архітектури є кращим шляхом. Різні алгоритми, функції та API розроблені в рамках xDEVS, забезпечують простоту використання, розширюваність і масштабованість до будь-якої симуляції DEVS. Повідомляється, що xDEVS є найефективнішим симулятором DEVS на сьогодні [29], і ця робота розширює його можливості до хмарного паралельного та розподіленого моделювання.

Незважаючи на те, що архітектури паралельного моделювання забезпечують прискорення моделювання, у високопродуктивному середовищі для оптимізації та аналізу моделі, архітектури розподіленого моделювання продовжуватимуть знаходити свої ніші в навчанні, тестуванні та оцінці сумісності в різних середовищах та в інтеграції нових систем.

Майбутня робота. Ми встановили аргументи для збільшення використання паралельної архітектури та порівняли з розподіленими архітектурами. Однак є ідея об’єднати ці дві архітектури для досягнення максимальної ефективності. Майбутня робота включає в себе використання таких гібридних розгортань, де розподілені вузли можуть виконувати паралельне моделювання. Це вимагатиме модифікації моделі xDEVS і вимагає серйозних зусиль для розробки. Також доцільно провести порівняльне дослідження з іншими двигунами моделювання DEVS, які включають паралельний інтерфейс і підтримку об’єднуючої архітектури.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Fujimoto, R. M. Parallel and Distributed Simulation Systems, Wiley, 2000.

2. Fujimoto, R. M. Research challenges in parallel and distributed simulation, ACM Transactions on Modeling and Computer Simulation (TOMACS) 26 (4) (2016) 1-29

3. D'Angelo, G., Ferretti, S., Ghini V., Multi-level simulation of internet of things on smart territories, Simulation Modelling Practice and Theory 73 (2017) 3-21. doi:https://doi.org/10.1016/j.simpat.2016.10.008.

4. Kratzke, N., Siegfried, R. Towards cloud-native simulations - lessons learned from the front-line of cloud computing, The Journal of Defense Modeling and Simulation 18 (1) (2021) 39-58. doi:10.1177/1548512919895327.

5. Buyya, R., Srirama, S.N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. D. C. D. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D.Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, H. Shen, H. A manifesto for future generation cloud computing: Research directions for the next decade, ACM Computing Surveys 51 (5) (Nov. 2018). doi:10.1145/3241737.

6. Kewley, R., Kester, N., McDonnell, J. Devs distributed modeling frame work - a parallel devs implementation via microservices, in: 2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS), 2016, pp. 1-8. doi:10.23919/TMS.2016.7918828.

7. Mell, P., Grance, P. The NIST denition of Cloud Computing, Tech. rep., National Institute of Standards and Technology (2021). doi:10.6028/NIST.SP.800-145.

8. D'Angelo, G., Marzolla, M. New trends in parallel and distributed simulation: From many-cores to cloud computing, Simulation Modelling Practice

and Theory 49 (2014) 320-335. doi:https://doi.org/10.1016/j.simpat.2014.06.007.

9. Zeigler, B.P., Praehofer, G., Kim, T. G. Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd Edition, Academic Press, 2000.

10. Risco-Martin, J.L. xDEVS: A cross-platform discrete event system simulator, https://github.com/iscar-ucm/xdevs (2014 (accessed September 20, 2021)).

11. Bagrodia, R., Meyer, R., Takai, M., Chen, Y.-A., Zeng, X., Martin, J., Song, H.Y. Parsec: a parallel simulation environment for complex systems, Computer 31 (10) (1998) 77-85. doi:10.1109/2.722293.

12. Aniszewski, W., Arrufat, T., Crialesi-Esposito, M., Dabiri, S., Fuster, D. Ling, Y., Lu, J., Malan, L. Parallel, robust, interface simula tor (paris), Computer Physics Communications 263 (2021) 107849. doi:https://doi.org/10.1016/j.cpc.2021.107849.

13. Ubal, R. Jang, B. Mistry, P., Schaa, D., Kaeli, D. Multi2sim: A simulation framework for cpu-gpu computing, in: 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2012, pp. 335-344.

14. Wang, R.M., Thakur, C. S., van Schaik, A. An fpga-based massively parallel neuromorphic cortex simulator, Frontiers in Neuroscience 12 (2018) 213. doi:10.3389/fnins.2018.00213. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00213

15. Pelkey, J., Riley, G. Distributed simulation with mpi in ns-3, in: Proceed ings of the 4th International ICST Conference on Simulation Tools and Techniques, 2011, pp. 410-414.

16. Hofer, R.C., Loper, M. L. Dis today [distributed interactive simulation], Proceedings of the IEEE 83 (8) (1995) 1124-1137.

17. Dahmann, J.S., Fujimoto, R. M., Weatherly, R. M. The department of defense high level architecture, in: Proceedings of the 29th conference on Winter simulation, 1997, pp. 142-149.

18. Liu, Q., Wainer, G. A performance evaluation of the lightweight time warp protocol in optimistic parallel simulation of devs-based environmental models, in: 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation, IEEE, 2009, pp. 27-34

19. Nutaro, J. On constructing optimistic simulation algorithms for the discrete event system specication, ACM Transactions on Modeling and Computer Simulation (TOMACS) 19 (1) (2009) 1-21.

20. Lanuza, J., Trabes, G.G., Wainer, G.A. Parallel execution of devs in shared memory multicore architectures, in: 2020 Spring Simulation Conference (SpringSim), IEEE, 2020, pp. 1-11.

21. Mittal, S., Martin, J. L. R. Netcentric System of Systems Engineering with DEVS Unied Process, 1st Edition, CRC Press, 2013.

22. Mittal, S., Risco-Martin, J.L. DEVSML Studio: a framework for integrating domain-specic languages for discrete and continuous hybrid systems into DEVS-based M&S environment, in: Summer Simulation Multiconference (SummerSim 2016), 2016.

23. Risco, J. L., Mittal, S., Atienza, D., Hidalgo, J.I., Lanchares, J. Optimization of dynamic data types in embedded systems using DEVS/SOA-based modeling and simulation, in: Proceedings of the 3rd International ICST Conference on Scalable Information Systems 2009, no. CONF, ICST, 2008, pp. 1-11.

24. Mittal, S., Risco-Martin, J. L., Zeigler, B.P. DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in DEVS Unied Process, SIMULATION 85 (7) (2009) 419-450.

25. Al-Zoubi, K., Wainer, G. Performing distributed simulation with restful web-services, in: Proceedings of the 2009 Winter Simulation Conference (WSC), IEEE, 2009, pp. 1323-1334.

26. Y. Van Tendeloo, H. Vangheluwe, Pythonpdevs: a distributed parallel devs simulator., in: SpringSim (TMS-DEVS), 2015, pp. 91-98.

27. Gannon, D., Barga, R., Sundaresan, N. Cloud-native applications, IEEE Cloud Computing 4 (5) (2017) 16-21. doi:10.1109/MCC.2017.4250939.

28. Mittal, S., Risco-Martin, J.L. DEVSML 3.0 stack: rapid deployment of DEVS farm in distributed cloud environment using microservices and containers, in: Spring Simulation Multiconference (SpringSim), 2017, pp. 1-19.

29. Risco-Martin, J.L., Mittal, S., Fabero, J. C., Zapater, M., Hermida, R. Reconsidering the performance of DEVS modeling and simulation environments using the DEVStone benchmark, Simulation 93 (6) (2017) 459-476.

30. Glinsky, E., Wainer, G. DEVStone: a benchmarking technique for studying performance of DEVS modeling and simulation environments, in: Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications, IEEE, 2005, pp. 265-272.

31. Van Tendeloo, Y., Vangheluwe, H. The modular architecture of the Python (P) DEVS simulation kernel, in: Proceedings of the 2014 Symposium on Theory of Modeling and Simulation-DEVS, 2014, pp. 387-392.

32. Cardenas, R., Henares, K., Arroba, P., Wainer, G., Risco-Martin, J. L., A DEVS simulation algorithm based on shared memory for enhancing per formance, in: 2020 Winter Simulation Conference (WSC), 2020, pp. 2184 - 2195.

33. Волк М.О., Лабазов В.Г., Кипаренко Д.О., Привалов Б.В., Шумов Д.О., Самойлов І.А. Розподілене комп'ютерне моделювання в системах хмарних обчислень. Вісник Херсонського національного технічного університету. No 1(88), 2024 с. 211-217. DOI: https://doi.org/10.35546/kntu2078-4481.2024.1.29. **Фахове видання**