Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Використання обчислювального сервісу AWS для розв'язання просторових задач розбиття множин»

Виконав: студент групи К23-1М
Спеціальність 122 «Комп'ютерні науки»
Григорук П.О.
(прізвище та ініціали)

Керівник к.ф.-м.н., доц. Фірсов О.Д.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місто роботи)

(посада)

(посада)

(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Григорук П.О. Використання обчислювального сервісу AWS для розв’язання просторових задач розбиття множин.

Дипломна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп’ютерні науки» – Університет митної справи та фінансів, Дніпро, 2025.

Існує широкий клас теоретичних задач, які мають назву неперервними задачам оптимального розбиття множин (ОРМ). Для цих задач у лінійному випадку вдається отримати аналітичний розв’язок, у явному вигляді, при цьому в аналітичний вираз входять параметри, які відшукуються як оптимальний розв’язок двоїстої скінченновимірної задачі оптимізації з негладким цільовим функціоналом. Для розв’язання негладкої скінченновимірної задачі, застосовуються методи недиференційованої оптимізації. У випадку нелінійних критеріїв якості розв’язок вихідної нескінченновимірної задачі зводиться до розв’язання деякого допоміжного операторного рівняння з параметрами. Актуальність таких задач зумовлена їх практичною значимістю, оскільки при математичному моделюванні реальних процесів часто потрібно враховувати зміну стану системи з часом. Також виникає необхідність розв’язання задач по розрахунку розміщення об’єктів пов’язаних із транспортною інфраструктурою.

Особливістю розв’язання описаних задач є застосування чисельних методів для отримання рішення. Важливим аспектом при розв’язанні задачі є перехід із нескінченновимірного представлення до дискретного. Етап та обґрунтування такого переходу є принциповим моментом для застосування чисельних методів та обчислювальних процедур.

Використання сіток для організації обчислень вимагає великих обчислювальних ресурсів. Особливість ситуації полягає у тому, що попит на ресурси виникає у процесі ускладнення задачі та уточнення розв’язку. Тобто проблема полягає у тому, що треба мати значні обчислювальні ресурси незалежно від формулювання конкретної задачі.

Сучасним середовищем, яке надає практично безмежні обчислювальні ресурси є хмари. Один з найпотужніших хмарних сервісів є Amazon Web Services (AWS).

Магістерська робота присвячена застосуванню Amazon Web Services (AWS) для чисельного розвязання неперервних задач оптимального розбиття множин: задач OPM, в яких деякі параметри змінюються з часом.

Ключові слова: множина, модель, оптимізація, розбиття, AWS, HPC.

ABSTRACT

Hrygoruck P.O. Using the AWS computing service to solve spatial set partitioning problems.

Diploma Thesis for the degree of Master in specialty 122 "Computer Science" - University of Customs and Finance, Dnipro, 2025.

There is a wide class of theoretical problems called continuous optimal set partitioning problems (OPP). For these problems in the linear case, it is possible to obtain an analytical solution in an explicit form, while the analytical expression includes parameters that are sought as the optimal solution of a dual finite-dimensional optimization problem with a non-smooth objective functional. To solve a non-smooth finite-dimensional problem, non-differentiated optimization methods are used. In the case of nonlinear quality criteria, the solution of the initial infinite-dimensional problem is reduced to the solution of some auxiliary operator equation with parameters. The relevance of such problems is due to their practical significance, since when mathematically modeling real processes it is often necessary to take into account the change in the state of the system over time. There is also a need to solve problems for calculating the placement of objects related to transport infrastructure.

A feature of solving the described problems is the use of numerical methods to obtain a solution. An important aspect when solving a problem is the transition from an infinite-dimensional representation to a discrete one. The stage and justification of such a transition is a fundamental point for the application of numerical methods and computational procedures.

The use of grids to organize calculations requires large computational resources. The peculiarity of the situation is that the demand for resources arises in the process of complicating the problem and refining the solution. That is, the problem is that it is necessary to have significant computational resources regardless of the formulation of a specific problem.

A modern environment that provides practically unlimited computational resources is the cloud. One of the most powerful cloud services is Amazon Web Services (AWS).

The master's thesis is devoted to the use of Amazon Web Services (AWS) for numerical solution of continuous optimal set partitioning problems: ORM problems, in which some parameters change over time.

Keywords: set, model, optimization, partitioning, AWS, HPC.

ЗМІСТ

ВСТУП

У багатьох галузях сучасного життя, наприклад транспортних технологіях, системах навігації, управлінні транспортними засобами, постає проблема знайти розв’язання оптимізаційних задач. Можна виділити такі задачі: задачу розміщення джерел постачання палива та виділення зон із санітарно-припустимими нормами забруднення під час планування виробничих приміщень, задачу раціонального керування транспортом, задачу районування транспортної мережі. В залежності від наявності обмежень на деякі параметри задачі, властивостей множини, цілей, які необхідно досягти, можна виділити велику кількість задач оптимального розбиття множин (ОРМ). Все це розмаїття задач ОРМ умовно можна поділити на два класи. До одного класу входять дискретні задачі ОРМ, у яких множина, що підлягає розбиттю, містить скінченну кількість елементів. Інший клас містить задачі, у яких множина, що розбивається, є континуальною. Такі задачі називають неперервними задачами ОРМ. Для цих задач у лінійному випадку вдається отримати аналітичний розв’язок, у явному вигляді, при цьому в аналітичний вираз входять параметри, які відшукуються як оптимальний розв’язок двоїстої скінченновимірної задачі оптимізації з негладким цільовим функціоналом. Для розв’язання негладкої скінченновимірної задачі, застосовуються методи недиференційованої оптимізації – r-алгоритм Шора та його модифікації. У випадку нелінійних критеріїв якості розв’язок вихідної нескінченновимірної задачі зводиться до розв’язання деякого допоміжного операторного рівняння з параметрами. Актуальність таких задач зумовлена їх практичною значимістю, оскільки при математичному моделюванні реальних процесів часто потрібно враховувати зміну стану системи з часом. Також виникає необхідність розв’язання задач по розрахунку розміщення об’єктів пов’язаних із транспортною інфраструктурою.

Особливістю розв’язання описаних задач є застосування чисельних методів для отримання рішення. Важливим аспектом при розв’язанні задачі є

перехід із нескінченновимірного представлення до дискретного. Етап та обґрунтування такого переходу є принциповим моментом для застосування чисельних методів та обчислювальних процедур. Фактично можна запропонувати два шляхи пошуку розв'язку, чи застосовувати методи локальної оптимізації на неперервній множині, розуміючи відповідні обмеження, чи використовувати методи глобальної оптимізації, але використовуючи методи типу "брут форс".

Використання сіток для організації обчислень вимагає великих обчислювальних ресурсів. Особливість ситуації полягає у тому, що попит на ресурси виникає у процесі ускладнення задачі та уточнення розв'язку. Тобто проблема полягає у тому, що треба мати значні обчислювальні ресурси незалежно від формулювання конкретної задачі.

Сучасним середовищем, яке надає практично безмежні обчислювальні ресурси є хмари. Один з найпотужніших хмарних сервісів є Amazon Web Services (AWS).

AWS Lambda — це безсерверні обчислення, які дають змогу розробникам виконувати код, не турбуючись про керування серверами, що робить його важливим компонентом поточної безсерверної архітектури. Він працює, виконуючи ваш код у високодоступному обчислювальному середовищі, яке керує всіма аспектами обчислювальних ресурсів, такими як керування сервером, операційна система, обчислювальна потужність, масштабування та ведення журналу.

Магістерська робота присвячена застосуванню Amazon Web Services (AWS) для чисельного розвязання неперервних динамічних задач оптимального розбиття множин: задач ОРМ, в яких деякі параметри змінюються з часом.

Метою роботи є розробка і обґрунтування технології організації обчислювального процесу для виконання спеціального алгоритма розв'язання неперервних задач оптимального розбиття множин у частковому випадку із заданими обмеженнями.

Об'єктом дослідження є безсерверні обчислення Amazon Web Services (AWS) .

Предметом дослідження є технологія обчислень за допомогою Amazon Web Services (AWS) для розв'язання неперервних задач ОРМ для спеціальних випадків.

Методи дослідження, що використовуються у магістерській роботі – це безсерверні хмарні обчислення, паралельні обчислення, чисельні методи розв'язання оптимізаційних задач, методи нескінченновимірної оптимізації, теорія неперервних задач оптимального розбиття множин(ОРМ).

Структура магістерської роботи: робота складається з трьох розділів, об'єм роботи – 68 сторінок, робота містить 8 рисунків, перелік використаних джерел має 17 посилань.

РОЗДІЛ 1. ДОССЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Задачі оптимального розбиття множин

У реальному житті задачі оптимального розбиття множин (ОРМ) виникають у багатьох галузях життя людини, наприклад: планування перевезення вантажів, складання розкладів авіаперельотів, балансування потужностей, розміщення підприємств.

Багато з них можуть бути сформульовані як задачі *скінченновимірної* оптимізації, так і як задачі *нескінченновимірної* оптимізації.

У більшості практичних випадків, через складність реальних процесів, доводиться вводити ряд обмежень на параметри задачі і формулювати задачу оптимального розбиття множин *з обмеженнями*.

За іншою класифікацією, усі задачі теорії оптимального розбиття множин умовно можна поділити на *дискретні* – коли множина, що розбивається, має скінченну потужність, та *неперервні* – у противному випадку.

Математична постановка дискретної задачі ОРМ наступна:

$$(c, x) \to \min , \tag{1.1}$$

$$Ax = e , \tag{1.2}$$

$$x_j = 0 \vee 1, \quad \forall j \in N . \tag{1.3}$$

де A – матриця розміру $m \times n$ із 0 та 1, c – довільний n-вимірний вектор, $e(1,...,1)$ – це m-вимірний вектор, $N = \{1,...,n\}$.

Детальний огляд дискретних задач оптимального розбиття зробили E. Balas та M. W. Padberg. Деякі із застосувань цієї задачі, описані у літературі, такі: планування залізничних перевезень, вантажних перевезень, авіаперельотів, створення маршрутів для танкерів, інформаційного пошуку, планування перемикання електричних схем, розкрою, балансування лінії

збірки, розташування морських бурових платформ, деякі інші задачі розміщення підприємств, політичних округів та інші.

Усі відомі алгоритми розв'язання дискретних задач ОРМ можна поділити на дві групи: *точні* та *евристичні*. Перші знаходять оптимальний розв'язок задачі ОРМ, другі – намагаються знайти «гарний» розв'язок, проте швидко.

Більшість точних алгоритмів розв'язують ЛП-релаксацію (тобто від обмежень (1.3) здійснюється перехід до обмежень виду $0 \le x_j \le 1, \quad \forall j \in N$).

G. Laporte зробив огляд точних та наближених підходів до розв'язання задачі маршрутизації транспорту (Vehicle Routing Problem, VRP), яка полягає у тому, щоб для деякого парку транспортних засобів, який розташовано у одному або декількох депо, визначити оптимальний набір маршрутів до деякої кількості віддалених споживачів.

Один із поширених точних підходів до розв'язання задачі цілочисельного лінійного програмування – це метод гілок та меж, або пошук по дереву.

Евристичних алгоритмів, що розв'язують задачу ОРМ небагато, оскільки, як правило, задача має велику кількість обмежень, і виникає проблема пошуку її допустимого розв'язку. P.S. Chu та J.E. Beasley для задачі оптимального розбиття запропонували евристику, що базується на генетичному алгоритмі, у якому за фітнес-функцію взято суму цільової функції та штрафної.

Коли споживачів дуже багато, то формулювання скінченновимірної задачі ОРМ стає недоцільним через труднощі, пов'язані з розв'язанням задач великої розмірності, і виникає потреба у розгляданні нескінченновимірних аналогів таких задач. Яскравим прикладом таких задач є нескінченновимірна задача розміщення підприємств з одночасним розбиттям неперервно заповненого споживачами регіону на підмножини, кожна з яких обслуговується одним підприємством, з метою мінімізації транспортних та виробничих витрат. Споживачами тут можуть бути виборці, абоненти, точки зрошувальної території тощо.

Також виникають задачі, що зводяться до задач ОРМ, в яких множина, що підлягає розбиттю, нескінченна за своєю природою. Наприклад, до таких задач відносяться задача пошуку областей тяжіння локальних мінімумів багатоекстремальної функції, неперервні задачі покриття, задача Неймана-Пірсона та ін. [11].

Канторович формулює задачу про переміщення мас, математична постановка якої наступна, [1].

Задано два розподіли мас, що визначаються адитивними функціями сукупності $\Phi(e)$ та $\Phi'(e)$, причому $\Phi(R)=\Phi'(R)=1$. Переміщенням мас називається функція $\Psi(e,e')$, яка характеризує масу, переміщену із сукупності e у e', $[\Psi(e,R)=\Phi(e), \Psi(R,e')=\Phi'(e')]$. Робота, пов'язана із здійсненням даного переміщення, визначається інтегралом

$$W(\Psi)=\int_R\int_R r(x,y)\Psi(de,de'),$$

де $r(x,y)$ – відстань між точками x та y. Необхідно знайти найвигідніше переміщення Ψ_0, для якого робота W була б мінімальна.

Ця задача виникла як узагальнення сформульованої раніше практичної задачі про прикріплення пунктів споживання до пунктів виробництва. Пізніше виявилося, що сформульована загальна задача про переміщення мас містить задачу Монжа як частинний випадок, [16].

У роботі [13] досліджується питання: коли і як дискретну задачу оптимізації можна звести до неперервної? Тут зазначено, що у більшості

публікацій для здійснення редукції дискретної задачі оптимізації до неперервної пропонується наступний підхід. Вихідна задача записується в такий спосіб:

$$f(x)\to\min_{x\in\Omega},\quad \Omega\subset R^n,$$

де Ω – дискретна множина. Шукаються неперервна множина $\Omega_c \supset \Omega$ та штрафна функція ϕ, такі що $(f+\phi)|_\Omega = f|_\Omega$ і $f+\phi$ увігнута на Ω_c. Встановлено [84], що неперервна увігнута задача $\min\{f(x)+\phi(x) : x \in \Omega_c\}$ має розв'язок в точці екстремуму $x^* \in \Omega$. Тому, в деякому сенсі ця задача еквівалентна вихідній дискретній задачі.

Оскільки ітераційні неперервні алгоритми приводять у стаціонарну точку, відшукання глобального мінімуму залишається важкою задачею. Тому, в той час як вони мають цікаве теоретичне значення, на практиці вони незастосовні.

Brimberg J. та ін. у розглянули клас задач розміщення підприємств, відомих як багатоджерельні задачі Вебера (також відомі як неперервні задачі розміщення-розподілу). Ціль в таких задачах – «згенерувати» точки у неперервному просторі для розміщення нових підприємств відносно множини існуючих підприємств, розташованих у заданих точках простору.

Отже, до неперервних задач ОРМ відносять задачі, в яких розбиттю підлягає континуальна множина. Такі задачі виникають також у багатьох сферах сучасної науки, наприклад, у робототехніці, керуванні та біології. Все більш важливими у багатьох інженерних дисциплінах стають проблеми координації. Швидке розвертання великих груп автономних транспортних засобів стає можливим завдяки технологічним досягненням у комп'ютерних, мережевих системах та зменшенню розмірів електромеханічних систем. Ці мережі майбутнього з багатьма засобами пересунення мають координувати свої дії для виконання важких просторово-розподілених задач (наприклад, пошук та відновлення операцій, розвідка, наглядання та моніторинг навколишнього середовища для виявлення забруднень та оцінки). Такий сценарій майбутнього мотивує вивчення алгоритмів для автономії, адаптації та координації мереж з багатьма пересувними об'єктами. Також важливо взяти до уваги усі обмеження поведінки таких об'єктів.

Координаційні алгоритми повинні бути адаптивними та розподіленими для того, щоб отримана замкнена мережа могла масштабуватися, щоб дотримуватися обмежень пропускної спроможності, терпіти невдачі та адаптуватися до зміни середовищ, топології та змістовних задач. Закони взаємодії мають ці властивості і оптимізують показники усієї мережі для змістовних просторово-розподілених задач.

Corley H.W. та Roberts S.D. розглянули клас задач двовимірного локаційного аналізу, пов'язаних з визначенням оптимальних областей на площині згідно з різними критеріями. Задачі з цього класу ними названо *задачами проектування регіону*. Такі задачі виникають, наприклад, при районуванні, проектуванні підприємств, розміщенні складів та урбаністичному плануванні. Багато задач регіонального проектування можуть бути сформульовані як задачі оптимального розбиття множин.

Серед практичних моделей, які можуть бути сформульовані в рамках задачі ОРМ є *модель районування.*

Велика кількість неперервних задач оптимального розбиття множин можуть бути зведені до побудови так званих діаграм Вороного. За означенням, діаграма Вороного – це набір багатогранників, що називаються комірками Вороного, сформованих n точками-генераторами. Кожний генератор p_i міститься у комірці Вороного $V(p_i)$, яка має наступну властивість:

$$V(p_i) = \{q \mid d(p_i, q) \le d(p_j, q), i \ne j\},$$

де $d(x, y)$ – відстань від точки x до y.

Тобто, множина усіх таких q – це множина точок ближчих до p_i, ніж до будь-якого іншого p_j. Тоді діаграма Вороного задається так:

$$V = \{V(p_1), ..., V(p_n)\}.$$

Серед вітчизняних вчених, які досліджували неперервні задачі ОРМ, варто виділити таких: Туєв С.В., Сухарєв А. Г. , Кісельова О.М. [1; 2; 3; 8], Коряшкіна Л.С. , Ус С.А. та ін. Теорія неперервних задач ОРМ ґрунтується на єдиному підході, що полягає у зведенні вихідних нескінченновимірних задач оптимізації через функціонал Лагранжа до негладких, як правило, скінченновимірних задач оптимізації, для чисельного розв'язання яких застосовані сучасні ефективні методи недиференційовної оптимізації, а саме різні варіанти r-алгоритму Шора. Особливістю такого підходу є той факт, що розв'язок вихідних нескінченновимірних задач оптимізації вдається одержати в аналітичному вигляді, причому в аналітичний вираз входять параметри, які відшукуються як оптимальний розв'язок допоміжних скінченновимірних задач оптимізації з негладкими цільовими функціями у випадку лінійних задач ОРМ, або є розв'язком деякого допоміжного операторного рівняння з параметрами – у випадку нелінійних задач ОРМ [1].

У [9] сформульована і досліджена неперервна нелінійна задача оптимального розбиття множини Ω з E_n на неперетинні підмножини $\Omega_1,...,\Omega_N$ із відшуканням координат центрів (із розміщенням центрів) підмножин, невідомих заздалегідь, при обмеженнях у формі нерівностей. Математично обґрунтовано перехід від нескінченновимірних задач оптимізації через функціонал Лагранжа до двоїстих скінченновимірних негладких задач та до допоміжних операторних рівнянь, що аналітично пов'язують прямі та двоїсті змінні.

Як узагальнення задач досліджених у [9], досліджено неперервну нелінійну багатопродуктову задачу оптимального розбиття множини Ω з n-вимірного евклідового простору E_n на підмножини з заданим розміщенням центрів підмножин при обмеженнях у формі рівностей та нерівностей, яка має наступну математичну постановку:

нехай Ω – обмежена, вимірна за Лебегом множина в n -вимірному евклідовому просторі E_n. Необхідно розбити її на $M \times N$ вимірних за Лебегом підмножин $\Omega_1^1,...,\Omega_N^1;\ \Omega_1^2,...,\Omega_N^2;...;\ \Omega_1^M,...,\Omega_N^M$ (серед яких можуть бути і порожні) так, щоб

$$mes(\Omega_i^j \cap \Omega_k^j) = 0,\ i \neq k,\ i,k = 1,...,\ N,\ j = 1,...,\ M,$$

де mes(·) – міра Лебега,

$$\bigcup_{i=1}^{N} \Omega_i^j = \Omega,\ j = 1,2,...,\ M,$$

$$\sum_{j=1}^{M} \int_{\Omega_i^j} \rho^j(x)dx = b_i,\ i = 1,...,\ p,$$

$$\sum_{j=1}^{M} \int_{\Omega_i^j} \rho^j(x)dx \leq b_i,\ i = p+1,...,N,$$

та при цьому функціонал

$$F(\Omega_1^1,...,\Omega_N^1;\ \Omega_1^2,...,\Omega_N^2;...;\ \Omega_1^M,...,\ \Omega_N^M) =$$

$$= \sum_{j=1}^{M} \sum_{i=1}^{N} \left[\varphi_i^j \left(\int_{\Omega_i^j} \rho^j(x)dx \right) + \int_{\Omega_i^j} c^j(x,\tau_i)\rho^j(x)dx \right]$$

досягав свого мінімального значення.

1.2. Проблематика високовиробничіх обчислень

У роботі [17] академіком В.М. Глушковим окреслені основні напрямки досліджень в області високовиробничіх обчислень. Серед них можна виділити два основних, це: розробка паралельного програмного забезпечення та узгодження архітектур багатопроцесорних ЕОМ і обчислювальних процесів. Під розробкою програмного забезпечення розуміється не тільки розпаралелення арифметичних і логічних операцій, але й організація

обчислювального процесу в системі взаємодіючих процесорів, а також побудова нових паралельних алгоритмів розв’язку задач, тобто урахування паралелізму на всіх рівнях.

Для дослідження структури зв'язків на множині операцій, якою і є програма, В.В. Воєводіним було запропоновано використовувати орієнтовані графи. Дві операції називаються залежними, якщо результат виконання однієї операції є аргументом іншої. Нехай окремим операціям відповідають вершини графа. Залежні операції з'єднаємо дугами. Дуга завжди йде з тієї вершини, що відповідає операції, яка виконується раніше. Якщо дві множини операцій не зв'язані шляхами графів залежностей, то вони представляють незалежні гілки обчислень і їх можна виконувати паралельно. Тому основні зусилля фахівців в області вивчення структури програм були спрямовані на розробку необхідних і достатніх критеріїв незалежності множин операцій.

З метою автоматизації процесу аналізу програм, колективом лабораторії паралельних інформаційних технологій, була розроблена програмна система. Вона одержала назву V-Ray system. Система дозволила вивчати структуру програм, як на макро, так і на мікро рівні.

Сучасні багатопроцесорні системи можна розділити на універсальні і проблемно-орієнтовані. Проблемно-орієнтовані системи можуть ефективно розв’язувати тільки одну задачу, для якої вони створені. Універсальні, можуть розв’язувати будь-яку задачу, але їх продуктивність не є задовільною.

Співробітниками НДІ багатопроцесорних обчислювальних систем Каляєвим А.В., Каляєвим І.А., Левіним І.І. розроблена концепція багатопроцесорних обчислювальних систем із програмуємою архітектурою, що являє собою об'єднання ідеї універсальності і проблемної орієнтованості систем . Така архітектура дозволяє організовувати багатопроцесорну систему під структуру розв'язуваної задачі. Тобто в рамках універсальної паралельної обчислювальної системи можуть створюватися проблемно-орієнтовані обчислювальні структури. Завдяки цьому, реальна продуктивність паралельного комп'ютера з програмуємою архітектурою наближається до

максимальної при розв’язку широкого класу задач і відношення "вартість/продуктивність" залишається постійним із ростом числа процесорів.

Для створення таких систем використовується метод структурно-процедурної організації обчислень. Вихідною формою представлення задачі є паралельна форма - інформаційний граф. Вершинам графа ставляться у відповідність операції, що виконуються процесорами, а дугам графа ставляться у відповідність інформаційні зв'язки між процесорами. Паралельна форма представляє повну структуру задачі і виділяє однотипні ділянки обчислень (базові підграфи), далі задача перетвориться з абсолютно-паралельної форми в послідовно-паралельну (структурно-процедурну) форму.

Структурно-процедурна організація обчислень припускає поділ задачі на структурну і процедурну компоненти. При цьому структурний компонент реалізується у вигляді апаратно реалізованих фрагментів обчислень - кадрів, а процедурна - у вигляді послідовної програми, що описує виклики кадрів. Кадр може бути представлений як двовимірний потік даних, що проходить через структурно реалізований підграф задачі. Структурно-процедурна організація обчислень дозволяє будувати модульно-нарощувані обчислювальні системи з масовим паралелізмом і максимальною продуктивністю.

У лабораторії обчислювальних комплексів факультету обчислювальної математики і кібернетики під керівництвом Смелянського В.Г. ведуться дослідження із синтезу та оптимізації архітектур багатопроцесорних обчислювальних систем на основі розроблених формальних моделей функціонування багатопроцесорних обчислювальних систем.

Задача синтезу архітектур обчислювальних систем, у загальному вигляді, може бути поставлена в такий спосіб. Для заданого поводження прикладної програми потрібно синтезувати архітектуру, паралельну програму і вибрати спосіб організації паралельного обчислювального процесу. При цьому повинні оптимізуватися критерії оцінки якості розв’язку і виконуватися обмеження на припустимі розв’язки. Критерії оцінки якості розв’язку, обмеження на

припустимі розв’язки і параметри моделей визначаються при конкретизації задачі синтезу архітектур.

Задача синтезу архітектур обчислювальних систем відноситься до класу задач структурного синтезу. На відміну від задач параметричного синтезу, задачі структурного синтезу в загальному випадку не можуть бути віднесені до класу, що можуть бути формально розв'язані. При розв’язанні задач структурного синтезу приходиться мати справу із початково невизначеними структурними зв'язками, числом і типом компонентів, не метричними характеристиками компонентів, нечіткими і суперечливими критеріями оцінки якості отриманого розв’язку для системи в цілому. Ці особливості задач, роблять їх аналітично нерозв'язними в загальному випадку.

Формальні моделі опису функціонування обчислювальних систем, запропоновані в [17], дозволяють визначити структуру обчислювальних систем і програм як конструктивні об'єкти, що складаються з елементарних компонентів. Задача синтезу архітектур може бути сформульована як дискретна екстремальна багатопараметрична і багатокритеріальна задача з обмеженнями (+): потрібно знайти екстремум цільової функції залежної від моделі програми, архітектури обчислювальної системи і призначень завдань (елементарних компонентів програми) для виконання на визначеному процесорі (компоненті архітектури обчислювальної системи) при заданих обмеженнях.

Найбільш актуальною, серед множини задач синтезу, є задача синтезу однорідної повнозв’язної архітектури, оптимальної по числу процесорів і паралельної програми, що виконується за заданий директивний термін на даній архітектурі. Вона може бути сформульована в такий спосіб:

Для заданих:

- моделі програми;
- директивного терміну її виконання δ_i .

Потрібно визначити:

- h – число процесорів в обчислювальній системі;

- S - призначення завдань, що складають програму, на процесори системи (S - упорядкований список робочих інтервалів процесів, призначених на один процесор).

При цьому повинні виконуватися умови :

1. Час виконання програми не повинен перевищувати директивний термін.
2. Число процесорів повинне бути мінімальним за умови виконання п. 1.

Модель поводження програми являє собою ациклічний орієнтований граф G=(V,U). Вершині v (v=1,…,n), множини V відповідають робочі інтервали завдань, дугам - зв'язки, що визначають взаємодії між робочими інтервалами з множини V. Під робочим інтервалом мається на увазі інтервал часу, протягом якого виконується елементарне завдання. Робочі інтервали завдання безперервні. Чергування робочих інтервалів завдань, призначених на той самий процесор, припустимо, якщо не порушується частковий порядок, заданий U. Відношення U представляється в такий спосіб: якщо вершині v_i передує v_j, то робочий інтервал v_j необхідно виконати перед початком робочого інтервалу v_i. Кожна вершина має свій унікальний номер і мітки: приналежності робочого інтервалу до завдання та обчислювальної складності робочого інтервалу. Обчислювальною складністю робочого інтервалу вважається час його виконання на процесорі.

Архітектура являє собою повний граф, вершинами якого є процесори, дуги – зв'язки між ними.

Як було сказано вище, задача синтезу є аналітично не розв'язною в загальному випадку, у приведеному окремому випадку задача зводиться до відомої постановки описаної в пункті 1.1 і є NP-важкою. Але для реальних обчислювальних систем, у яких на досягнення оптимальної продуктивності обчислювальної системи впливає ряд неврахованих у даній моделі параметрів (швидкість і порядок взаємодії процесорів між собою, з оперативною пам'яттю, а також внутрішня архітектура вхідних в обчислювальну систему процесорів), припустиме одержання наближеного розв’язку, для отримання якого,

використовують евристичні алгоритми, зокрема, нейромережі і генетичні алгоритми [12].

1.3. Високопродуктивні обчислення AWS

Високопродуктивні обчислення відрізняються від повсякденних обчислень швидкістю та потужністю обробки. Система HPC містить ті самі елементи, що й звичайний настільний комп'ютер, але забезпечує величезну обчислювальну потужність. Сьогодні більшість робочих навантажень HPC використовують масово розподілені кластери невеликих машин, а не монолітні суперкомп'ютери.

Кожна машина в кластері HPC , відомому як вузол, зазвичай містить кілька процесорів, кожен з яких має від двох до чотирьох ядер. Невеликий кластер HPC може мати чотири вузли з 16 ядрами, хоча більшість організацій, швидше за все, використовуватимуть кластери, що містять 16-64 вузли та 64-256 ядер.

Amazon Web Services (AWS) надає різноманітні служби, які підтримують сценарії HPC. Серед них Amazon EC2, який зазвичай використовується для виконання масивних паралельних робочих навантажень, а також спеціалізовані служби, як-от AWS ParallelCluster і FSx for Lustre, спеціально створені для середовища HPC.

AWS ParallelCluster — це сервіс, спеціально розроблений для керування високопродуктивними кластерами в хмарі. Він забезпечує просту текстову конфігурацію для моделювання та надання ресурсів HPC у повністю автоматизований спосіб. Можно створювати кластери HPC, використовуючи кілька типів екземплярів EC2, а також керувати надсиланням завдань і плануванням за допомогою AWS Batch або планувальника Slurm з відкритим кодом.

Ціноутворення пропонується без додаткових витрат —платите лише за ресурси AWS, які керують кластерами.

Варіанти використання для HPC:

- робочі навантаження HPC для виробництва, які потребують певних обчислювальних ресурсів, сховищ і мережевих ресурсів

- швидке створення прототипів кластерів HPC без використання спеціальних сценаріїв або складних конфігурацій.

1.4. Приклади використання AWS HPC.

AstraZeneca використовує петабайти даних геномного секвенування для дослідження та розробки ліків. Щоб швидко обробляти дані в масштабі, AstraZeneca використовувала Amazon Web Services (AWS), щоб створити швидке й ефективне рішення для отримання вражаючої інформації про геноміку.

Maxar працював з AWS над створенням HPC-рішення, яке включає чотири ключові технології. Компанія покладається на Amazon Elastic Compute Cloud (Amazon EC2) для високозахищених обчислювальних ресурсів із змінним розміром і можливості конфігурації потужності з мінімальними труднощами. Maxar також використовує мережевий інтерфейс Elastic Fabric Adapter (EFA) для запуску своєї програми за допомогою інтерфейсу апаратного обходу, який прискорює зв'язок між інстанціями. На додаток до розширених обчислень і мереж, програма використовує Amazon FSx for Lustre для прискорення пропускної здатності програми для читання/запису. Maxar також використовує переваги AWS ParallelCluster , інструменту керування кластерами з відкритим кодом, який спрощує розгортання кластерів HPC за допомогою простого текстового файлу, який автоматично моделює та надає ресурси.

1.5. Висновки по першого розділу.

Науково технічний прогрес вимагає розв’язання прикладних задач. У цьому розділі розглянуто цікаву задачу оптимального розбиття множин, особливістю, розв’язання якої є застосування чисельних методів, що далі вимагає використання великих обчислювальних ресурсів.

Далі виникає проблема адаптації обчислювального процесу під конкретну архітектуру, що само по собі є складною науковою задачею.

Існування хмарних сервісів, таких як AWS змінює парадигму високовиробничих обчислень, та надає можливості обійти деякі проблеми пов’язані з побудовою оптимізованого обчислювального процесу та вирішувати задачу за допомогою “прямих” обчислень.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ЗАДАЧІ ОПТИМАЛЬНОГО РОЗМІЩЕННЯ

2.1. Узагальнена задача оптимального розбиття множин

На теперішній час існує велика кількість робіт, в яких досліджуються процеси керування системами з зосередженими параметрами, поведінка яких описується звичайними диференційними рівняннями. Проте актуальним залишаються питання: створення нових підходів до розв’язання задач керування такими системами; застосування цих підходів до оптимізації певних складних систем.

Серед всього розмаїття задач оптимального керування можна виділити задачі, які зводяться до неперервних задач ОРМ і розв’язуються відповідними методами теорії оптимального розбиття. Такі задачі, у свою чергу, можна розбити на декілька класів в залежності від того, які саме параметри є динамічними:

1) задачі оптимального розбиття множин, в яких цільовий функціонал залежить від функцій попиту і вартості, які є фазовими траєкторіями деякої заданої керованої системи;

2) задачі оптимального розбиття множин, в яких швидкість змінення функції попиту в кожній точці множини, що розглядається, залежить від приналежності цієї точки до поточної підмножини. Припускається, що границі між підмножинами можуть змінюватися з часом;

3) задачі оптимального розбиття множин, в яких виконується п.2 і центри підмножин рухаються з часом за деяким заданим законом або підлягають керуванню.

Всі ці класи задач характеризуються залежністю деяких характеристик керованого процесу або самої керуючої функції від розбиття множини. Серед всієї сукупності неперервних динамічних задач ОРМ виділяється частина задач, в яких розбиття завідомо статичне. У випадку, коли границям між підмножинами, що складають розбиття заданої множини, ще й дозволено

змінюватися із часом, такі задачі можна вважати задачами оптимального розбиття множин з керованими границями між підмножинами. У цьому разі, шукане оптимальне розбиття множини є динамічним.

У багатьох задачах економіки, фізики, техніки, медицини тощо виникає необхідність у розгляданні задач оптимального керування системами з зосередженими параметрами. Серед задач економіки варто відзначити такі: однопродуктова динамічна задача розподілу валового продукту з метою розвитку економічного об'єкту (модель Леонтьєва), неперервна задача оптимального розподілу капіталовкладень у галузі на заданому інтервалі планування, задача оптимального розподілу капіталовкладень між галузями; серед задач фізики відзначимо задачу про рух матеріальної точки заданої маси під дією деякої сили на певному інтервалі часу з метою мінімізації витрат пального, перевантажень при маневрах літальних апаратів тощо, в залежності від фізичного змісту, що надається в окремій задачі.

Методи розв'язання різних класів неперервних задач ОРМ засновані на єдиному підході, який полягає у переході від задачі оптимального розбиття множин, у якій невідомі входять до границь інтегрування, до задачі нескінченновимірного математичного програмування за допомогою введення характеристичних функцій підмножин. Розглянемо це на прикладі найпростішої задачі ОРМ – задачі А1 із [27]:

знайти

$$\min_{\overline{\omega}\in P_N(\Omega)} J(\overline{\omega})$$

де

$$J(\overline{\omega}) = \sum_{i=1}^{N} \int_{\Omega_i} (c(x,\tau_i) + a_i)\rho(x)dx,$$

$$P_N(\Omega) = \left\{ \overline{\omega} = (\Omega_1,...,\Omega_N) : \ \bigcup_{i=1}^{N} \Omega_i = \Omega,\ mes(\Omega_i \cap \Omega_j) = 0,\ i \neq j,\ i,j = \overline{1,N} \right\},$$

функції $c(x,\tau_i)$ – дійсні, обмежені, визначені на $\Omega \times \Omega$, вимірні за x при будь-

якому фіксованому τ_i із Ω для всіх $i=\overline{1,N}$; $\rho(x)$ – обмежена, вимірна, невід'ємна на Ω функція; $a_1,...,a_N$ – задані невід'ємні числа.

Введемо характеристичну функцію

$$\lambda_i(x)=\begin{cases}1, & x\in\Omega_i,\\ 0, & x\in\Omega\setminus\Omega_i,\end{cases} \quad i=1,...,N$$

підмножини Ω_i, $i=\overline{1,N}$ і перепишемо задачу у наступному вигляді:

знайти

$$\min_{\lambda(\cdot)\in\Lambda}\int_\Omega\sum_{i=1}^N(c(x,\tau_i)+a_i)\rho(x)\lambda_i(x)dx,$$

де

$$\Lambda=\{\lambda(x)=(\lambda_1(x),\ldots,\lambda_N(x)):\quad \lambda_i(x)=0\vee 1,\ i=\overline{1,N},$$
$$\sum_{i=1}^N\lambda_i(x)=1,\ м.в.\ для\ x\in\Omega\}.$$

Потім здійснюється перехід (обґрунтований у [27]) від задачі з булевими значеннями змінних до наступної задачі:

знайти

$$\min_{\lambda(\cdot)\in\Lambda_1}\int_\Omega\sum_{i=1}^N(c(x,\tau_i)+a_i)\rho(x)\lambda_i(x)dx,$$

де

$$\Lambda_1=\{\lambda(x)=(\lambda_1(x),\ldots,\lambda_N(x)):\quad 0\le\lambda_i(x)\le 1,\ i=\overline{1,N},$$
$$\sum_{i=1}^N\lambda_i(x)=1,\ м.в.\ для\ x\in\Omega\}.$$

Для такої задачі оптимальний розв'язок досягається на вектор-функції $\lambda^*(x)=(\lambda_1^*(x),...,\lambda_N^*(x))$, де

$$\lambda_i^*(x)=\begin{cases}1, & якщо\ c(x,\tau_i)+a_i\le c(x,\tau_k)+a_k,\\ & i\ne k,\ м.в.\ для\ x\in\Omega,\ k=\overline{1,N},\\ 0 & в\ інших\ випадках,\end{cases} \qquad i=\overline{1,N}.$$

Якщо ж постановка неперервних задач оптимального розбиття множин більш складна і має додаткові обмеження, то еквівалентні їм задачі нескінченновимірного програмування шляхом уведення функціоналу Лагранжа зводяться до негладких скінченовимірних задач оптимізації, для розв'язання яких застосовуються методи недиференційовної оптимізації. Останні – це насамперед різні модифікації r-алгоритму Шора – субградієнтні методи з розтягненням простору у напрямку різниці двох послідовних субградієнтів [11-13]. До теперішнього часу r-алгоритми є одним з найбільш ефективних засобів розв'язання задач недиференційовної оптимізації. При мінімізації гладких функцій вони конкурентоспроможні з найбільш вдалими реалізаціями методів спряжених напрямків та методів квазіньютонівського типу.

Розглянемо базову задачу оптимального розбиття множин з розміщенням центрів:

Задача А2. Нехай Ω – обмежена, вимірна за Лебегом множина в n-мірном євклідовом просторі E_n. Позначим через $P_N(\Omega)$ клас всіх можливих розбиттів множини Ω на N підмножин:

$$\hat{P}_N(\Omega) = \left\{\overline{\omega} = (\Omega_1,...,\Omega_N) : \Omega_i \subseteq \Omega, i = \overline{1,N}; \bigcup_{i=1}^{N} \Omega_i = \Omega; \Omega_i \bigcap \Omega_j = \varnothing, i \neq j; i, j = \overline{1,N}\right\}.$$

Треба визначити розбиття $\overline{\omega}^* \in \hat{P}_N(\Omega)$ та набір "центрів" підмножин $\tau^* = (\tau_1, \tau_2,..., \tau_N) \in \Omega^N$, що приводять до мінімального значення функціоналу

$$F(\overline{\omega}, \tau) = \sum_{i=1}^{N} \int_{\Omega_i} \left(c(x, \tau_i) + a_i\right)\rho(x)dx .$$

Тут $c(x,\tau_i)$ – дійсні, обмежені, визначені на Ω×Ω, вимірні по x при довільном фіксованом $\tau_i \in \Omega$ ($\forall$ i=1,..,N) функції; $\rho(x)$ – обмежена, невідіємна, вимірювана на Ω функція; a_i ($\forall$ i=1,..,N) – задані невідіємні значення.

2.2. Прикладна задача оптимального робиття множин

Якщо функції $c(x,\tau_i)$ є тією чи іншою метрикою у просторі E^2, $a_i=0$ ($\forall$ i=1,..,N), неформально завдання А2 можна сформулювати наступним чином: потрібно знайти таке розбиття вихідної множини на задане число підмножин і такі координати центрів цих підмножин, при яких сума завислих відстаней від точок множини до відповідного центру була б мінімальна. У фізичних завданнях мінімізований інтеграл трактується як робота, що здійснюється точкою (фізичним тілом) при переміщенні вздовж траєкторії, що веде від центру до кожної точки підмножини. З економічної точки зору критерій якості завдання А2 є сумарною вартістю переміщення в центр (або назад) всього ресурсу, що знаходиться в кожній точці множини Ω.

Далі дослідемо задачу А2, коли множина є частиною плоскої кривої, що описується залежністю $y = f(x)$ на заданому відрізку. Постановка задачі А2 на кривій для вирішення прикладних задач передбачає, що як аналізовані області може виступати ділянка кривої, заданої як аналітично, так і таблично (остання в процесі вирішення задачі інтерполується). Крім того, з міркувань практики на функцію накладаються обмеження у вигляді безперервності та диференційності.

У загальному вигляді перше завдання (назвемо його завданням A2R) сформулюємо так. Нехай є ділянка деякої плоскої кривої. Потрібно розмістити на ній задану кількість джерел деякого ресурсу і закріпити кожну точку кривої за певним джерелом так, щоб мінімізувати витрати на переміщення всього ресурсу від джерел до відповідних точок кривої по найкоротшому шляху. Функцію вартості в цьому випадку вважатимемо пропорційною радіусу-вектору, що з'єднує точку - джерело ресурсу, з точкою відповідного підмножини, так що $(\tau_i, f(\tau_i))$ – виток ресурсу, з точкою відповідної підмножини $(x, f(x))$, так що: $c_R(x,\tau_i)=((x-\tau_i)^2+(f(x)-f(\tau_i))^2)^{1/2}$ (рис. 1). Очевидно,

координати так званих «центрів» підмножин мають вигляд і, в цілому, визначаються точками $(\tau_i, f(\tau_i))$ та, в цілом, визначаються точками $\tau_i \in [a,b]$.

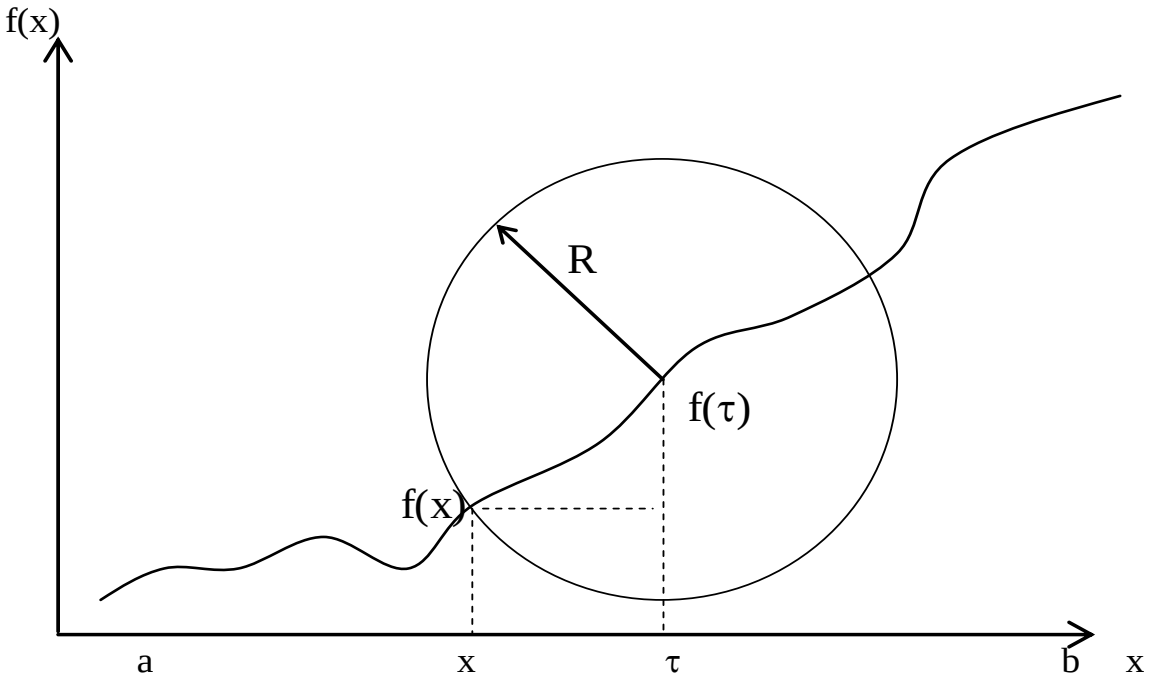


Рисунок 2.1. - Геометрічна інтерпретація подінтегральної функції задачі A2R

Задача A2R. Нехай $\Omega=\{(x,y): a\le x\le b;\ y=f(x)\}$, где $f(x)$ – дійсна, обмежена, диференційована, визначена на [a,b] функція. Необхідно рнайти розбиття $\overline{\omega}^* \in \hat{P}_N(\Omega)$ та набір “центрів” підмножин, що визначаються точками відрізку $[a,b]$ $\tau^* = (\tau_1^*, \tau_2^*, ..., \tau_N^*) \in [a,b]^N$, яке надає мінімальне значення функціоналу

$$F(\overline{\omega},\tau) = \sum_{i=1}^{N} \int_{\Omega_i} ((x-\tau_i)^2+(f(x)-f(\tau_i))^2)^{1/2} dx.$$

Застосуємо методику рішення неперервних задач ОРМ, вводимо характеристичні функції $\lambda_1(\cdot),...,\lambda_N(\cdot)$ підмножин $\Omega_1,...,\Omega_N$, та від задачі A2R переходимо до еквівалентної задачі нескінечномірного програмування:

$$I(\lambda(\cdot),\tau) = \sum_{i=1}^{N} \int_a^b \sqrt{(x-\tau_i)^2 + (f(x)-f(\tau_i))^2}\, \lambda_i(x) dx \to \min_{(\lambda(\cdot),\tau)\in\Lambda_1\times[a,b]^N}, \qquad (2.1)$$

де

$$\Lambda_1 = \{\lambda(x) = (\lambda_1(x),...,\lambda_N(x)) : \sum_{i=1}^{N} \lambda_i(x) = 1 \text{ п.в.для } x \in \Omega;\ \lambda_i(x) = 0 \vee 1; \text{п.в. для } x \in \Omega;\ i = \overline{1,N}\}.$$

Аналітичний вираз для першої компоненти $\lambda^*(\cdot)$ оптимального рішення $(\lambda^*(\cdot), \tau^*)$ задачі (3.1) може бути отримано для кожного фіксованого $\tau = (\tau_1, \tau_2,..., \tau_N)$ у вигляді

$$\lambda^*_i(x) = \begin{cases} 1, \sqrt{(x-\tau_i)^2 + (f(x) - f(\tau_i)^2} = \min\limits_{k=\overline{1,N}} \sqrt{(x-\tau_k)^2 + (f(x) - f(\tau_k)^2} \\ 0, \quad в \quad остальных \quad случаях \end{cases}, \quad i = \overline{1,N}$$

Для пошуку другої компоненти $\tau^* = (\tau_1, \tau_2,..., \tau_N) \in [a,b]^N$ оптимального рішення задачи (3.1) треба розвязати задачу скінчерномірної оптимизації у вигляді

$$G(\tau) = \int_a^b \min_{k=\overline{1,N}} \sqrt{(x-\tau_k)^2 + (f(x) - f(\tau_k)^2}\, \rho(x)dx \to \min_{\tau \in [0,1]^N}, \qquad (2.2)$$

цільова функція якої є у озагальному випадку багатоекстремальною та недифферінційованою. Рішення задачі (3.2) може бути отримано наприклад за допомогою алгоритма А2 [27].

Аналіз властивостей оптимальних розвязків та властивостей мінімізуємой функції $G(\tau)$ для часного випадку задачі A2R, коли *f(x)=Const*, приведен у [27]. Там показано, що задача (3.2) багатоекстремальна. Але на кожной з підмножин множини $[a,b]^N$, які створені заданнням довільних відношень порядка слідувая центрів $\tau_1, \tau_2,..., \tau_N$ на відрізку $[a,b]$: а) функція одноекстремальна та випукла по $\tau_1, \tau_2,..., \tau_N$; б) має точку локального мінімума. Мінімальне значення функції в точках локального мінімума, кожна з котрих належить підмножені множини $[a,b]^N$, що створена відношення порядка слідуваняя центрів $\tau_1, \tau_2,..., \tau_N$ на відрізку $[a,b]$, співпадають.

З практичної точки зору інтерес представляє дослідження і вирішення задачі розбиття ділянки спеціальної кривої, що моделює реальну систему.

2.3. Використання сіткових методів для чисельного розв’язання оптимізаційних задач

В основі розв'язання рівнянь з частинними похідними методом кінцевих різниць лежить різницева апроксимація похідних, яка багато в чому нагадує апроксимацію похідних, як при розв'язанні крайової задачі для звичайних диференціальних рівнянь. Розв'язання здійснюється в три етапи. Спочатку в області розв'язку вводять рівномірну сітку "вузлових точок", що відповідає характеру задачі і граничним умовам. Потім розв'язуване рівняння з частинними похідними записують у найбільш зручній системі координат і, представляючи похідні в кінцеворізницевій формі, приводять його до виду різницевого рівняння. Отримане різницеве рівняння використовують надалі для опису функціонального зв'язку між сусідніми вузлами сітки. Різницеве рівняння записують для всіх вузлів сітки і отримують в результаті систему n алгебраїчних рівнянь з n невідомими. На останньому етапі отриману систему алгебраїчних рівнянь розв'язують одним із чисельних методів.

Ця процедура, яка складається з трьох етапів, може здатися простою і такою, що прямо проводить до розв'язку, однак це не так – широке розмаїття типів і розмірів сіток, видів рівнянь із частинними похідними, можливих кінцево-різницевих апроксимацій цих рівнянь і методів розв'язання отриманих систем алгебраїчних рівнянь роблять задачу чисельного розв'язання рівнянь із частинними похідними виключно багатогранним і цікавим дослідженням.

Етапи чисельного розв'язання диференціальних рівнянь із частинними похідними методом кінцевих різниць Варто розглянути всі три етапи рішення детальніше.

Етап 1. Сітки, що застосовуються при поданні диференціальних рівнянь з частинними похідними в кінцево-різницевій формі. Усі раніше наведені рівняння з частинними похідними були записані в декартовій системі координат, однак іноді буває зручніше користуватися іншими системами

координат, що володіють спеціальними геометричними властивостями і враховують форму фізичного тіла у випадку задач ОРМ форми множини. Найчастіше в інженерній практиці застосовується декартова, циліндрична та сферична системи координат. На рис. 2.5 показані сітки які найчастіше застосовують при розв'язанні рівнянь з частинними похідними – прямокутна, полярна, трикутна та скошена.

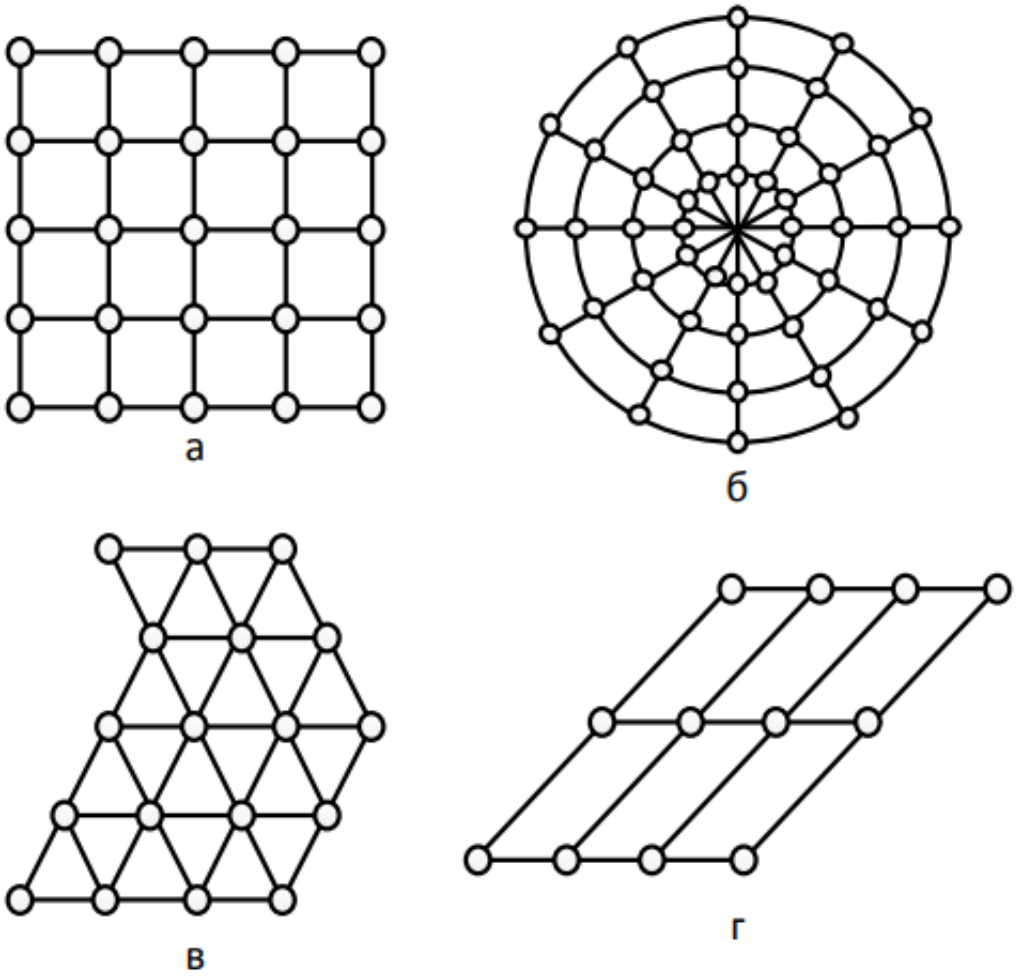


Рис. 2.4. Види сіток, які найчастіше використовуються при чисельному розв'язанні диференціальних рівнянь в частинних похідних: а – прямокутна, б – полярна, в – трикутна, г – скошена.

Етап 2. Подання частинних похідних у кінцево-різницевому вигляді. Для подання частинних похідних у кінцево-різницевому вигляді застосовують формули, аналогічні формулам чисельного диференціювання. Вони тільки

переписуються для випадку декількох змінних. Для двох змінних на практиці найчастіше застосовують симетричні формули

Етап 3. Чисельні методи розв'язання систем лінійних алгебраїчних рівнянь. Для розв'язання систем лінійних алгебраїчних рівнянь на 3-му етапі методу кінцевих різниць застосовують чисельні методи. Зазвичай матриці такої системи виявляються розрідженними, оскільки в більшій частині розрахункових схем застосовуються лише сусідні вузли, а не всі вузли сітки

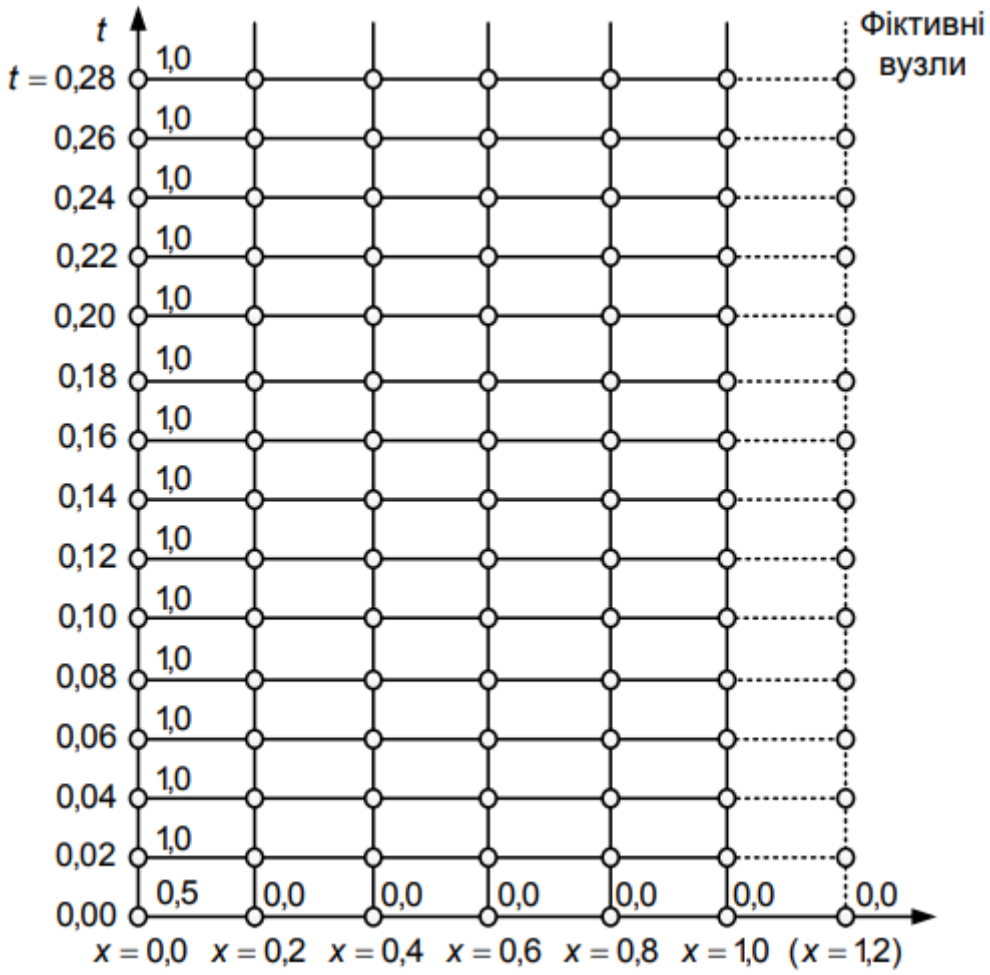


Рисунок 2.5. Приклад сітку, що побудована для твердого стержня, що занурений у твердий матеріал на глубіну L.

2.4. Висновки по розділу

У розділі наведено формальну постановку узагальненої задачі оптимального розбиття множин та наведено концептуальний теоретично обґрунтований підхід до пошуку розв’язку. Також приведена конкретна задача, розв’язання якої має практичну цінність. Відповідно загальна схема розв’язання також може бути застосована для цієї задачі.

Але розв’язання задачі включає необхідність вирішення двох проблем

- визначення множини де гарантовано знаходиться оптимум, що пов’язано з використання методів градієнтного спуску
- перехід від теоретично обґрунтованого апарату неперервних множин до дискретного представлення, яке відповідає реальним параметрам систем та пошуку оптимального чисельного розв’язку.

Ці проблеми вирішуються для кожної конкретної задачі особливим чином.

Але можливо ці проблеми, які вимагають складних обґрунтувань та алгоритмічних та відповідно програмних реалізацій, перекласти на обчислювальну систему. Тобто розуміючи, що є достатньо обчислювальних ресурсів, шукати пошук оптимуму на всієї доступній множині та виконувати неперервно-дискретний перехід на етапі аналізу даних, що отримано з опису предметної області.

Отже складність описаних проблем трансформується у складність обчислювального процесу, що дозволяє спростити алгоритми, відповідно зробити обчислювальну схему прогнозованою та надійною та шукати глобальні оптимуми безпосередньо для всього простору рішень.

РОЗДІЛ 3. ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ AWS ДЛЯ ОРГАНІЗАЦІЇ ОЧИСЛЮВАЛЬНОГО ПРОЦЕСУ

3.1. Високопродуктивні обчислення

Високопродуктивні обчислення - це обчислення, що проводяться на комп'ютерних системах зі специфікаціями, які значно перевищують звичайні комп'ютери. Існують паралельні, розподілені обчислювальны системи, а також їх комбінації.

Паралельні обчислення передбачають розробку програм, які під час їх виконання створюють кілька паралельних і взаємодіючих між собою процесів. Наприклад, моделювання характеристик комірки сонячної батареї передбачає взаємодію трьох моделей, що описують: перенесення носіїв заряду, поширення падаючого світла всередині комірки, температурні ефекти, розтягування-стиснення. Так перенесення носіїв, розтягування-стиснення і показник заломлення матеріалу, який використовується в оптичній моделі падаючого світла, залежать від температури та моделі, що описують ці ефекти, повинні взаємодіяти один з одним у процесі розрахунку. Щоб прискорити розрахунки можна код моделі, що описує транспорт носіїв, виконувати у однієї ноді, код відповідальний поширення світла – іншому, температурну модель – третьому, тощо. Тобто ноди виконуватимуть взаємодіючі розрахунки паралельно.
Розподілені обчислення передбачають використання кількох нодів і процесів, що не взаємодіють один з одним. Дуже часто в такому випадку виконується той самий код на різних нодах. Наприклад, нам потрібно оцінити розтягування та стиснення того ж осередку сонячної батареї залежно від температури. У такому разі температура – вхідний параметр моделі і той самий програмний код цієї моделі можна виконати на різних нодах для різних значень температури.

Вибір між розподіленими та паралельними розрахунками залежить від організації програмного коду, який використовується для розрахунків, самої фізичної моделі, доступності систем обчислень для кінцевого користувача.

Взаємодія користувача із системою високопродуктивних обчислень

Якщо користувач використовує ВР системи віддалено, йому потрібен комп'ютер, на якому він буде:

- готувати програми для запуску на системах ВР;
- запускати розрахунки на системах ВР або підключатися до систем ВР для запуску програм;
- на якому він оброблятиме результати розрахунків.

Тут багато що залежить від особистої переваги користувача, доступності потрібного програмного забезпечення та інших вимог.

Що стосується операційної системи робочого комп'ютера користувача, то здебільшого зручніше всього використовувати ту ж операційну систему і її дистрибутив що встановлена на системі ВР. В даний момент більшість ВР систем працює під управлінням різних дистрибутивів Лінукс .

Якщо вкористовується змішана схему, при якій у вас на комп'ютері встановлена ОС сімейства MS Windows, а система ВВ побудована на ОС Лінукс, то вам знадобиться клієнт для підключення до віддаленої системи (наприклад, PuTTY), і, можливо X-сервер або відразу Cygwin. Аналогічно і щодо графічного інтерфейсу – його використання найчастіше не передбачено та ВВ системи використовуються в текстовому режимі з командного рядка (виключення - той же МАТЛАБ). зберегла результати у файлах чи передала їх користувачу іншим способом.

До продуктивності комп'ютера користувача вимоги мінімальні, оскільки основні розрахунки планується проводити системах ВР. Слідкувати за станом розрахунків, запускати та переривати їх можна з комп'ютера мінімальної конфігурації та з мобільного телефону.

Найчастіше для проведення ВР використовуються суперкомп'ютери, комп'ютерні кластери та гриди (computing grids). Суперкомп'ютери – комп'ютерні системи, що значно перевищують більшість існуючих комп'ютерів за своїми параметрами, такими як продуктивність, доступна оперативна пам'ять, доступна кількість процесорів.

Комп'ютерний кластер – група комп'ютерів, які можуть взаємодіяти один з одним для нарощування доступної пам'яті та кількості процесорів, залучених у роботу. Найчастіше такі кластери будуються усередині дослідницьких груп чи організацій.

Гриди - це групи кластерів і суперкомп'ютерів, розкиданих по різних містах та країнах. Так, наприклад, ви можете передати своє обчислювальне завдання на сервер у Швейцарії, але воно виконуватиметься або на кластерах у Німеччині, Франції чи Польщі. Найбільш відомий приклад ґріда – європейська грід-система EGEE , що об'єднує в собі близько сорока тисяч процесорів та кілька петабайтів дискового простору.

Кінцевому користувачеві часто складно або неможливо розрізнити суперкомп'ютери та кластери.

Кластери HPCondor

Програмне забезпечення для організації такого кластера можна завантажити безкоштовно зі сторінки проекту . Кластери цього типу складаються з робочих комп'ютерів та сервера. Перевага такого кластера в тому, що робочими комп'ютерами можуть виступати звичайні офісні та лабораторні комп'ютери, на яких встановлено клієнтське програмне забезпечення. Вдень ці комп'ютери можуть використовуватися для основної роботи, але як тільки ними перестають користуватися (це залежить від налаштувань) сервер починає запускати на цих комп'ютерах завдання, які були передані йому раніше. Обов'язковою умовою використання цього кластера є встановлення клієнтського програмного забезпечення та на той комп'ютер, з якого користувачі передають завдання. Тобто і їхній комп'ютер має бути частиною кластера.

Для виконання програми, ця програма повинна бути скомпільована в код, що виконується для потрібної ОС і разом з необхідними бібліотеками передана на сервер.

Для запуску програми в кластері, необхідно написати простий конфігураційний скрипт, в якому записані вимоги до середовища виконання

вашої програми (розмір оперативної пам'яті, операційна система тощо) і список файлів, що передаються разом з цією програмою.

Для передачі завдання на сервер кластера, необхідно буде в командному рядку набрати 'condor_submit cost_top.txt'. Після цього ваше завдання буде поставлене в чергу і через деякий час сервер буде готовий запустити ваше завдання на комп'ютерах клієнтів. Час очікування в черзі залежить від пріоритету кожного з користувачів і навантаження на кластер і вибирається системою балансування завдань сервера.

У кластерів цього типу є обмеження:

- з моменту постановки завдання в чергу і до моменту закінчення розрахунку ваш клієнтський комп'ютер повинен бути включений і підключений до локальної мережі, оскільки сервер і клієнт обмінюються файлами;
- даний кластер підтримує лише розподілені ВР завдання;
- є складнощі у використанні будь-якої сторонньої програми (відмінної від написаної та відкопелюваної вами) та програм, що вимагають безлічі бібліотек.

Кластери МАТЛАБ

МАТЛАБ сам собою здатний створити кластер. Для цього вам знадобляться відповідні бібліотеки та сервер — Distributed Computing Toolbox та Distributed Computing Server. Зараз сучасні процесори комп'ютерів мають більш ніж одне ядро і МАТЛАБ здатний розгорнути власний локальний кластер прямо на базі вашого робочого комп'ютера. Така конфігурація кластера відома як локальна конфігурація. Вона зручна в тих випадках коли хочеться трохи прискорити розрахунки без особливих зусиль, як і тоді, коли потрібно протестувати програму перед її стартом на більш серйозній ВО системі, такій як суперкомп'ютер або кластер.

Поряд із локальною конфігурацією існує й інші конфігурації. Наприклад, кластер об'єднує групу комп'ютерів у локальній мережі, групу комп'ютерів у кластері або в гриді. Якщо адміністратори мають можливість і вони не

лінуються, то вони зазвичай налаштовують кластери МАТЛАБ і проводять навчальні курси для того, щоб користувачам було легко користуватися такими кластерами.

Переваги кластерів МАТЛАБ:

- клієнтський комп'ютер, з якого передаються завдання для розрахунку, може бути вимкнений після передачі завдання та користувач може забрати результати розрахунків пізніше;
- можуть виконувати як розподілені, так і паралельні обчислювальні завдання;
- користувачам МАТЛАБ легше почати користуватися такими кластерами, оскільки мова програмування вже знайома;
- програми не потребують компіляції;
- адаптація програми для паралельних розрахунків, у якій вже є оператори циклу 'for', дуже проста – достатньо замінити такий оператор на 'parfor' і додати кілька рядків для ініціалізації кластера та його закриття після закінчення роботи.

Недоліки:

МАТЛАБ - не безкоштовний продукт і частини користувачів він просто не по кишені;

- кластерне ПЗ не поставляється з програмою для балансування навантаження (вона може бути встановлена окремо), що призводить до ситуацій, коли деякі користувачі займають всі ноди кластера і блокують доступ інших користувачів.

Суперкомп'ютери та гриди

Іноді складно знайти різницю між суперкомп'ютером, обчислювальним кластером і гридом. Всі вони мають велику кількість процесорів та пам'яті у ВР системі. Серед установленого програмного забезпечення вони мають компілятори і бібліотеки MPI і OpenMP. Іноді встановлено МАТЛАБ та інші програми, що підтримують використання групи нод та їх пам'яті.

Найчастіше зустрічається алгоритм роботи такий:

- користувач підключається (як правило по SSH) до спеціальних нодів (англ. login nodes), на яких він інтерактивно може виконувати частину команд і з яких може контролювати свої розрахунки;
- завантажує модулі, необхідні для виконання того чи іншого завдання, наприклад компілятор gcc і бібліотеку MPI;
- якщо необхідно, то компілює свою програму за допомогою потрібних бібліотек;
- аналогічно кластеру HPCondor, готує файл налаштувань та команд для виконання своєї програми (англ. job submiossion file);
- передає цей файл налаштувань та команд за допомогою команди 'qsub имя_файла' у чергу виконання;
- як тільки виконання програми буде завершено, користувач може отримати результати виконання (і простіше їх зберігати в файли).

3.2. AWS Lambda

AWS Lambda – це сервіс безсерверних обчислень, який запускає програмний код у відповідь на певні події та відповідає за автоматичне виділення необхідних обчислювальних ресурсів. Перелік подій включає зміни в стані або оновлення, наприклад, коли користувач поміщає товар у кошик на веб-сайті інтернет-комерції. AWS Lambda можна використовувати для розширення можливостей інших сервісів AWS за допомогою спеціальної логіки або для створення власних серверних сервісів із застосуванням можливостей масштабування, продуктивності та безпеки AWS. Сервіс AWS Lambda автоматично запускає програмний код у відповідьрізні події, такі як HTTP-запити черезAPI шлюз Amazon, зміна об'єктів у кошиках. Простий сервіс зберігання даних Amazon (Amazon S3), оновлення таблицьAmazon DynamoDBабо зміна станів уAWS Step Functions.

Lambda запускає код у високопродуктивному обчислювальному середовищі та повністю виконує адміністрування обчислювальних ресурсів. У

тому числі обслуговування сервера та операційної системи, виділення ресурсів та автоматичне масштабування, розгортання виправлень коду та системи безпеки, а також моніторинг коду та ведення журналів. Від вас потрібний лише програмний код.

AWS Lambda дозволяє додавати свій код до ресурсів AWS, наприклад до кошиків сервісу Amazon S3 та таблиць Amazon DynamoDB.

Щоб розпочати роботу з AWS Lambda потрібно створити необхідні функції, завантаживши свій код (або написавши його безпосередньо в консолі Lambda), а також задавши обсяг пам'яті, період очікування та роль Управління ідентифікацією та доступом AWS (IAM). Потім необхідно вказати ресурс AWS, який буде тригером для функції: конкретний кошик сервісу Amazon S3, таблицю Amazon DynamoDB або потік Amazon Kinesis. Зафіксувавши зміну ресурсу, Lambda виконає налаштовану функцію, запустить обчислювальні ресурси, необхідні для обробки запитів, і буде керувати ними.

AWS Lambda можна використовувати для створення нових сервісів додатків, які активуватимуться на вимогу за допомогою інтерфейсу прикладного програмування (API) Lambda або спеціальних адрес API, створених за допомогою Amazon API Gateway, а не на пристрої клієнта, дозволяючи не залежати від користувальницьких операційних систем, знижуючи енергоспоживання на стороні клієнта та спрощуючи встановлення оновлень.

Для роботи з AWS Lambda не потрібно освоювати нові мови, інструменти чи інфраструктуру. Сервіс працює з будь-якими сторонніми бібліотеками, навіть вбудованими. Крім того, можна запакувати будь-який код (платформу, SDK, бібліотеку тощо) як рівень Lambda, щоб використовувати його з різними функціями та керувати ним. Lambda має вбудовану підтримку Java, Go, PowerShell, Node.js, C#, Python та Ruby, а також надає API середовища виконання для створення функцій за допомогою будь-яких інших мов програмування.

Завдяки ретельному управлінню інфраструктурою AWS Lambda код виконується у високопродуктивному, стійкому до відмови, що дозволяє зосередити свої зусилля на розробці різноманітних серверних сервісів. Lambda вирыщує проблеми оновлення серверної операційної системи (ОС), а також проблеми, пов'язані з розширенням існуючих або введенням в експлуатацію нових серверів у міру зростання навантаження. AWS Lambda забезпечує ефективне розгортання коду, виконує завдання адміністрування, технічного обслуговування, виправлення вразливостей системи та забезпечує можливість моніторингу та ведення журналів засобами Amazon CloudWatch .

AWS Lambda підтримує необхідні обсяги обчислювальних ресурсів у кількох зонах доступності (AZ) у кожному з регіонів AWS, захищаючи код від несправностей окремих одиниць обладнання або збоїв у роботі центрів обробки даних (ЦОД). AWS Lambda та функції, що працюють у рамках цього сервісу, забезпечують передбачувану та надійну операційну продуктивність. Сервіс AWS Lambda розроблений для забезпечення високої доступності як самого сервісу, так і функцій, які він виконує. Сервіс працює без планових простоїв та перерв на обслуговування.

AWS Lambda підтримує пакування та розгортання функцій у вигляді образів контейнерів, що спрощує для клієнтів створення програм на основі Lambda за допомогою відомих інструментів, робочих процесів та залежностей для образів контейнерів. До інших переваг роботи з Lambda для клієнтів належать простота експлуатації, автоматичне масштабування з часом запуску менше секунди, висока доступність, модель оплати в міру використання та вбудована інтеграція з більш ніж 200 сервісами AWS та програмами «ПЗ як послуга» (SaaS). Корпоративні клієнти можуть використовувати узгоджений набір інструментів як зі своїми програмами Lambda, так і з контейнерними програмами, спрощуючи необхідне централізоване управління, таке як сканування безпеки та підпис образів.

AWS Lambda викликає код тільки тоді, коли це необхідно, і автоматично масштабує ресурси відповідно до обсягу запитів, що надходять, без будь-якого

ручного налаштування. Кількість запитів, що обробляються, не обмежена. AWS Lambda зазвичай починає запуск коду протягом мілісекунд після того, як сталася подія. Оскільки Lambda масштабується автоматично, продуктивність залишається стабільно високою зі збільшенням частоти подій. Оскільки код виконується без збереження станів, Lambda може створювати необхідну кількість інстансів без довгих процедур розгортання або затримки через налаштування.

Проксі-сервер Amazon RDS реалізує керування пулами підключення для роботи з реляційними базами даних. За допомогою RDS Proxy можна ефективно керувати тисячами одночасних підключень до реляційних баз даних. Це дозволяє легко розробляти високомасштабувані, безпечні та безсерверні програми на основі Lambda, які підключаються до реляційних баз даних. На даний момент RDS Proxy підтримує бази даних MySQL та Aurora. RDS Proxy для безсерверних програм можна настроїти на консолі Amazon RDS або на консолі AWS Lambda.

Provisioned Concurrency дозволяє краще контролювати продуктивність безсерверних програм. Коли цю можливість увімкнено, функції знаходяться в ініціалізованому стані і готові до швидкого реагування в межах ста мілісекунд. Provisioned Concurrency ідеально підходить для роботи з програмами AWS Lambda, які потребують більшого контролю за часом запуску функцій.

Використання Amazon Elastic File System (EFS) для AWS Lambda дозволяє безпечно зчитувати, записувати та зберігати великі обсяги даних із низькою затримкою під час роботи в будь-якому масштабі. Писати код і завантажувати дані в тимчасове сховище для подальшої обробки не потрібно. Це заощаджує час і полегшує код додатків, дозволяючи зосередитися на бізнес-логіці. EFS для Lambda ідеально підходить для низки прикладів використання, включаючи обробку або резервне копіювання великих обсягів даних, а також завантаження великих довідкових файлів або моделей. Система також дозволяє обмінюватись файлами між безсерверними інстансами або контейнерними

програмами і навіть запускати виведення машинного навчання (ML) за допомогою EFS для AWS Lambda.

За допомогою Lambda@Edge AWS Lambda може запускати відповідний код у місцях AWS по всьому світу у відповідь на події Amazon CloudFront , наприклад запити контенту від серверів джерела або відвідувачів або у зворотному напрямку. Це спрощує надання кінцевим клієнтам якіснішого, індивідуально налаштованого контенту з меншою затримкою.

Створюйте робочі процеси AWS Step Functions, щоб координувати виконання багатьох функцій AWS Lambda для вирішення складних і тривалих завдань. Сервіс Step Functions дозволяє визначати робочі процеси, які активують набір функцій Lambda за допомогою послідовних, паралельних, розгалужених списків операцій, у тому числі можливостей обробки помилок. За допомогою Step Functions та Lambda можна створювати структуровані тривалі процеси для додатків та серверів.

Вбудований пакет засобів розробки програмного забезпечення (SDK) AWS Lambda інтегрується з AWS Identity and Access Management (IAM) для забезпечення безпечного доступу інших сервісів AWS до коду. AWS Lambda за замовчуванням запускає ваш код до Amazon Virtual Private Cloud (VPC). За потреби можна налаштувати доступ до ресурсів AWS Lambda за межами власного VPC, щоб використовувати групи користувачів безпеки та мережні списки контролю доступу. Завдяки цьому функція Lambda отримує безпечний доступ до ресурсів VPC. AWS Lambda відповідає вимогам SOC , HIPAA , PCI та ISO.

Підписання коду AWS Lambda дозволяє забезпечити розгортання у функціях Lambda лише незмінного коду від підтверджених розробників. Для цього існують артефакти коду з цифровим підписом. Завдяки цьому можна підвищити швидкість та гнучкість розробки програм навіть у великих групах розробників, забезпечуючи при цьому високі стандарти безпеки.

Оплата за використання AWS Lambda нараховується за стабільну пропускну здатність або час виконання, а не за кількість серверів, що

використовуються. Використовуючи функції Lambda, ви платите лише за виконані запити та час обчислень, необхідний для запуску коду. Рахунок за використання виставляється з точністю до однієї мілісекунди, завдяки чому автоматичне масштабування від кількох запитів на день до тисяч запитів на секунду стає простим та економічним. Коли функція, для якої налаштовано сервіс Provisioned Concurrency, увімкнена та виконується, платита знімається за запити та час виконання.

Для своїх функцій можна задатинеобхідний обсяг пам'яті, і AWS Lambda виділить пропорційну кількість ресурсів ЦПУ, пропускну здатність мережі та дискових операцій читання/запису (I/O).

Розширення AWS Lambda дозволяють легко виконувати інтеграцію зі звичними інструментами для моніторингу, спостереження, забезпечення безпеки та управління. Lambda викликає функцію у середовищі виконання, забезпечуючи безпечне та ізольоване середовище для виконання коду функції. Розширення Lambda запускаються серед виконання Lambda, разом із кодом функції. Розширення Lambda можуть використовувати API телеметрії AWS Lambda для збору докладної діагностичної інформації, такої як журнали, метрики та дані відстеження, безпосередньо з Lambda і відправляти їх у вибране місце призначення. У середовищі виконання Lambda також можна легко інтегрувати агенти безпеки, які мають мінімальний вплив на продуктивність функцій.

Функції AWS Lambda, що працюють на Graviton2 з архітектурою процесора Arm від AWS, забезпечують продуктивність до 34% краще, ніж у функцій на процесорах x86. Graviton2, які зараз доступні в AWS, забезпечують меншу затримку, найбільшу енергоефективність, продуктивність понад 19% та знижену на 20% вартість.

AWS Lambda інтегрується з іншими сервісами AWS , забезпечуючи вбудований моніторинг функцій Lambda. Щоб полегшити моніторинг та усунення неполадок, Lambda автоматично відстежує функції від вашого імені та передає журнали до Журналів Amazon CloudWatch, метрики – до метриків

Amazon CloudWatch та даних трасування – до AWS X-Ray (якщо для цієї функції включено трасування). Сервіс Lambda надає розширені засоби керування веденням журналів, такі як можливість вбудованого збору журналів Lambda у структурованому форматі JSON, керування їх фільтрацією на рівні журналів без внесення змін до коду та налаштування групи журналів Amazon CloudWatch, до якої їх відправляє Lambda. Аналітику Amazon CloudWatch Lambda можна також використовувати, щоб збирати покращені метрики продуктивності та журнали для ваших функцій. Крім того, Lambda дозволяє легко використовувати інструменти моніторингу та спостереження від ваших кращих постачальників інструментів за допомогою розширень Lambda.

3.3. Реалізація технології

Масштабований характер і змінний попит робочих навантажень обчислювальної математики робить їх добре підходящими для середовища хмарних обчислень.

ОРМ — це дослідження характеристик множин, зазвичай у присутності об’єкта. Типові потоки значень, які цікавлять інженерів і вчених, включають дані про параметри, середовищ, таких як населення, ресурси та відстані. Відповідно ОРМ — це дослідження цих даних за допомогою чисельного підходу. ОРМ передбачає вирішення інтегралів (маси, вартості, енергії та інших) у скінченній області.

Зараз доступно багато інструментів для розрахунків, включаючи спеціалізовані та власні інструменти. Це різноманіття є результатом широкої області фізичних та соціальних проблем, які вирішуються за допомогою чисельних методів. Не існує універсального коду для всіх програм, хоча є пакети, які пропонують великі можливості. Широкі можливості доступні в комерційних пакетах, таких як ANSYS Fluent, Siemens Simcenter STAR-CCM+,

Metacomp Technologies CFD++, і пакетах з відкритим кодом, таких як OpenFOAM і SU2.

Типове моделювання включає наступні чотири кроки.

Визначте геометрію . У деяких випадках цей крок є простим, наприклад моделювання на квадраті. В інших випадках цей крок включає складні компоненти та рухомі границі, наприклад, моделювання динамічного процесу. У багатьох випадках створення геометрії займає багато часу. Етап геометрії є інтенсивним для роботи з графікою та вимагає потужної графічної робочої станції, бажано з графічним процесором (GPU). Часто геометрію надає дизайнер, але інженер повинен «очистити» геометрію для введення в розв'язувач потоку або інструмент генерації сітки. Це може бути виснажливим і трудомістким кроком, а іноді може потребувати великого обсягу системної пам'яті (RAM) залежно від складності геометрії.

Створіть сітку або сітку . Створення сітки є критично важливим кроком, оскільки точність обчислень залежить від розміру, розташування комірок, а також асиметрії чи ортогональності комірок. На наступному малюнку гібридна сітка показана на розрізі крила літака. Генерація сітки може бути ітеративною разом із рішенням, де виправлення сітки обумовлені розумінням особливостей та градієнтів у рішенні. Мешінг часто є інтерактивним процесом, і його еліптична природа зазвичай вимагає значного обсягу пам'яті. Як і визначення геометрії, створення однієї сітки може зайняти години, дні, тижні, а іноді й місяці. Багато кодів створення сітки все ще обмежені одним вузлом, тому Amazon Elastic Compute Cloud загального призначення, Часто використовуються екземпляри (Amazon EC2), наприклад сімейство M, або екземпляри, оптимізовані для пам'яті, наприклад сімейство R.

Генерація сітки. Цей крок є основним предметом розробки. Для деяких рішень десятки тисяч ядер (процесорів) працюють протягом тижнів, щоб знайти рішення. І навпаки, деякі завдання можуть бути виконані всього за кілька хвилин за належного масштабування.

Існують варіанти рівнянь моделі залежно від бажаної точності рішення. Наприклад, підвищення точності моделювання турбулентності досягається за допомогою різних наборів наближених рівнянь, таких як середнє Нав'є-Стокса Рейнольдса (RANS), моделювання великих вихрів (LES), моделювання затримки відокремлених вихрів (DDES) і пряме чисельне моделювання (DNS). Ці підходи зазвичай не змінюють фундаментальних обчислювальних характеристик або масштабування моделювання.

Постобробка через візуалізацію — приклади цього кроку включають створення зображень, відео та постобробку. Подібно до етапів геометрії та сітки, це може потребувати великої кількості графіки та пам'яті, тому часто перевагу надають графічним процесорам (GPU).

Відповідно до задачі, застосуємо сервіс AWS Lambda.

Консоль керування AWS, у верхній панелі навігації Lambda відкриваємо AWS Lambda Console.

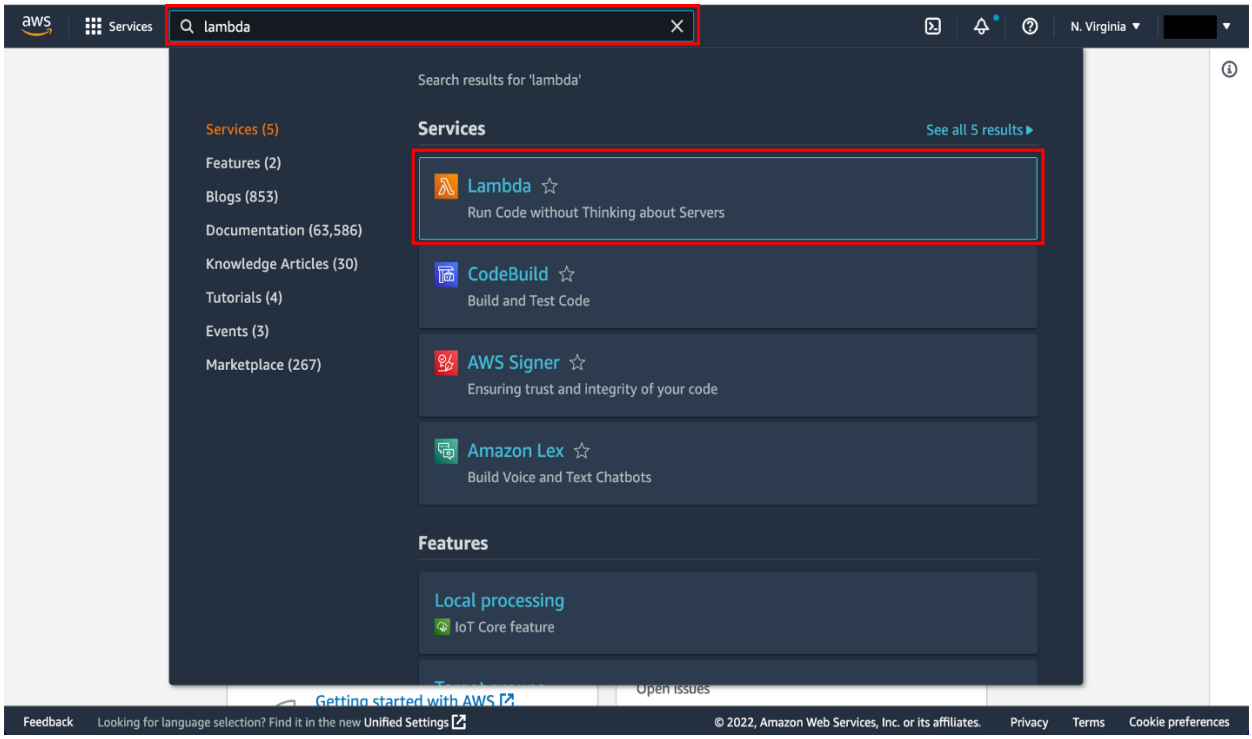


Рисунок 3.1. - Консоль керування AWS

Скріни надають приклад коду для мінімальної обробки. Більшість завдань обробляє події з певних джерел подій, наприклад Amazon S3, Amazon DynamoDB або спеціальної програми.

У консолі AWS Lambda обираємо «Створити функцію» .

Консоль показує цю сторінку, лише якщо немає створених лямбда-функцій. Якщо вже створили функції то Лямбда > Функції . На сторінці списку обираємо «Створити функцію» .

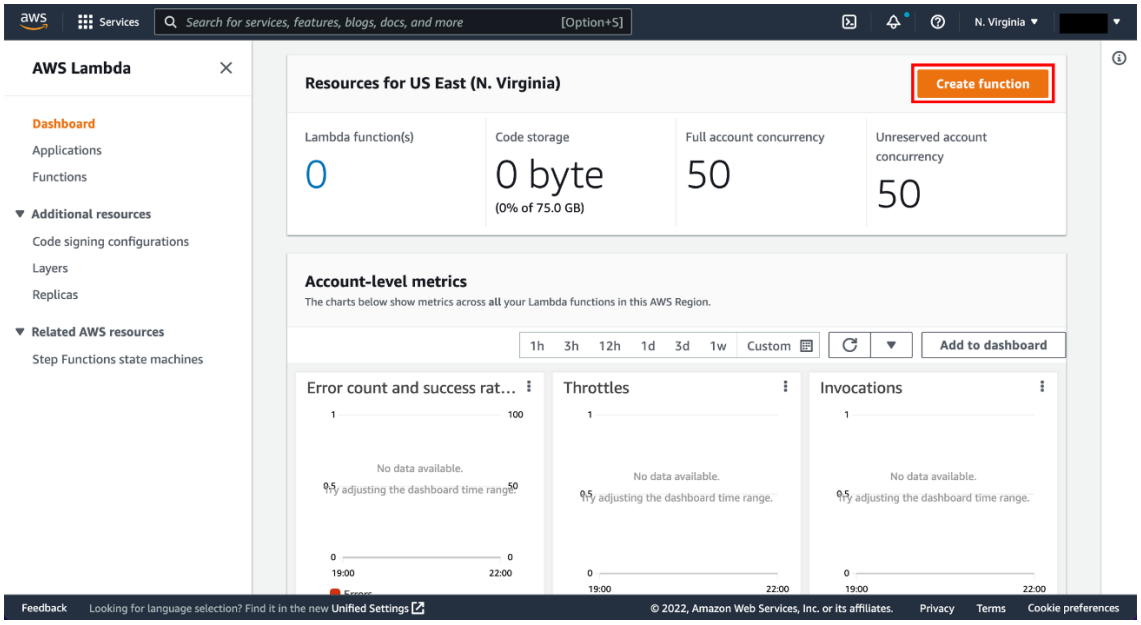


Рисунок 3.2. - Консоль AWS Lambda

Функція Lambda складається з наданого коду, пов’язаних залежностей і конфігурації. Інформація про конфігурацію, включає обчислювальні ресурси, (наприклад, пам’ять), час очікування виконання та роль IAM, яку AWS Lambda може взяти на себе для виконання функції Lambda.

Основна інформація:

Назва: тут назваємо лямбда-функцію.

Роль: створюємо роль IAM (називається роллю виконання) з необхідними дозволами, які AWS Lambda може отримати для виклику функції Lambda від вашого імені.

Ім'я ролі: lambda_ORM_execution.

Код функції лямбда:

Розміщено у додатку.

Щоб продовжити створення функції:

Обираємо використання.

У полі «Назви проекту» переконайтеся, що вибрано функцію «ORM»

У полі імені функції ввjlbvj ORM.

Для ролі виконання обираємо Створити нову роль із шаблонів політики AWS.

У полі Ім’я ролі вводимо lambda_ORM_execution.

Натискаємо кнопку Створити функцію .

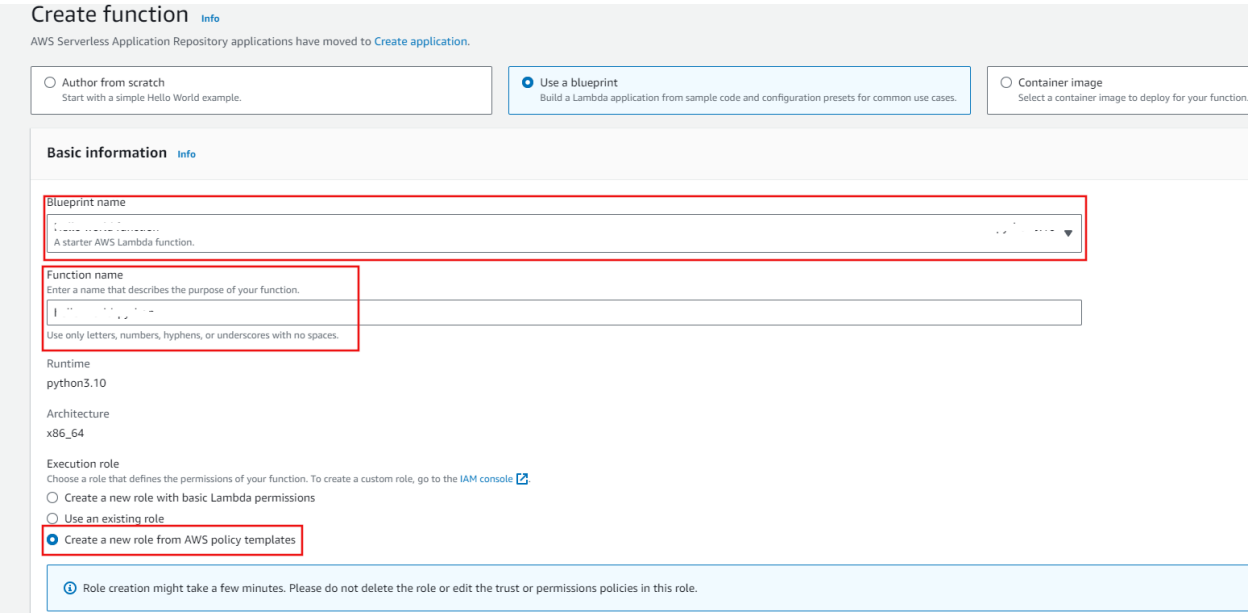


Рисунок 3.3. - Створення функції

Налаштування Lambda.

Середа виконання: наразі можна створювати код функції Lambda на Java, Node.js, C#, Go або Python. Для цьєї роботи використовуємо C# як середовище виконання.

Обробник: можна вказати обробник (метод/функцію у вашому коді), за допомогою якого AWS Lambda може розпочати виконання вашого коду. AWS Lambda надає дані про події як вхідні дані для цього обробника, який обробляє подію.

Викликаємо функцію лямбда та перевіряємо результати. Відкриваємо редактор, у якому вводимо подію, щоб перевірити свою функцію.

Обираємо Створити нову подію.

Ввводимо назву події Test .

Зберігаємо налаштування за замовчуванням «Приватно» для налаштувань спільного доступу до подій.

Виберіть функцію зі списку шаблонів.

Після успішного виконання переглянемо результати в консолі:

Вкладка «Результати виконання» підтверджує, що виконання виконано успішно.

У розділі «Журнали функцій» відображатимуться журнали, згенеровані виконанням функції Lambda, а також ключова інформація, яка надсилається у вихідні дані журналу.

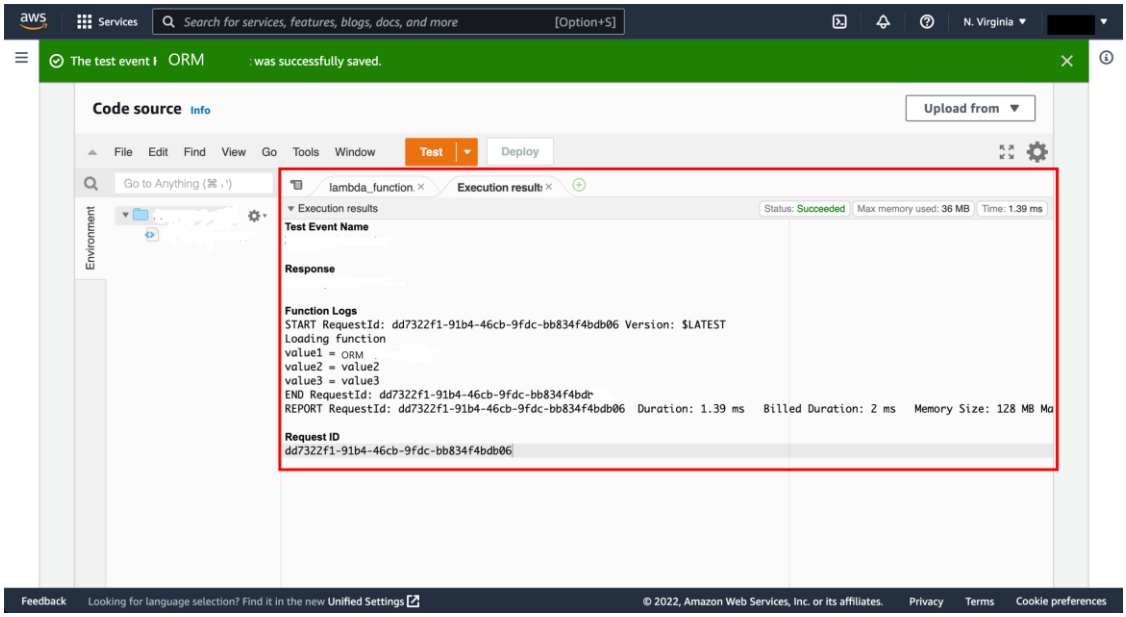


Рисунок 3.5. - Результат роботи сервісу

AWS Lambda автоматично відстежує функції Lambda та звітує про показники через Amazon CloudWatch. Щоб допомогти відстежувати код під час його виконання, Lambda автоматично відстежує кількість запитів, затримку на

запит і кількість запитів, що призвели до помилки, і публікує відповідні показники.

Викликаємо функцію Lambda ще кілька разів, кілька разів натиснувши кнопку Test . Це створить показники, які можна переглянути на наступному кроці.

Обераємо вкладку «Монітор», щоб переглянути результати.

На вкладці «Моніторинг» відображатимуться сім показників CloudWatch: *виклики, тривалість, кількість помилок і рівень успіху (%), дроселі, збої асинхронної доставки, IteratorAge та одночасне виконання.*

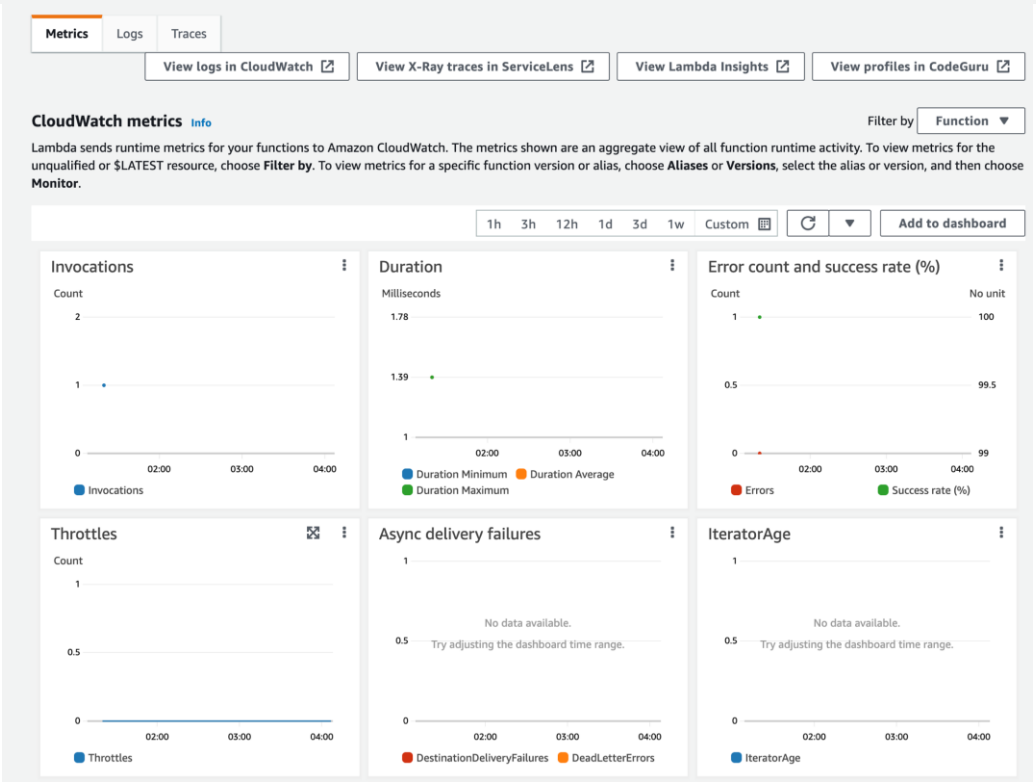


Рисунок 3.6. - Метрики роботи функції.

Таким чином функція готова для проведення обчислювального експерименту.

Більш того за рахунок параметризації обчислювального процесу, можна отримати інформацію про обчислювальні затрати та хід обчислень, що цікаво для того, щоб зробити висновки про ефективність запропонованого підходу вцілому.

Ще один суттєвий момент, це розуміння витрат, AWS це комерційна система, використання функцій якої вимагає відповідних коштів.

3.4. Тестування рівня функції

У зв'язку з використанням обчислювального середовища AWS, який бере на себе всі нюанси функціювання додатку як такого виникає необхідність проведення тільки тестування логіки функціювання реалізованій у додатку. Стандартний спосіб тестування моделей, це розв'язання спрощеної задачі з відомим результатом за допомогою розробленої обчислювальної схеми.

Для задачі оптимального розбиття множин спрощеним аналогом є задача побудови діаграм Вороного.

Отже спрощуємо задачу та отримуємо у якості результату розбиття стандартної множини на багатогранники та порівнюємо з відомими майже класичними результатами.

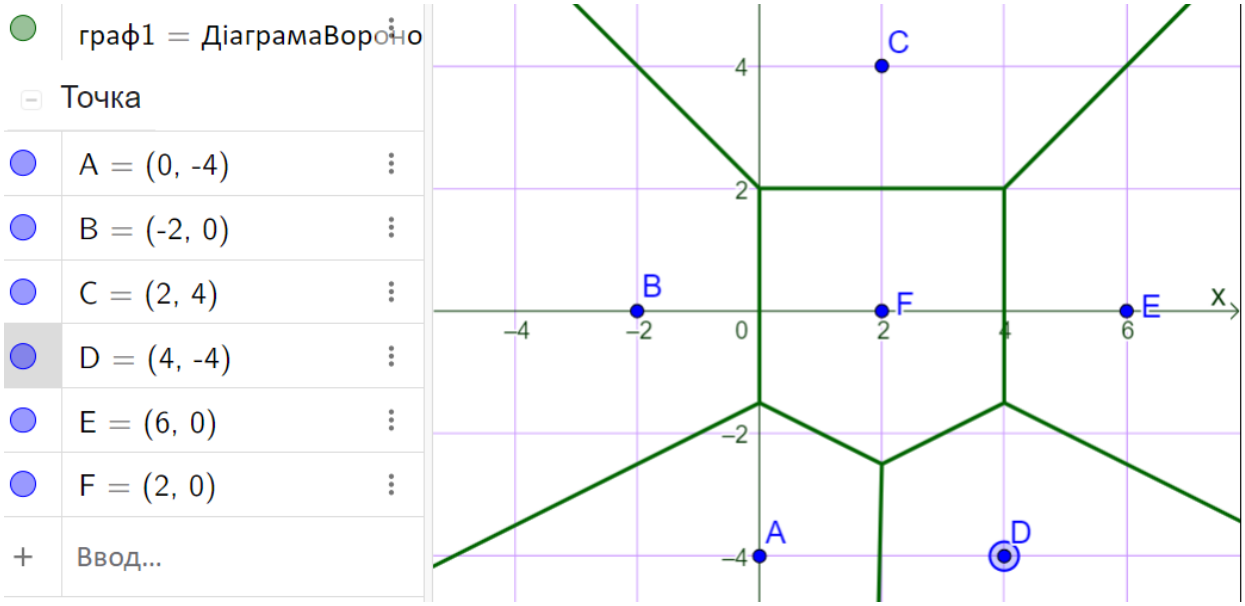


Рисунок 3.7. -Діаграма Вороного побудована для шести точок.

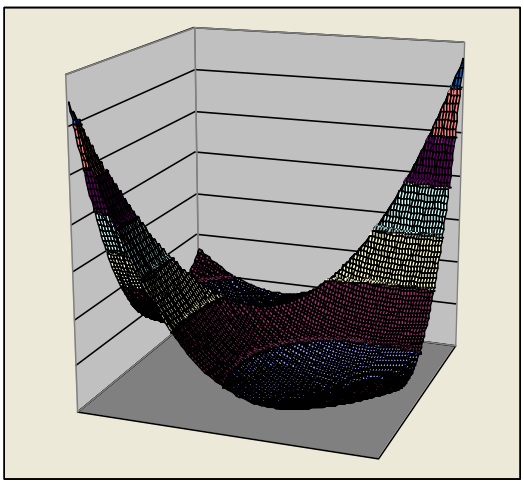
Аналогічний результат отримаємо при реалізації обчислювальної процедури у спрощеному вигляді.

Тобто реалізація відповідає вимогам до запланованого обчислювального експерименту і можна переходити до тестування вже безпосередньо в рамках розв’язання основної задачі.

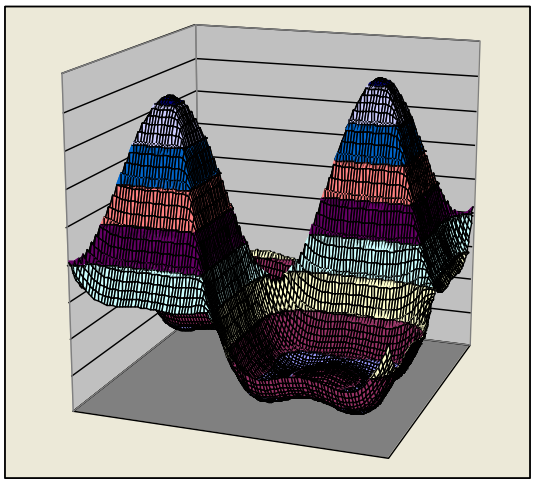
РОЗДІЛ 4. ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ ТА ДОСЛІДЖЕННЯ

Як було зазначено вище, з практичної точки зору інтерес представляє дослідження та вирішення завдання розбиття ділянки спеціальної кривої, що моделює реальну систему. Наприклад, коли йдеться про будівництво траси, виникає проблема оптимального розміщення складів, робочих містечок, асфальтових заводів уздовж траси, що будується так, щоб витрати на транспортування трудових або матеріальних ресурсів уздовж об'єкта, що будується, були мінімальні. Тому для обчислювальних експериментів було обрано криві види $f(x)=x^2$, $f(x)=x^3$, $f(x)=ln(x)$, $f(x)=sin(x/\pi)$, $f(x) = Asint(\omega x)$, $A, t, \omega \in Z$. Остання представляє особливий інтерес, оскільки відомо, що будь-яка безперервна функція на інтервалі $[0,2\pi]$ може бути представлена у вигляді тригонометричного ряду. Отже, властивості оптимальних розв'язків задачі A2R для будь-яких безперервних функцій, а також застосування вище зазначеного алгоритму, будуть визначатися властивостями задач A2R з функціями виду $f(x)=Asint(\omega x)$при різних параметрах з точністю розкладання у тригонометричний ряд.

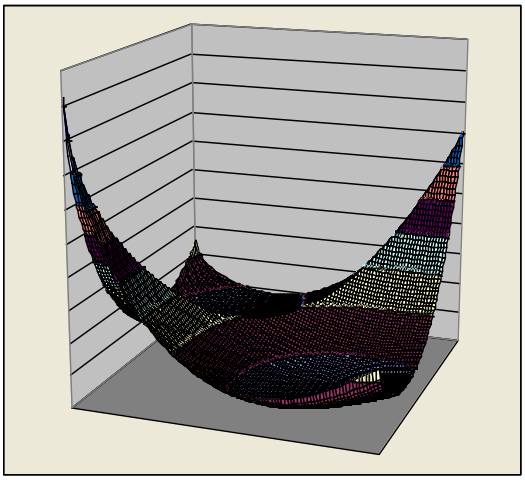
На рисунку 4.1 наведено приклади поверхонь мінімізованих функцій $G(\tau_1, \tau_2)$, до яких зводяться завдання A2R при $N = 2, [a,b]=[0,1]$ (тільки для двох центрів можна візуалізувати поверхню, що задається функцією, що мінімізується), отриманих за результатами розрахунків за допомогою технології AWS Lambda, опис та реалізація якої наведено у розділі 3. Як видно з наведених рисунків, задачу A2R можна умовно розбити на два класи. До першого відносяться задачі, в яких цільова функція $G(\tau_1, \tau_2)$ має два мінімуми, симетричні щодо діагоналі квадрата, відповідного області Ωі що починається в точці (0; 0). Причому значення функції у точках мінімуму збігаються [1]. Що можна тратувати як проходження тесту на виконання функції зі здалегіть відомим результатом. Зауважимо, що до цього класу задач належать ті, у яких як f(x) виступає функція, монотонна на $[a,b]$.



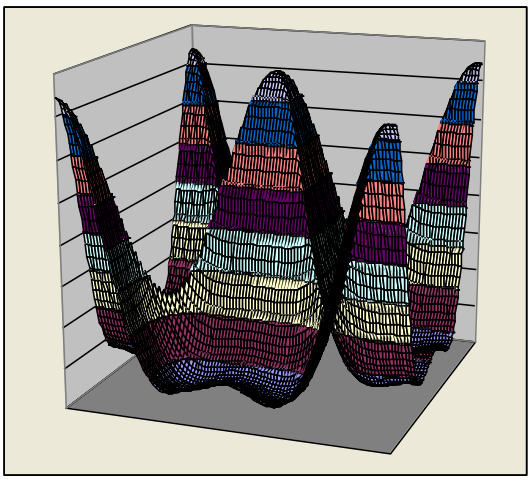
а)



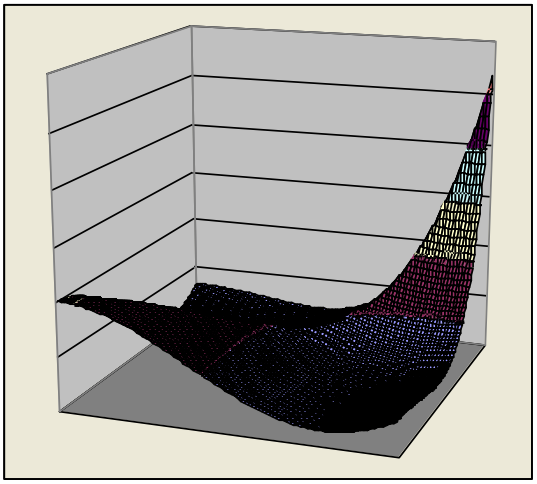
г)



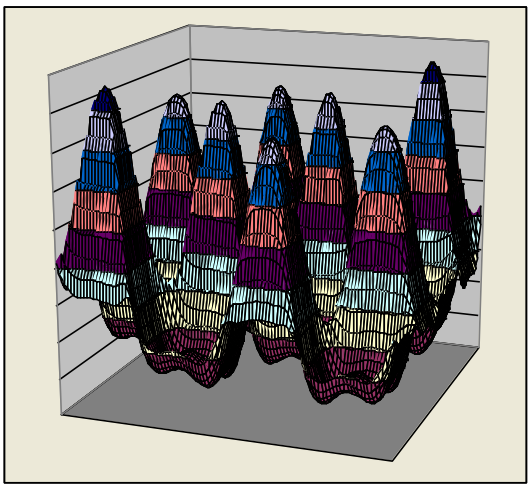
б)



д)



в)



е)

Рисунок.4.1. - Поверхні, побудовані за значеннями функції$G(\tau_1,\tau_2)$.Розбивається ділянка кривої а) f(x)=const, б) f(x)=ln(x), в) f(x)=x2, г) f(x)=sin($\pi x/2$), д) f(x)=sin($2\pi x$), е) f(x)=sin($4\pi x$), x∈[0,1]. Значення F нормовані.

Задача A2R застосовується на практиці, коли з центру до кожної точки області можна дістатися уздовж відповідного радіусу вектора.

Уточнимо модель і сформулюємо наступне завдання оптимального розбиття плоскою кривою (назвемо її завданням A2L). Введемо додаткові обмеження на траєкторію пересування. Очевидно, що у разі роботи з транспортними комунікаціями, пересування ресурсів здійснюватиметься лише з них. Наприклад, підвезення матеріалів на дорозі, що будується, швидше за все, здійснюється по її готових ділянках.

Загалом задачу сформулюємо так. Необхідно мінімізувати витрати на переміщення всього ресурсу до кожної точки вздовж кривої на заданій її ділянці із заздалегідь відомого числа джерел уздовж цієї кривої. При цьому потрібно знайти оптимальне розташування джерел.

Завдання A2L. НехайΩ= {(x, y): a$\le x \le b$; $y=f(x)$}, де f(x) – дійсна, обмежена, диференційована, визначена на$[a,b]$функція. Необхідно знайтирозбиття$\overline{\omega}^* \in \hat{P}_N(\Omega)$та набір “центрів” підмножин, що визначаються точками відрізка$[a,b]$ $\tau^* = (\tau_1^*, \tau_2^*, ..., \tau_N^*) \in [a,b]^N$, що доставляють мінімальне значення функціоналу

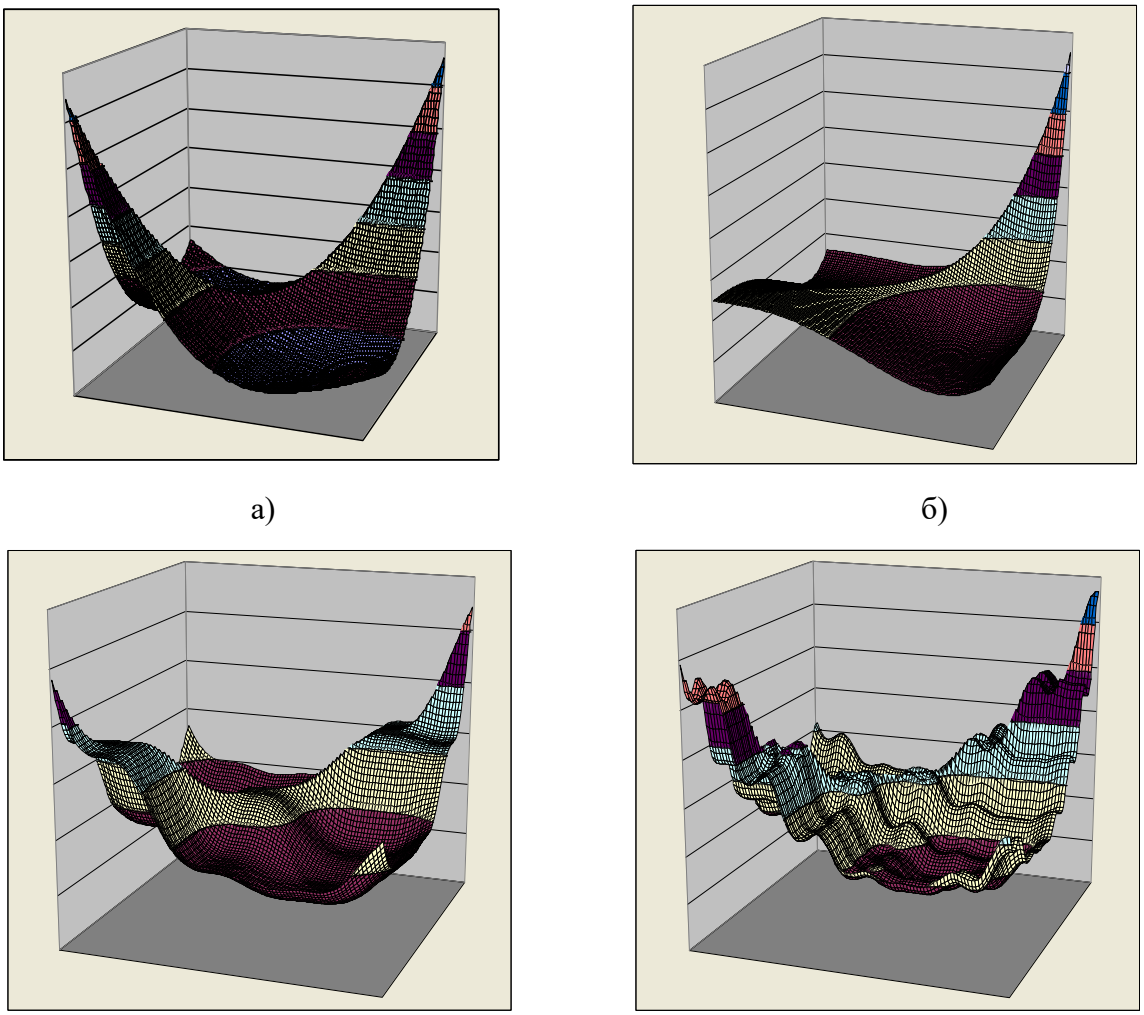
$$F(\overline{\omega}, \tau) = \sum_{i=1}^{N} \int_{\Omega_i} \left| \int_{\tau_i}^{x} \sqrt{1 + f'^2(\xi)}\,\rho(x)\,d\xi \right| dx,$$

де$\int_{\tau_i}^{x} \sqrt{1 + f'^2(\xi)}\,d\xi$- Довжина дуги кривої від центруτ_iдо точки x,$\rho(x)$– задана дійсна, обмежена на$[a,b]$функція (далі без втрати спільності вважатимемо$\rho(x)$ =1).

Очевидно, що для$f(x)$=Const завдання A2R та A2L збігаються.

Залежно від вибору функції f(x) цільова функція для завдання A2L буде набувати конкретного вигляду. Наприклад, якщо f(x)=x2, то c(x, τ_i)=/ $\tau_i \sqrt{1+\tau_i^2}$ /2+ln $|\tau_i + \sqrt{1+\tau_i^2}|$ /2–x $\sqrt{1+x^2}$ /2+ln $|x + \sqrt{1+x^2}|$ /2| ,відповідно для f(x)=ln(sin(x)) отримуємо c(x, τ_i)=| $ln(tg(x/2)/tg(\tau_i/2))$| . Приклади поверхонь, що утворюються

значеннями функції F, побудованої для множини Ω= {(x, y): $0.02 \leq x \leq 1$; $y=f(x)$},наведено на рисунку.



а) б)

в) г)

Рисунок4.2 Поверхня цільової функцій F задачі A2L, яка залежить від$\tau_1, \tau_2 \in [0,1]$. Розбивається ділянка кривої а) f (x) = const, б) f (x) = x, в) f (x) = sin (x), г) f (x) = sin (2x)

Для завдання A2L з розміщенням двох центрів на інтервалі [0;1] рішення можна отримати в аналітичному вигляді, попередньо припустивши, що $f(x)$ - монотонна функція, $\tau_1 \leq \tau_2$. Розкриємо модуль і перепишемо функціонал задачі A2L у вигляді:

$$\hat{F}(p,\tau_1,\tau_2)=\int_0^{\tau_1}\int_x^{\tau_1}\sqrt{1+f'^2(\xi)}\,d\xi dx+\int_{\tau_1}^{p}\int_{\tau_1}^{x}\sqrt{1+f'^2(\xi)}\,d\xi dx+\int_{p}^{\tau_2}\int_{x}^{\tau_2}\sqrt{1+f'^2(\xi)}\,d\xi dx+$$

$$+\int_{\tau_2}^{1}\int_{\tau_2}^{x}\sqrt{1+f'^2(\xi)}\,d\xi dx.$$

Випишемо для функції $\hat{F}(p,\tau_1,\tau_2)$ необхідні умови безумовного екстремуму (обчислення приватних похідних функції $\hat{F}(p,\tau_1,\tau_2)$ здійснюється за допомогою формули Лейбниця диференціювання інтеграла за параметром):

$$\begin{cases}\frac{dF}{d\tau_1}=\int_0^{\tau_1}\sqrt{1+f'^2(\xi)}\,d\xi-\int_{\tau_1}^{p}\sqrt{1+f'^2(\xi)}\,d\xi=0;\\ \frac{dF}{d\tau_2}=\int_p^{\tau_2}\sqrt{1+f'^2(\xi)}\,d\xi-\int_{\tau_2}^{1}\sqrt{1+f'^2(\xi)}\,d\xi=0;\\ \frac{dF}{dp}=\int_{\tau_1}^{p}\sqrt{1+f'^2(\xi)}\,d\xi-\int_{p}^{\tau_2}\sqrt{1+f'^2(\xi)}\,d\xi=0.\end{cases} \quad (3.1)$$

Серед стаціонарних точок виберемо ті, які задовольнятимуть умовам:

$$0\le\tau_1\le p\le\tau_2\le 1.$$

Аналізуючи систему (3.1), можна дійти невтішного висновку у тому, що точка (*p, f(p)*) – кордон між областями Ω_1 і Ω_2 – лежить на кривій $y=f(x)$ так, що відстані від цієї точки вздовж кривої до точок (τ_1, $f(\tau_1)$) і (τ_2, $f(\tau_2)$) однакові. Крапка (τ_1, $f(\tau_1)$) знаходиться на середині відстані вздовж кривої між (0,f(0)) та (p, f(p)), а (τ_2, $f(\tau_2)$) – на середині відстані вздовж кривої між (p, f(p)) та (1,f(1)).

У разі ж коли *f(x)* -періодична, або кількість центрів, що розміщуються більше двох,аналітичне розв'язання задачі одержати практично неможливо, т.к. у цих випадкахЗавдання A2L - багатоекстремальна (про що свідчать поверхні цільових функцій задачі, наведені на рисунку 4.2 в), г)).

Далі уточнимо модель, для оптимізації якої вирішується завдання A2L. Досі передбачалося, що витрати пропорційні довжині траєкторії між центром та кожною точкою на ній. Тепер зважимо на той факт, що рух уздовж кривої ускладнюється її формою, а точніше кривизною, що викликає додаткові

витрати, тому до повної вартості додамо витрати, пропорційні показнику кривизни. Розглянемо два випадки.

1. Врахуємо вплив кривизни на вартість переміщення через кожну проміжну точку, тоді сумарна вартість визначиться виваженою сумою ($\beta_1,\beta_2 \geq 0, \beta_1+\beta_2 \neq 0$):

$$c(x,\tau_i)=\beta_1 \left| \int_{\tau_i}^{x} \sqrt{1+f'^2(x)}\,dx \right| + \beta_2 \left| \int_{\tau_i}^{x} f''(x) \;/\; (\sqrt{1+f'^2(x)})^3\,dx \right|, \; i=1,..,n,$$

де $f''(x)/(\sqrt{1+f'^2(x)})^3$ - кривизна кривої у точці x.

Завдання А2(L+K). Нехай $\Omega = \{(x, y): a \leq x \leq b;\ y=f(x)\}$, де f(x) – дійсна, обмежена, диференційована, визначена на $[a,b]$ функція. Необхідно знайти розбиття $\bar{\omega}^* \in \hat{P}_N(\Omega)$ та набір "центрів" підмножин, що визначаються точками відрізка $[a,b]$ $\tau^* = (\tau_1^*, \tau_2^*, ..., \tau_N^*) \in [a,b]^N$, що доставляють мінімальне значення функціоналу

$$F(\bar{\omega},\tau) = \sum_{i=1}^{N} \int_{\Omega_i} \left(\left| \int_{\tau_i}^{x} \sqrt{1+f'^2(\xi)}\,d\xi \right| + \left| \int_{\tau_i}^{x} f''(\xi)/(\sqrt{1+f'^2(\xi)})^3\,d\xi \right| \right) dx,$$

Як і в попередньому випадку для кожної функції $f(x)$ отримуватимемо конкретну функцію $c(x, \tau_i)$.

Окремо проінтегруємо другий доданок:

$$K(\tau_i,x) = \int_{\tau_i}^{x} f''(x)/(\sqrt{1+f'^2(x)})^3\,dx = f'(\tau_i)/\sqrt{1+f'^2(\tau_i)} - f'(x)/\sqrt{1+f'^2(x)};$$

тоді:

$$F(\bar{\omega},\tau) = \sum_{i=1}^{N} \int_{\Omega_i} \left(\left| \int_{\tau_i}^{x} \sqrt{1+f'^2(\xi)}\,d\xi \right| + \left| f'(\tau_i)/\sqrt{1+f'^2(\tau_i)} - f'(x)/\sqrt{1+f'^2(x)} \right| \right) dx.$$

Як приклад розглянемо вид цільової функції завдання з двома центрами, коли $f(x)=x2/2,\ x \in [0,1]$.

Інтеграл довжини кривої у разі:

$$L(\tau_i,x) = \int_{\tau_i}^{x} \sqrt{1+f'^2(\xi)}\,d\xi = \tau_i \sqrt{1+\tau_i^2}/2 + \ln\left|\tau_i + \sqrt{1+\tau_i^2}\right|/2 - x\sqrt{1+x^2}/2 + \ln\left|x+\sqrt{1+x^2}\right|/2;$$

тоді функція набуде вигляду:

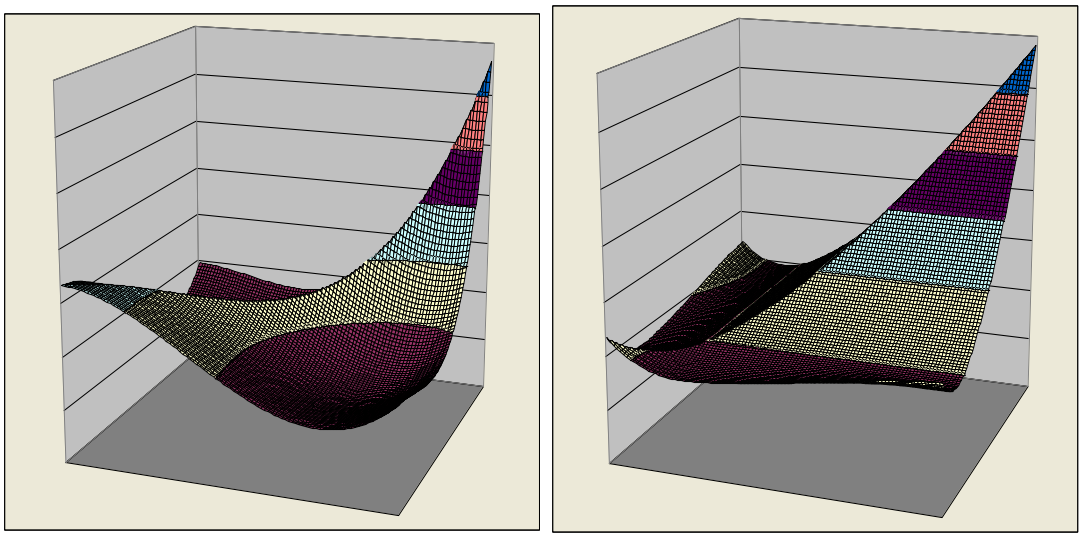
$$F=\int_0^{\tau_1} (L(x,\tau 1)+K(x,\tau 1))dx+\int_{\tau_1}^{p} (L(\tau 1,\, x)+K(\tau 1,\, x))dx+\int_{p}^{\tau_2} (L(x,\tau 2)+ K(x,\tau 2))dx+\int_{\tau_2}^{1} (L(\tau 2,\, x)+$$

$$K(\tau 2,\, x))\, dx.$$

Для випадку, коли f(x)=const, f(x)=x друга похідна дорівнює нулю, тому завдання зводиться до завдання A2R або A2L. Приклади поверхонь, що утворюються функцією F для задачі з двома центрами при різних видах функцій f(x), наведено на рис.4.

Для f(x)=x2 на інтервалі [0;1] кривизна монотонно зменшується, а збільшення функції на одиницю довжини монотонно зростає. Функція F досягає мінімуму в точці$\tau^{(1)}$=(0.03, 0.24) та симетричною їй щодо прямої $y = x$ точці$\tau^{(2)}$= (0.24, 0.03). Кордон між підмножинами відповідає значенню p = 0.1215.

Ця задача може бути узагальнена на випадок кривої лежачої на поверхні (модель ділянки траси).



а) б)

Рисунок 4.3. Поверхня цільової функцій F задачі A2(L+K), яка залежить від$\tau_1,\tau_2 \in [0,1]$. Розбивається ділянка кривої а) f (x) = x2, б) f (x) = sin (x), $x \in [0,1]$

2. Припустимо, що витрати на доставку пропорційні довжині кривої та кривизні в кожній точці (кривизна виступає в ролі функції густини):

$$c(x,\tau_i)= \left| \int_{\tau_i}^{x} \sqrt{1+f'^2(x)} \cdot f''(x)/(\sqrt{1+f'^2(x)})^3 \, dx \right|, \; i=1,..,n.$$

Завдання А2(L·До).НехайΩ= {(x, y): a$\le x \le b$; $y=f(x)$}, де f(x) – дійсна, обмежена, диференційована, визначена на$[a,b]$функція. Необхідно знайтипару $(\bar{\omega}^*,\tau) \in \hat{P}_N(\Omega) \times [a,b]^N$, що доставляє мінімальне значення функціоналу

$$F(\bar{\omega},\tau) = \sum_{i=1}^{N} \int_{\Omega_i} \left| \int_{\tau_i}^{x} \sqrt{1+f'^2(x)} \cdot f''(x)/(\sqrt{1+f'^2(x)})^3 \, dx \right| d\xi.$$

Проінтегруємопідмодульний вираз:

$\int_{\tau_i}^{x} \sqrt{1+f'^2(x)} \cdot f''(x)/(\sqrt{1+f'^2(x)})^3 \, dx$= arctg($f'^2(\tau_i)$) - arctg ($f'^2(x)$); тоді:

$$F(\{\Omega_1,..,\Omega_N\}) = \sum_{i=1}^{N} \int_{\Omega_i} \left| \text{arctg}(f'^2(\tau_i)) - \text{arctg} (f'^2(x)) \right| dx.$$

Задача вироджується, якщо f(x) = const, f (x) = x (рис. 4.4, а). Загалом схема рішення залишається незмінною. Приклади поверхонь, що утворюються цільовими функціями задачі А2(L·К) для задачі з двома центрами за різних f(x), наведені на рис.4.4 б)-г). Неважко помітити, що властивості цільових функцій для деяких завдань А2(L·До) збігаються з властивостями цільових функцій завдань А2(L+К), А2L (при тих самих завданнях f(x)). Тому й алгоритми вирішення цих завдань одні й самі.

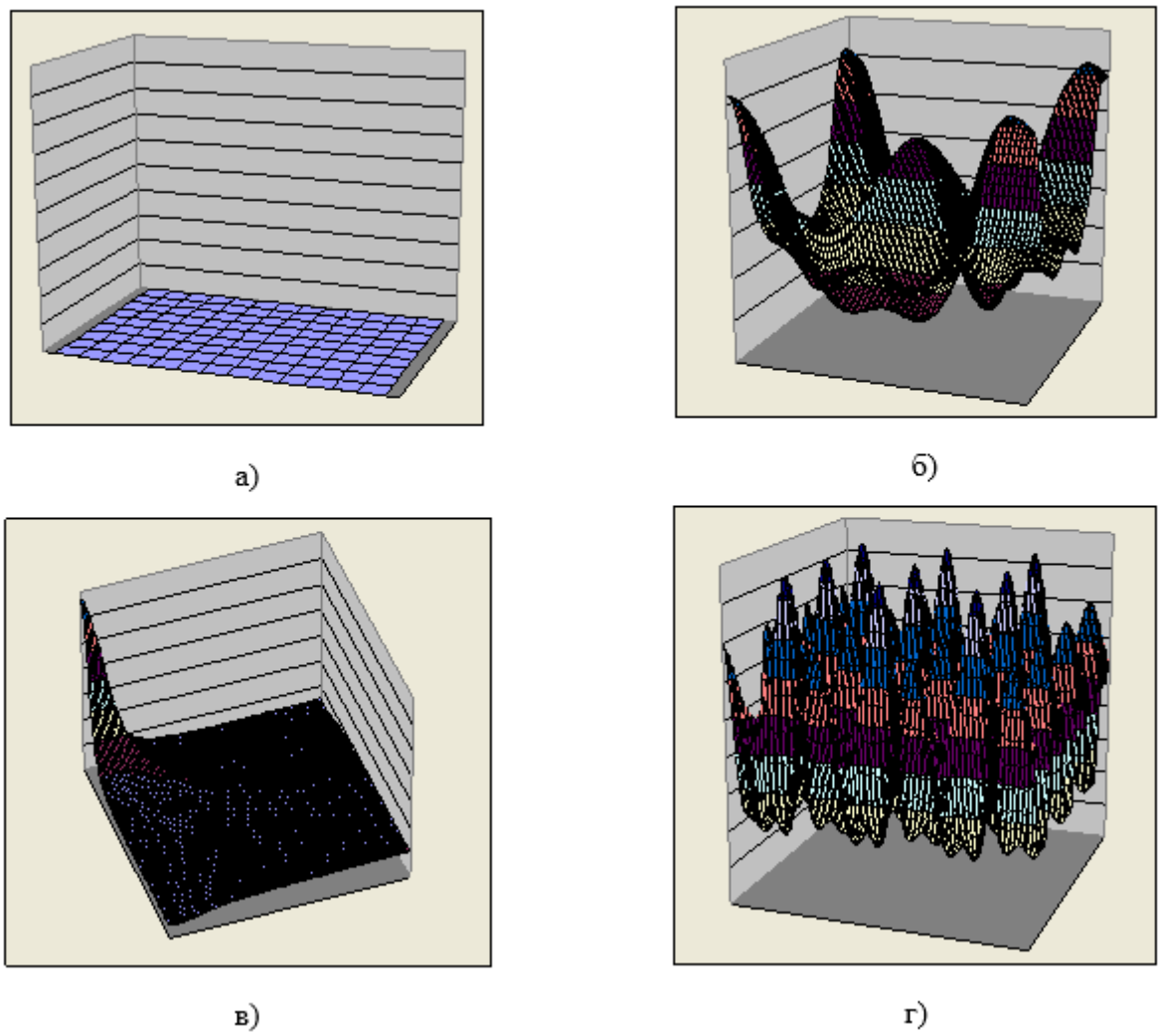


Рисунок 4.4. Поверхні, збудовані за значеннями функції F задачі A2(L)·K). Розбивається безліч - ділянка кривої а) f (x) = const, б) f (x) = x2, в) f (x) = sin (x / 3), г) f (x) = sin (x), $x \in [0,1]$

Таким чином отримали чисельні результати, які у часткових випадках можуть бути візуалізовані і проаналізовани.

Загальна послідовность дій при застосуванні високопродуктивного сервісу може бути описана наступним чином.

1. Розглядаємо загальну постановку задачі.
2. Формулюємо конкретну задачу, тобто загальну задачу із обмеженнями.

3. Здійснюємо трансформацію складності, а саме оптимізаційні процедури трансформуємо у технологічні із застосуванням методів глобального перебору.
4. Виконуємо перехід від неперервної задачі до дискретної, тобто вирішуємо проблему відповідності чисельного представлення вимогам предметної області.
5. Реалізуємо обчислювальний процес на високопродуктивному сервисі.
6. Отримаємо результати розвязання задачі за допомогою стандартних алгоритмів сортування. Треба зазначити, що масив наявних даних дозволяє ефективно отримати значення у довільних точках.
7. Будуємо графічне представлення гіперповерхонь для візуально аналізу.

Очевидно, що така технологій розвязання задач не завжди може бути застосована і вимагає розуміння методів які використовуються при класичних способах пошуку розвязків. Але наявність широкодоступних обчислювальних ресурсів дійсно змінює приоритети та вимагає створювати нові рішення.

ВИСНОВКИ

Метою роботи є розробка та оцінка ефективності системи хмарних сервісів AWS для розв’язання задач оптимального розбиття множин у часткових випадках. Основним завданням є інтеграція AWS Lambda у обчислювальний процес при розв’язанні оптимізаційної задачі. Крім того, оцінюється ефективність застосування нечіткої логіки для покращення точності, зменшення помилок при проектуванні та підвищення ефективності.

Методи, застосовані для розв’язку поставленої задачі, включають:

- методи системного аналізу для дослідження технологій застосування високопродуктивних обчислень;

- методи функціонального та об’єктного моделювання предметної області;

- методи тестування та верифікації програмного коду для перевірки коректності та ефективності розробленої системи.

Отримані результати:

1. Представлено теоретичні та методологічні засади інтеграції високопродуктивних обчислювальних сервісів у методологію розв’язання математичних задач.

2. Розглянуто класичну задачу оптимального розбиття множин.

3. Розглянуто задачу оптимального розбиття з обмеженнями.

4.Проаналізовано можливість переходу до дискретного представлення моделі на етапі розрахунку

5. Реалізовано застосування хмарних сервісів для організації обчислювального процесу.

6. Створено програмне забезпечення, яке застосовуєAWS Lambda для розвязання оптимізаційної задачі.

7. Проведено обчислювальний експеримент

8. Проаналізовано отримані результати.

Кінцевий ефект від використання запропонованих рішень полягає у:

- підвищенні точності та гнучкості розв’язання оптимізаційних задач;

- зменшенні помилок у процесі проектування завдяки переносу складності;

-- підвищенні ефективності управління результатами за рахунок накопичення даних.

Напрямками подальшої роботи є:

- удосконалення методів інтеграції хмарних сервісів у математичні розрахунки;

- розробка та тестування нових алгоритмів для підвищення ефективності обробки великих обсягів даних;

- дослідження можливостей застосування хмарних сервісів для вдосконалення процесів оптимізації та прийняття рішень у реальних задачах;

- створення інтерфейсу для зручнішого використання результатів розрахунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Киселева, Е.М. Модели и методы решения непрерывных задач оптимального разбиения множеств: линейные, нелинейные, динамические задачи [Text] / Е.М. Киселева, Л.С. Коряшкина. – К. : Наукова думка, 2013. – 606 с.

2. Киселева, Е.М. Решение непрерывных задач оптимального покрытия шарами с использованием теории оптимального разбиения множеств [Text] / Е.М. Киселева, Л.И. Лозовская, Е.В. Тимошенко // Кибернетика и системный анализ. – 2009. – №3. –С.98–117.

3. Киселева, Е. М. Непрерывные задачи оптимального разбиения множеств: теория, алгоритмы, приложения [Text] / Е.М. Киселева, Н.З. Шор – К.: Наукова думка, 2005. – 564 с.

4. Шор, Н.З. Методы минимизации недифференцируемых функций и их приложение [Text]/. – К.:Наукова думка, 1979. – 200с.

8. Bakolas E., Tsiotras P. The Zermelo–Voronoi diagram: a dynamic partition problem [Text] //Automatica. –2010. –N12. –P. 2059–2067.

9. Balzer M. Capacity-constrained Voronoi diagrams in continuous spaces [Text] // The International Symposium on Voronoi Diagrams in Science and Engineering. –2009. –10p.

10. Blyuss O., Koriashkina L., Kiseleva E., Molchanov R. Optimal placement of irradiation sources in the planning of radiotherapy[Text] // Mathematical models and methods of solving Computational and mathematical methods in medicine. – 2015.

11. Jooyandeh Mohammadreza, Mohades Ali, Mirzakhah Maryam. Uncertain Voronoi diagram [Text] // Information Processing Letters. – 2009. –109, N13. – P. 709–712.

12. Kiseleva E.M., Koriashkina L.S. Theory of Continuous Optimal set Partitioning Problems as a Universal Mathematical Formalism for Constructing

Voronoi Diagrams and Their Generalizations [Text] // Cybernetics and Systems Analysis, –2015. – P. 325–335.

12. Koriashkina L., Cherevatenko A., Mykhalova O. The continuous problems of the optimal multiplex partitioning an application of sets[Text] // Power Engineering and Information Technologies in Technical Objects Control: –2016 Annual Proceedings, CRC Press/ Balkema. P.233–240.

14. Lau B., Sprunk C., Burgard W. Efficient grid-based spatial representations for robot navigation in dynamic environments[Text] // Robotics and Autonomous Systems. – 2013. – 61, N 10. – P. 1116–1130.

15.Us S., Stanina O. The Methods and Algorithms for Solving Multi-stage Location-allocation Problem [Text] / Power Engineering and Information Technologies in Technical Objects Control:2016 Annual Proceedings CRC Press, –2017.

16. Overview of Amazon Web Services (2024) https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html?did=wp_card&trk=wp_card

17. Глушков В. М. Введение в АСУ. — Киев: Техника, 1972. — 312 с.

Додаток А Лістинг C# коду

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    class Logic
    {

        //double[,] doublesMatrix = new double[10,10];

        //podaem pustoy dvumerny massiv, odnomerny dlya obrabotki, dlinu otrezka
        public static void CreateMatrix(double[,] dMatrix, double[] A, int points)
        {
            double s=0;

            for (int ii = 0; ii < A.Length - points + 1; ii++)
            {
                //s = 0;
                //sravnivaem odin ryad s pointsami
                for (int i = 0; i < A.Length - points + 1; i++)
                {
                    s = 0;
                    //summa kvadratov raznostey n=points elementov
                    for (int j = 0; j < points; j++)
                    {
                        s = s + Math.Pow((A[A.Length - 1 - i - j] - A[A.Length - 1 -
j - ii]), 2);
                    }
                    //mera v massiv, c uchetom dliny z-vektora
                    dMatrix[i, ii] = Math.Sqrt(s/points);
                }
                //zapolnenie pustyh strok
                for (int i = (A.Length - points + 1); i < A.Length; i++)
                {
                    dMatrix[i, ii] = 9.9999;
                }
                    //if (s == 0) dMatrix[i, ii] = 9.9999;

            }
            //zapolnenie pustyh stolbtsov
            for (int ii = A.Length - points + 1; ii < A.Length; ii++)
            {
                for (int i = 0; i < A.Length; i++)
                {
                    dMatrix[i, ii] = 9.9999;
                }
            }
        }



        public static void SaveToFile(double[] arr, string txtFile)
        {
```

```
            using (System.IO.StreamWriter file = new System.IO.StreamWriter(txtFile,
false))
            {
                foreach (double d in arr)
                {
                    file.WriteLine(d);
                }
            }
        }

        public static void SaveToFileMatrix(double[,] arr, string txtFile)
        {
            using (System.IO.StreamWriter file = new System.IO.StreamWriter(txtFile,
false))
            {
                int j = 0;
                for (int i=0; i< 1000; i++)
                {
                    for (j = 0; j < 1000; j++) { file.Write("{0:F4}",arr[i, j]);
file.Write(' '); }
                    file.WriteLine();
                }
            }
        }

        public static void Raspredelenie(double[,] arr, int[] counter, string
txtFile)
        {
            int N100=counter.Length;
            int NN = 10000;
            //opredelyaem min max dvumernogo massiva arr
            double min = 0;
            double max = 1;



            //obnulyaem schetchik
            for (int j = 0; j < N100; j++)
            {
                counter[j] = 0;
            }
                //raspredelenie

                for (int i = 0; i < NN; i++)
                {
                    for (int j = 0; j < NN; j++)
                    {
                        for (int k = 0; k < N100 - 1; k++)
                        {
                            if (arr[i, j] != 9.9999)
                            {
                                if ((arr[i, j] >= min + k * (max - min) / N100) &
(arr[i, j] < min + (k + 1) * (max - min) / N100))
                                    counter[k] = counter[k] + 1;
                            }
                        }
                    }
                }

            using (System.IO.StreamWriter file1 = new
System.IO.StreamWriter("1/"+txtFile, false))
            {

                for (int i = 0; i < N100; i++)
                {
```

```
                    file1.Write(counter[i]);
                    file1.WriteLine();
                }
            }

        }


        public static void ReadFromFile(double[] arr, string txtFile)
        {
            using (System.IO.StreamReader file = new System.IO.StreamReader(txtFile,
true))
            {
                int i = 0;
                //while (!file.EndOfStream) { arr[i]=double.Parse(file.ReadLine());
i++; }
                for (i = 0; i < 10000; i++) { arr[i] =
double.Parse(file.ReadLine());}

                //mashtabiruem na interval >0
                double min = arr[0];
                double max = -arr[0];

                for (i = 0; i < 10000; i++)
                {
                        if (arr[i] < min) min = arr[i];
                        if (arr[i] > max) max = arr[i];
                }

                double t = 0;
                if (min < 0) t = -min;

                for (i = 0; i < 10000; i++)
                {
                    arr[i]=arr[i]+t;
                }

            }
        }

        public static void ReadFromFile12(double[] arr, string txtFile)
        {
           double[] tempArr = new double[10000];

            using (System.IO.StreamReader file = new System.IO.StreamReader(txtFile,
true))
            {
                int i = 0;
                //while (!file.EndOfStream) { arr[i]=double.Parse(file.ReadLine());
i++; }
                for (i = 0; i < 10000; i++) { tempArr[i] =
double.Parse(file.ReadLine()); }

                for (i = 0; i < 1000; i++) { arr[i] = tempArr[i]; }
            }
        }

        // Read 100 double from file
        public static void Read100(double[] arr, string txtFile)
        {
            using (System.IO.StreamReader file = new System.IO.StreamReader(txtFile,
true))
            {
               for (int i = 0; i < 100; i++) { arr[i] =
double.Parse(file.ReadLine()); }
```

```
            }
        }

        public static void NewLorenz(double[] arr, int size)
        {
            int i,j;

            double[] NewL = new double[size];
            Random objectR = new Random();
            for (i = 0; i < size; i++)
            {
                NewL[i] = arr[objectR.Next(100)];
            }



            using (System.IO.StreamWriter file1 = new
System.IO.StreamWriter("NewLx", false))
            {

                for ( i = 0; i < size; i++)
                {
                    file1.Write(NewL[i]);
                    file1.WriteLine();
                }
            }
        }


    }

}
```