ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ ТА ХМАРНІ ТЕХНОЛОГІЇ

методичні рекомендації до виконання індивідуальних завдань

Запоріжжя 2025

metinvest polytechnic

УДК 004.42(072)
Р64

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 6 від 28.02.2025 р.)

Укладачі
Костіков О.А., канд. фіз.-мат. наук, доцент;
Сагайда П.І., доктор техн. наук, професор.

Р64 Розподілені обчислення і хмарні технології : методичні рекомендації до виконання індивідуальних завдань (для студентів комп'ютерних спеціальностей першого (бакалаврського) рівня вищої освіти) / уклад.: О. А. Костіков, П. І. Сагайда. Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА». 2025. 35 с.

Методичні вказівки містять відомості щодо формування кейсу задач та критеріїв оцінювання індивідуальних завдань на платформі Moodle та приклади їх розв'язання з наведенням алгоритмів, комп'ютерних програм, скріншотів результатів та коментарів до розробленого програмного коду. Матеріал методичних вказівок має на меті підвищити якість виконання роботи, виробити навички розв'язання задач, пов'язаних з принципами реалізації розподілених та паралельних обчислень та використанням хмарних технологій.

Рекомендовано для студентів комп'ютерних спеціальностей та форм навчання першого (бакалаврського) рівня освіти.

УДК 004.42(072)

© ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ МЕТІНВЕСТ ПОЛІТЕХНІКА», 2025

ЗМІСТ

ВСТУП

Навчальна дисципліна «Розподілені обчислення та хмарні технології» є складовою освітньо-наукової програми підготовки фахівців за першим (бакалаврським) рівнем вищої освіти у галузі знань 12 Інформаційні технології за спеціальністю 122 Комп'ютерні науки в рамках освітньо-професійної програми «Комп'ютерні науки».

Дана дисципліна є обов'язковою дисципліною блоку спеціалізації «Комп'ютерні науки».

Курс «Розподілені обчислення та хмарні технології» присвячений вивченню сучасних підходів до проектування розподілених систем та інструментальних засобів їх реалізації. Дисципліна охоплює сучасні підходи, стандарти та технології для розподілених обчислень.

Мета дисципліни – формування комплексу знань і здобуття навичок з розробки систем паралельних та розподілених обчислень для широкого практичного застосування у системах різного призначення.

Завдання дисципліни: опанування студентами методів та алгоритмів побудови систем розподілених та паралельних обчислень для різного застосування..

В рамках освоєння дисципліни «Розподілені обчислення та хмарні технології» за змістовними модулями першого семестру студентам пропонується виконати індивідуальні завдання з розв'язування прикладних задач за темами відповідних змістовних модулів. Представлені методичні рекомендації містять відомості щодо формування кейсу задач та критеріїв оцінювання індивідуального завдання на платформі Moodle та приклади їх розв'язання з наведенням алгоритмів, їх програмної реалізації, скріншотів результатів та коментарів до розроблених програм в обсязі, необхідному для виконання індивідуального завдання.

1 ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1.1 Відомості щодо формування кейсу задач та критеріїв оцінювання індивідуального завдання на платформі Moodle

На платформі Moodle кейс задач індивідуальних завдань з дисципліни «Розподілені обчислення та хмарні технології», за змістовними модулями першого семестру формується рандомно системою шляхом автоматичного вибору окремого варіанта серед наявних в рамках відповідної категорії. Кожен студент отримає однаковий за кількістю та змістовністю кейс задач в межах відповідної спеціальності з зазначенням переліку завдань, що виносяться на виконання. Заохочується використання мов програмування Python, C++ та Java для ілюстрації роботи алгоритмів, які розглядаються в задачі.

Індивідуальні завдання виконуються самостійно у зручний для студента час в межах терміну подачі роботи, передбачених у розділі «Розподіл балів за контрольними точками та графік їх виконання» та розміщується у відповідному розділі на платформі Moodle. Розв’язання кожного завдання завантажується у вигляді файлу з розширенням .docx або .pdf, або .jpg, або .png, або .txt (за наявності розробленої програми у форматах .cpp, .py, .java завантажується додатково).

Максимальна кількість балів вказана за кожне окреме завдання у системі Moodle та визначається в залежності від обґрунтування ходу розв’язання, програмної реалізації, коментування програмного коду, правильності отриманого розв’язку та аналізу результату.

Використання штучного інтелекту (ШІ) не забороняється, оскільки пропозиції відомих застосунків ШІ суттєво залежать від обміркованої постановки питання і уточнюючих питань; однак в разі, якщо відповідь, отримана з використанням ШІ, містить суттєві похибки або не є комплексною, або не відповідає за усталеним оформленням, термінологією, або іншим вимогам до завдання, то оцінка за виконання знижується.

Перевірка індивідуального завдання виконується протягом тижня після завершення терміну подачі роботи. За побажанням студента при наявності похибок або виконання індивідуального завдання не в повному обсязі допускається доопрацювання до передостаннього тижня навчання.

1.2 Кейс задач індивідуальне завдання

Завдання 1. Розв’язання задач на визначення міток часу Лампорта.

Завдання 2. Розв’язання задач на використання векторного годинника.

Завдання 3. Розв’язання задач на використання алгоритму взаємного виключення Лампорта.

Завдання 4. Розв’язання задач на використання операцій типу «крапка-крапка» інтерфейсу передачі повідомлень(Message Passing Interface) MPI.

Завдання 5. Розв’язання задач на використання колективних операцій MPI.

Завдання 6. Розв’язання задач на використання директиви розпаралелення циклів #pragma omp for відкритого стандарту для розпаралелення програм OpenMP.

Завдання 7. Розв’язання задач на використання корпоративної служби аналітики Azure Synapse Analytics.

Завдання 8. Розв’язання задач на використання хмарної платформи OpenStack.

РОЗВ’ЯЗАННЯ ТИПОВОГО ВАРІАНТУ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

2.1 Приклад розв’язання завдання 1

Завдання 1. Для подій в процесах, зображених на рис.1, поставити мітки часу Лампорта.

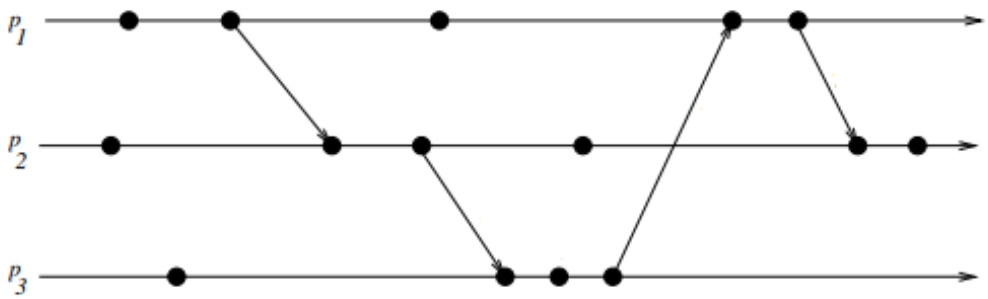


Рисунок – 1 Просторово-часова діаграма розподіленого виконання

Розв’язання.

На рисунку 1 показано просторово-часову діаграму розподіленого виконання за участю трьох процесів. Горизонтальна лінія відображає хід процесу; крапка вказує на подію; похила стрілка вказує на передачу повідомлення. Як правило, виконання події займає скінченну кількість часу; однак, оскільки ми припускаємо, що виконання події є атомарним (отже, неподільним і миттєвим), то виправдано позначати його крапкою на лінії процесу. На цьому рисунку для процесу p_1 друга подія - це подія надсилання повідомлення, третя подія - внутрішня подія, а четверта подія - подія отримання повідомлення.

Для визначення логічного часу для кожної події використовуємо алгоритм Лампорта.

Алгоритм роботи міток часу Лампорта можна описати кількома правилами:

Усі лічильники процесу починаються зі значення 0.

Процес збільшує свій лічильник для кожної події (внутрішня подія, надсилання повідомлення, отримання повідомлення) у цьому процесі. Коли процес надсилає повідомлення, він включає значення свого (збільшеного) лічильника до повідомлення. При отриманні повідомлення лічильник одержувача оновлюється до більшого з двох значень: поточного лічильника та мітки часу у отриманому повідомленні, а потім збільшується на одиницю.

Застосуємо цей алгоритм для ситуації, зображеної на рис.1.

Коли відбувається перша внутрішня подія для процесу p_1, його локальний лічильник збільшується на 1. Оскільки початкове значення

лічильника було 0, то після першої події значення лічильника буде дорівнювати 1. Друга подія для процесу p_1- надсилання повідомлення. Локальний годинник процесу p_1 збільшується на 1 і стає рівним 2. Ця мітка часу буде міститися у переданому до процесу p_2повідомленні. Для процесу p_2перша подія була внутрішньою, після якої локальний лічильник процесу p_2отримав значення 1. Друга подія для процесу p_2- отримання повідомлення. При отриманні повідомлення локальний лічильник процесу p_2 збільшить своє значення на 1 і буде дорівнювати 2. Для отримання нового часу в p_2 беремо максимум між отриманою від процесу p_1міткою часу та власним локальним часом процесу p_2 max(2,2) і збільшуємо його на 1. Це призведе до нової мітки часу зі значенням 3. Наступна подія для процесу p_2 – надсилання повідомлення процесу p_3. При цьому його мітка часу стане рівною 4, і це значення буде передано у повідомленні до процесу p_3. Прийняття повідомлення є подією для процесу p_3, тому значення його локального лічильника збільшиться на 1 і стане рівним 2, оскільки до прийняття повідомлення в процесі p_3 відбулася одна внутрішня подія, що викликало збільшення лічильника на 1. При прийнятті повідомлення значення лічильника p_3 оновиться до max(4,2)+1=5(4 – мітка часу, передана процесом p_2, 2 - значення локального лічильника процесу p_3). При передачі повідомлення від процесу p_3 до процесу p_1 значення локального часу процесу p_3 буде дорівнювати 7, оскільки між прийняттям повідомлення від процесу p_2 та відправленням повідомлення до процесу p_1 для процесу відбулася одна внутрішня подія. Значення 7 буде передано у повідомленні від процесу p_3 до процесу p_1. При цьому значення власного годинника процесу p_1 буде дорівнювати 4. При прийнятті повідомлення значення власного лічильника процесу p_1 оновиться до max(7,4)+1=8 (7 – мітка часу, передана процесом p_3, 4 - значення локального лічильника процесу p_1). Наступна подія для процесу p_1 – надсилання повідомлення процесу p_2. Мітка часу, що міститься у повідомленні, буде 9: локальний час процесу 8, збільшений на 1(8+1). Значення власного годинника процесу p_2 на момент прийняття цього повідомлення буде дорівнювати 6(значення мітки часу для попередньої події 5, збільшена на 1 (5+1=6), оскільки прийняття повідомлення є подією для процесу p_2). Тому оновлене значення лічильника процесу p_2 буде дорівнювати max(9,6)+1=10.

Мітки часу, отриманні в результаті застосування алгоритму Лампорта для визначення локального часу, показано на рис.2. Над подіями показано значення міток Лампорта, над стрілками – мітка часу, яка міститься у повідомленні, що передається від одного процесу до іншого.

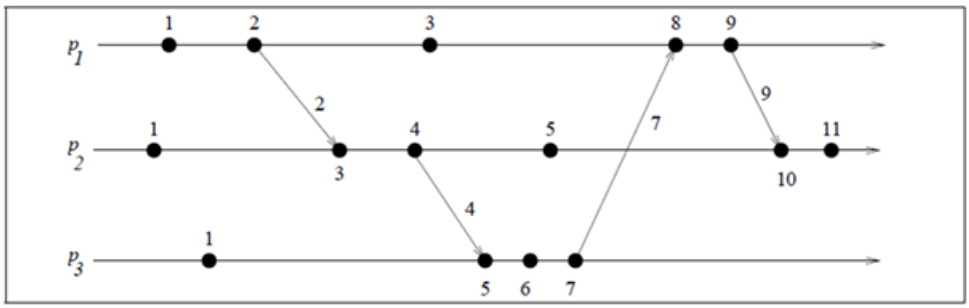


Рисунок 2 – Еволюція скалярного часу

2.2 Приклад розв'язання завдання 2

Завдання 2. Для подій в процесах, зображених на рис.1, поставити значення векторного годинника.

Розв'язання.

Векторний годинник системи з N процесами - це вектор з N лічильників, по одному лічильнику на процес. Векторні лічильники повинні відповідати наступним правилам оновлення:

Спочатку всі лічильники дорівнюють нулю ([0,0,0] у нашому прикладі)

Кожного разу, коли процес стикається з подією, він збільшує свій власний лічильник у векторі на одиницю.

Кожного разу, коли процес надсилає повідомлення, він додає до нього копію власного (збільшеного) вектору.

Кожного разу, коли процес отримує повідомлення, він збільшує свій лічильник у векторі на одиницю і оновлює кожен елемент у своєму векторі, беручи максимальне значення зі свого векторного лічильника і значення у векторі в отриманому повідомленні.

Використовуючи вищенаведений алгоритм, отримаємо значення векторного годинника для подій, зображених на рис.1.

Для першої внутрішньої події процесу p_1 значення векторного годинника буде дорівнювати [1,0,0], оскільки значення власного лічильника процесу збільшується на 1 при настанні події. Друга подія процесу p_1 – надсилання повідомлення процесу p_2, після якої значення векторного годинника стане [2,0,0]. Ця копія векторного годинника надається у повідомленні до процесу p_2. Після отримання цього повідомлення процес p_2 оновлює кожен елемент у своєму векторі. При прийнятті повідомлення значення векторного годинника процесу p_2 було [0,2,0]. Після оновлення отримаємо нове значення векторного годинника процесу p_2: [2,2,0]. Перша компонента обчислюється наступним чином: max(0,2)=2, де 0 – значення першої компоненти лічильника процесу p_2, 2- значення першої компоненти лічильника p_1. Для другої компоненти маємо: max(2,0)=2, де 2 – значення першої компоненти лічильника процесу p_2, 0- значення першої компоненти лічильника p_1. Третя компонента дорівнює max(0,0)=0. Значення векторного годинника для інших компонент знаходимо аналогічно.

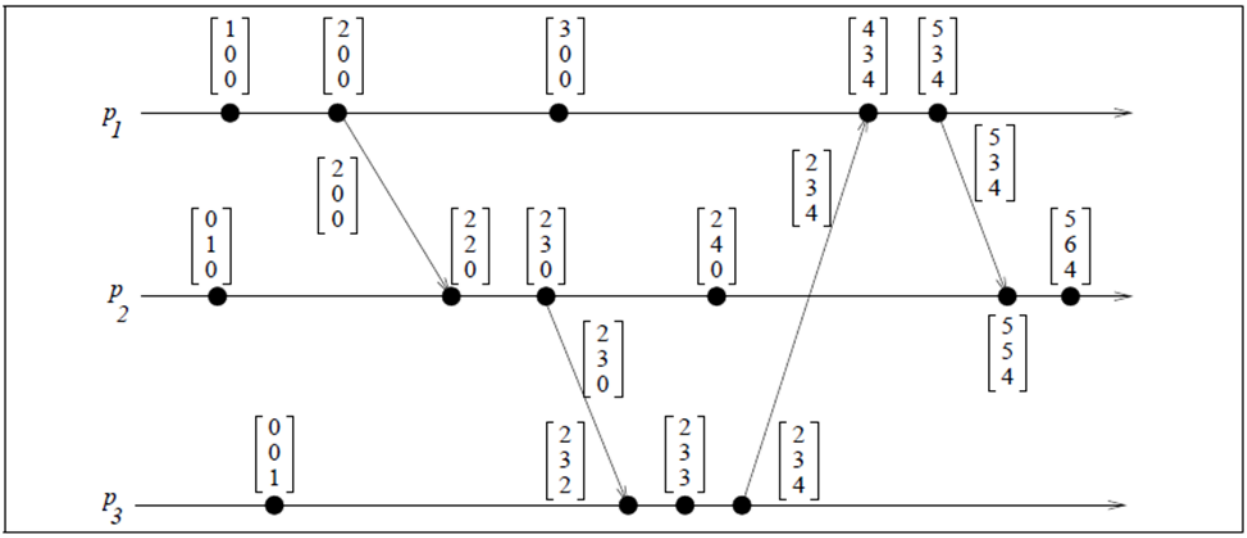


Рисунок 3 – Еволюція вектору часу

2.3 Приклад розв'язання завдання 3

Завдання 3. Проілюструвати роботу алгоритму Лампорта для трьох процесів в ситуації, коли практично одночасно процеси p_2 та p_3 запитують вхід в критичну секцію(КС). Після входу в КС процесу p_2 або p_3 процес p_1 запитує вхід в КС.

Розв'язання.

Позначимо через Q_i локальну чергу процесу p_i, у яку поміщаються всі запити на доступ до КС разом з їхніми відмітками часу.

Тоді алгоритм Лампорта визначатиметься такими правилами.

Запит на вхід у КС. Коли процесу p_i потрібен доступ до КС, він розсилає повідомлення $REQUEST(L_i, i)$ зі значенням свого логічного часу (L_i, i) усім іншим процесам і, крім того, поміщає цей запит у свою чергу Q_i.Тут L_i – скалярний час процесу p_i, i – номер процесу. Кожна така розсилка являє собою одну атомарну подію процесу p_i. Коли процес p_j отримує запит $REQUEST(L_i, i)$ від процесу p_i, він поміщає його у свою чергу Q_j і надсилає процесу p_i відповідь $REPLY(L_j, j)$ зі значенням свого логічного часу (L_j, j).

Вхід у КС. Процес p_i може увійти в КС, якщо одночасно виконуються дві умови:

1. запит $REQUEST(L_i, i)$ процесу p_i має найменше значення позначки часу серед усіх запитів, що перебувають у його власній черзі Q_i.
2. процес p_i отримав повідомлення від усіх інших процесів у системі з відміткою часу більшою, ніж (L_i, i). За умови того, що канали зв'язку мають властивість FIFO, дотримання цього правила гарантує, що процесу p_i відомо про всі запити, що передують його

поточному запиту.

Вихід із КС. Під час виходу з КС процес p_i видаляє свій запит із власної черги Q_i і розсилає всім іншим процесам повідомлення $RELEASE(L_i, i)$ з позначкою свого логічного часу.

Як і раніше, кожна така розсилка є однією атомарною подією процесу p_i. Отримавши повідомлення $RELEASE(L_i, i)$, процес p_j видаляє запит процесу p_i зі своєї черги Q_j. Після того, як процес p_j видалив запит процесу p_i з Q_j мітка часу його власного запиту може бути найменшою в Q_j, а p_j зможе розраховувати на вхід в КС.

Застосуємо алгоритм взаємного виключення Лампорта для рішення нашої задачі. . На рис. 4а приблизно в той же час процеси p_2 і p_3 запитують вхід в КС, змінюючи показання своїх скалярних годинників L_2 і L_3; локальна черга кожного процесу представлена прямокутником (припустимо, що в чергах запити сортуються в порядку зростання їх тимчасових міток).

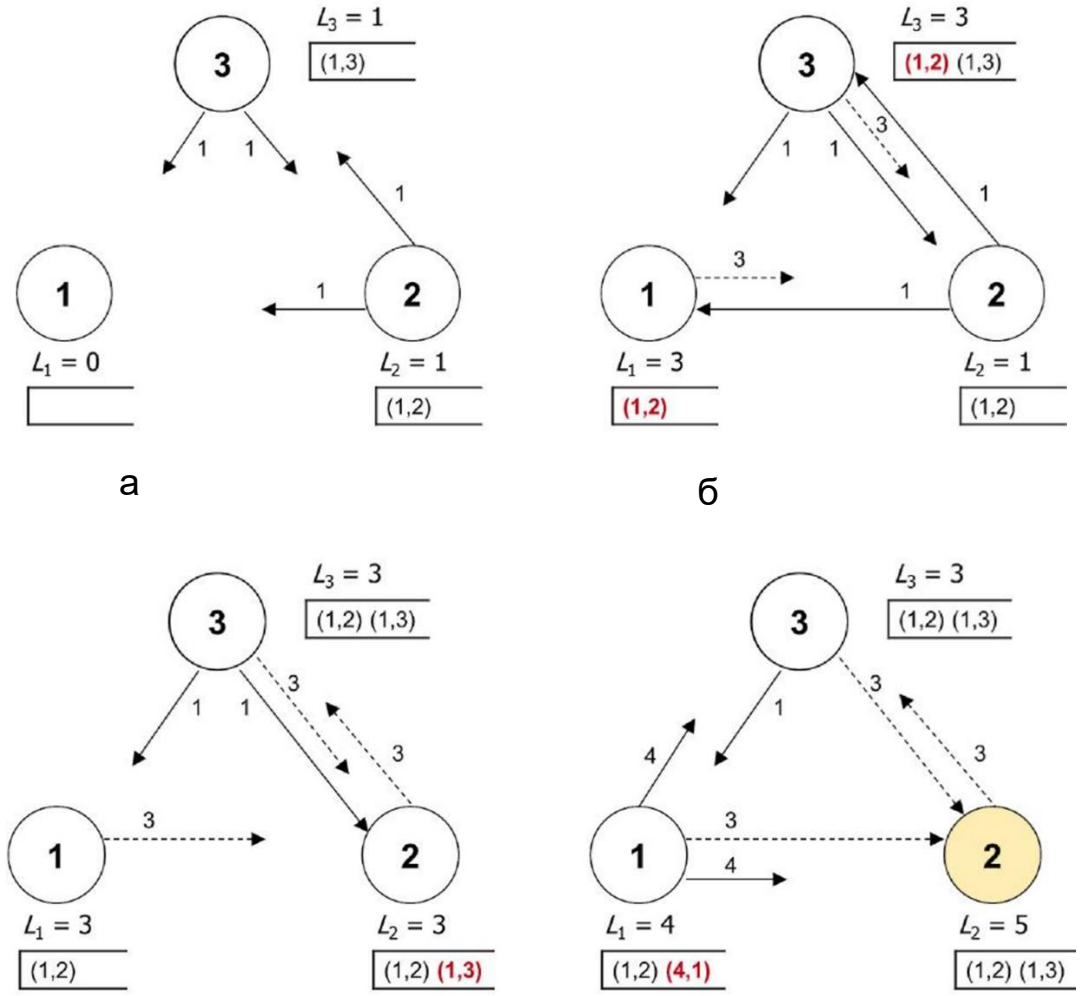
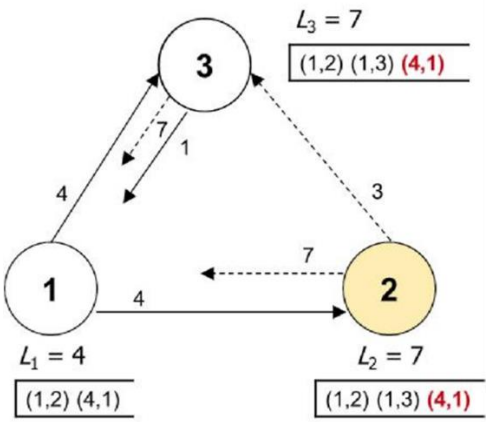
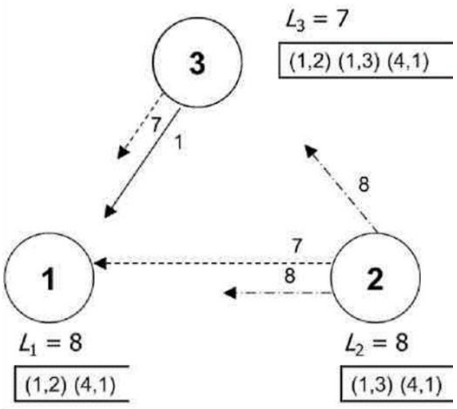


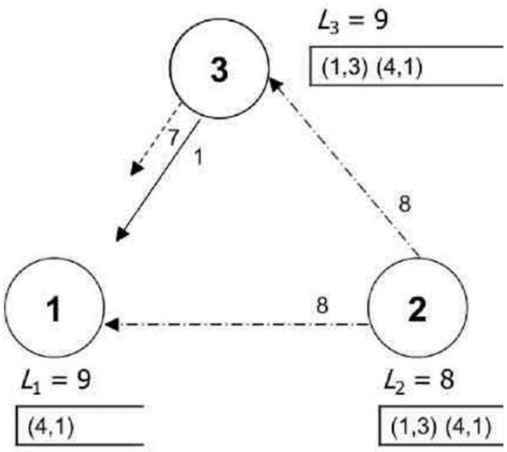
Рисунок 4 – Приклад роботи алгоритму Лампорта, лист 1



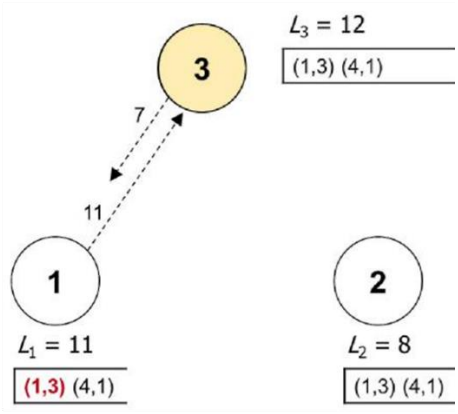
д



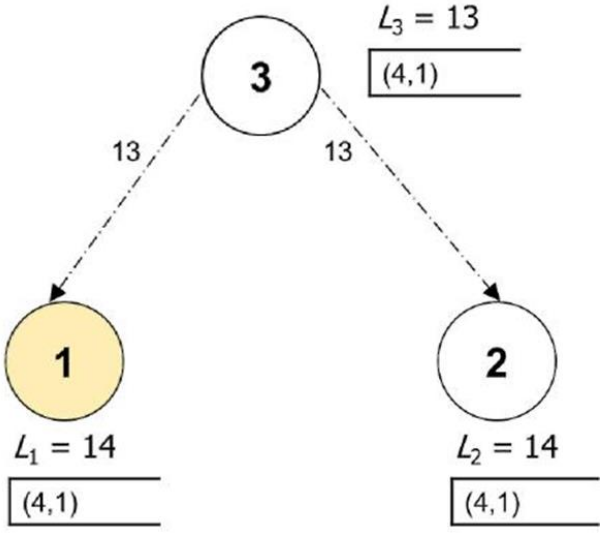
е



ж



з



і

Рисунок 4 – Приклад роботи алгоритму Лампорта, лист 2

На рисунку 4б процеси p_1 і p_3 отримують запит від p_2 і відправляють на нього повідомлення у відповідь. Так як мітка часу запиту p_2 менше часової мітки запиту p_3, то запит від p_2ставиться перед запитом від p_3 у власній черзі Q_3. Це дозволяє p_2 увійти в КС раніше p_3. p_2 зможе розраховувати на вхід в КС тільки при отриманні повідомлень у відповідь від p_1 і p_3. Зауважимо, що через властивість FIFO каналів зв'язку p_2 не зможе отримати повідомлення у відповідь від p_3 раніше за запит, відправлений раніше процесом p_3. Тому раніше p_2 отримає запит від p_3, помістить його у свою локальну чергу Q_2 і надішле повідомлення у відповідь на p_3. Оскільки мітка часу запиту, отриманого від p_3, більша, ніж мітка часу запиту p_2, запит з p_3 розміщується в кінці черги Q_2, як показано на рисунку 4в. Після отримання повідомлень у відповідь від p_1 і p_3, процес p_2 увійде в КС, тому що його запит стоїть на чолі власної черги Q_2 – див. рис. 4г. Далі в нашому сценарії процес p_1 переходить в стан запиту на вхід в КС і відправляє відповідне повідомлення іншим процесам. Отримавши запит від p_1, процеси p_2 і p_3 надсилати повідомлення у відповідь – див. 4д.

Зверніть увагу, що до цього часу запит на вхід в КС від процесу p_3 був отриманий і підтверджений процесом p_2. Однак той самий запит, надісланий процесу p_1, не був отриманий. На рис. 1е, процес p_2 виходить з КС, видаляє свій запит з черги Q_2 і відправляє всім процесам повідомлення $RELEASE$. Охочий увійти в КС процес p_1 отримує від p_2 повідомлення у відповідь, а потім повідомлення $RELEASE$, що призводить до видалення запиту з p_2 з черги Q_1 – див. рис. 1ж. Зверніть увагу, що на рис. 1ж і процес p_1, і процес p_3 знаходяться в стані запиту на вхід в КС.

У локальній черзі кожного з цих процесів перше місце займає власний запит процесу. Обидва ці процеси отримали повідомлення у відповідь від p_2 . Однак процес p_3 зможе отримати доступ до КС раніше, ніж p_1, так як його запит стався раніше запиту p_1. Дійсно, p_1 не зможе увійти в КС до тих пір, поки не отримає повідомлення у відповідь від p_3. У зв'язку з властивістю FIFO каналу зв'язку між p_3 і p_1, це повідомлення буде відправлено p_1 тільки після отримання запиту від p_3, як показано на малюнку 4ж.

Коли p_1 отримує запит від p_3, він ставить його попереду власного запиту у своїй черзі Q_1, оскільки запит від p_3 має меншу мітку часу, ніж запит від p_1 – див. рис. 4з. Тепер p_1 не зможе отримати доступ до КС доти, доки p_3 не ввійде і не вийде з КС. У свою чергу, p_3 увійде в КС при отриманні повідомлення у відповідь від p_1 – див. рис. 4з. При виході з КС p_3 відправить всім процесам повідомлення $RELEASE$., який видалить його запит з усіх черг, включаючи чергу процесу p_1. Після цього запит процесу p_1 буде першим у власній черзі Q_1, і він зможе увійти в КС, як показано на рис. 4і.

2.4 Приклад розв’язання завдання 4

Завдання 4.Припустимо, що ви повинні передати буфер масиву з усіх інших вузлів на один вузол за допомогою функцій відправки/прийому, які використовуються для внутрішньо процесної синхронної комунікації. За допомогою функцій MPI(Message Passing Interface – інтерфейс передачі повідомлень) реалізуйте функціональність, зображену на рис. 5.

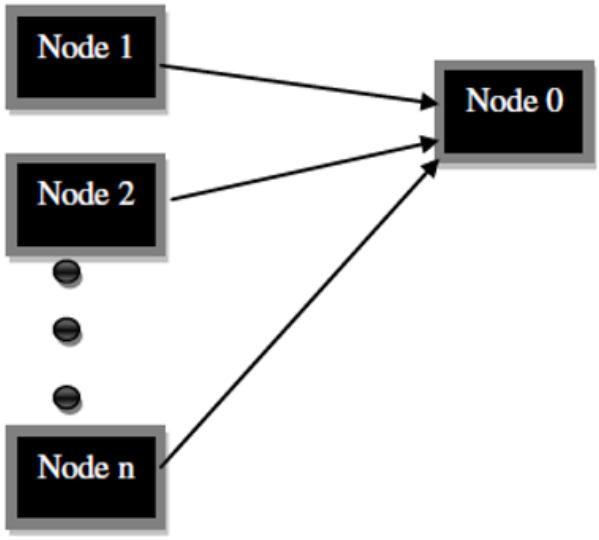


Рисунок 5 - Передача повідомлень між процесами

Розв’язання.

Вихідний код програми, яка реалізує заданий функціонал, наведено на рис.6. Дана програма написана на мові програмування C++ з використанням функцій MPI. Згідно з рис.5, кожен з процесів Node1, Node2, …, Node n повинен передати масив даних на вузол Node 0. Для реалізації цієї передачі будемо використовувати операції типу «крапка-крапка» MPI.

Пояснимо призначення кожного рядка програмного коду, наведеного на рис.6.

1. Цей рядок підключає стандартну бібліотеку вводу-виводу для введення даних з консолі та виводу їх на консоль.
2. В цьому рядку підключається бібліотека MPI для паралельних обчислень. Завдяки цьому ми можемо використовувати функції MPI в нашій програмі.
3. Цей рядок містить оголошення функції main, з якої починається виконання нашої програми. Ця функція містить позиційні і ключові параметри, які можна задавати в командному рядку. Функція повертає ціле число типу int.

В цьому рядку оголошуються змінні цілого типу mynode та totalnodes. Змінна mynode використовується для зберігання рангу(ідентифікатору)

поточного процесу, а змінна totalnodes – для зберігання кількості процесів.

```
1  #include <iostream>
2  #include <mpi.h>
3  int main(int argc, char** argv) {
4      int mynode, totalnodes;
5      int datasize=3; // number of data units to be sent/recv
6      MPI_Status status; // variable to contain status information
7      MPI_Init(&argc, &argv);
8      MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
9      MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
10     // Determine datasize
11     double* databuffer = new double[datasize];
12
13     // Fill in sender, receiver, tag on sender/receiver processes,
14     // and fill in databuffer on the sender process.
15     if (mynode != 0) {
16         for (int j = 0; j < datasize; j++)
17             databuffer[j] = (j+1)*mynode;
18         MPI_Send(databuffer, datasize, MPI_DOUBLE, 0,
19             1, MPI_COMM_WORLD);
20     }
21     else
22         for (int j = 1; j < totalnodes; j++) {
23             MPI_Recv(databuffer, datasize, MPI_DOUBLE, j, 1,
24                 MPI_COMM_WORLD, &status);
25             std::cout << "Data received from process " << j<<"\n";
26             for (int k = 0; k < datasize; k++)
27                 std::cout << databuffer[k] << " ";
28             std::cout << "\n";
29         }
30     // Send/Recv complete
31     MPI_Finalize();
32 }
```

Рисунок 6 – Вихідний код програми для завдання 4

4. В цьому рядку оголошується змінна datasize, яка визначає кількість елементів масиву, який передається процесами. При оголошення цій змінній присвоюється значення 3.

5. Цей рядок містить оголошення змінній status типу MPI_Status. MPI_Status – це структура, яка містить інформацію про прийняте повідомлення.

6. Функція MPI_Init(&argc, &argv), яка викликається в цьому рядку, ініціює MPI середовище.

7. У цьому рядку функція MPI_Comm_size(MPI_COMM_WORLD, &totalnodes) визначає кількість процесів у змінній totalnodes. MPI_COMM_WORLD – ідентифікатор глобального комунікатора, який містить всі процеси.

8. Функція MPI_Comm_rank(MPI_COMM_WORLD, &mynode) повертає ідентифікатор(ранг) процесу в рамках заданого комунікатора і зберігає його в змінній mynode.

9. Цей рядок містить коментар.

10. Виділяється пам’ять під масив чисел з плаваючою комою, який містить datasize елементів. Змінна databuffer містить показчик на цей масив, тобто адресу його першого елементу. Рядки з 12 по 14 містять коментарі.

15. Перевіряється, чи відрізняється ранг поточного процесу від 0.

16. Організується цикл for, в якому параметр циклу j змінюється від 0 до datasize-1 з кроком 1.

17. Присвоюються значення елементам масиву databuffer.

18. Дані масиву databuffer, який містить datasize дійсних чисел(тип даних MPI_DOUBLE) надсилаються процесу з рангом 0 за допомогою функції MPI_Send.Кожен процес з рангом, відмінним від 0, надсилає свій буфер даних. Тег повідомлення дорівнює 1.

21. Визначає блок коду, який буде виконуватися процесом з рангом 0.

22. Організується цикл for, в якому параметр циклу j змінюється від 1 до totalnodes-1 з кроком 1. Таким чином, змінна j послідовно приймає значення рангу всіх процесів в комунікаторі, відмінних від 0.

23. Дані буферу обміну від кожного процесу з рангом j, який містить datasize чисел з плаваючою комою, отримаються процесом з рангом 0. Тег повідомлення дорівнює 1. Він використовується для того, щоб розрізнити різні типи повідомлень. В змінній status міститься інформація про отримане повідомлення після завершення функції MPI_Recv.

26-31. В цих рядках в процесі з рангом 0 друкуються отримані дані від інших процесів.

33. Виклик функції MPI_Finalize() завершує роботу MPI-застосунку.

Результати роботи програми наведені на рис.7.

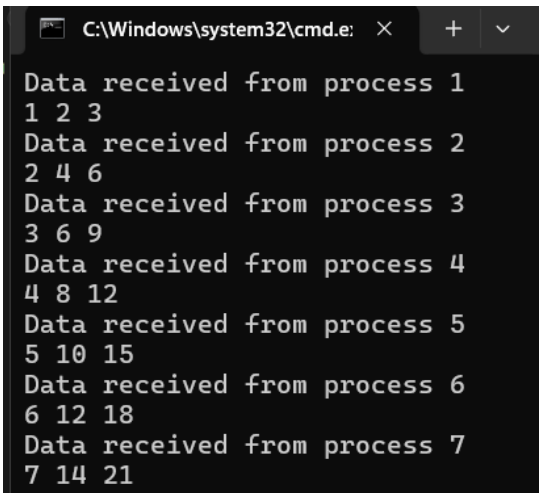


Рисунок 7 – Результати виконання завдання 4

2.5 Приклад розв'язання завдання 5

Завдання 5. Використовуючи колективні операції MPI, написати програму обчислення числа π.

Розв'язання. В нижченаведеній програми для обчислення числа π використовується той факт, що

$$\pi = 4\int_0^1 \frac{dx}{1+x^2}$$

Для наближеного обчислення інтеграла використовується формула середніх прямокутників.

На рис. 8 наведена MPI реалізація програми для обчислення числа π.

```
1  # include <iostream>
2  # include <mpi.h>
3  using namespace std;
4  double f(double x)
5  {
6      return 1 / (1 + x * x);
7  }
8  int main(int argc, char* argv[]) {
9      double pi, sum = 0, term, h; int myrank,
10         nprocs, n, i;
11     MPI_Init(&argc, &argv);
12     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
13     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
14     if (myrank == 0)
15     {
16         cout << "Number of iterations=";
17         cin >> n;
18     }
19     MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
20     h = 1.0 / n;
21     for (i = myrank + 1; i <= n; i += nprocs)
22         sum=sum+f(h * (i - 0.5));
23     term = 4 * h * sum;
24     MPI_Reduce(&term, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
25     if (myrank == 0)
26         cout << "Computed value of pi=" << pi << endl;
27     MPI_Finalize();
28     return 0;
29 }
```

Рисунок 8 - Вихідний код програми обчислення числа π

Наведемо коментарі для програмного коду, наведеному на рис.8.

Рядки 11, 12, 13 використовуються для виклику функцій MPI.

Розглянемо викликані тут функції детальніше.

Функція MPI_Init(&argc, &argv) ініціалізує бібліотеку MPI (її виклик обов'язковий перед тим, як розпочати виконання цієї бібліотеки); її параметри слугують для вибору з аргументів командного рядка тих, що відносяться до бібліотеки MPI.

Функція MPI_Comm_rank(MPI_COMM_WORLD, &myrank) визначає номер процесу (він присвоюється змінній myrank) у групі з комунікатором MPI_COMM_WORLD. Комунікатор - це певний спеціальний об'єкт, що ідентифікує групу, тобто сукупність процесів. Цей комунікатор визначає групу всіх процесів, які запущені для розв'язання даної задачі.

Процеси у будь-якій групі нумеруються цілими числами від 0 до (кількість процесів)-1.

Функція MPI_Comm_size(MPI_COMM_WORLD, &nprocs) дозволяє визначити кількість процесів у групі з комунікатором MPI_COMM_WORLD (це число присвоюється вихідному параметру nprocs). Воно визначається при запуску програми на виконання за допомогою опції – n p, де p – число процесів.

У рядках 14-18 здійснюється введення кількості n інтервалів, на які розбивається інтервал [0;1] при використанні квадратурної формули з процесу з номером 0 (нагадаємо, що в MPI привілейованого процесу не існує: роль кореневого процесу може виконувати будь-який процес за бажанням користувача).

У рядку 19 викликається функція MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD), яка здійснює розсилку інформації (у даному випадку числа n) з кореневого процесу (тобто процесу з номером 0) по всіх процесах групи з комунікатором MPI_COMM_WORLD; тут перший параметр - покажчик на значення, що розсилаються (у цьому випадку розсилається число n), другий параметр - кількість значень, що розсилаються (у даному випадку їх кількість дорівнює 1, оскільки розсилається одне число n), третій параметр визначає тип значення, що розсилається (тут має використовуватися тип, визначений у стандарті MPI: у нашому випадку це MPI_INT; тип має бути вказано, оскільки, по-перше, він визначає розмір відповідного елемента даних, і, по-друге, дозволяє правильно передавати дані зазначеного типу між комп’ютерами, що використовують різні представлення для елементарних типів даних). Четвертий параметр – це номер процесу, що розсилає повідомлення (у нашому випадку його номер дорівнює 0), п'ятий параметр - ім'я комунікатора групи (оскільки у даному випадку, використовується лише вихідна група паралельних процесів, то тут має стояти ім'я вихідного комунікатора, а саме MPI_COMM_WORLD.

У 20-му рядку обчислюється число h. Зауважимо, що оскільки програма копіюється в усі паралельні процеси, то кожен процес має свій екземпляр змінної h. Таким чином, значення h обчислюється незалежно в кожному процесі.

Далі (див. рядки 21, 22) кожен процес обчислює частину квадратурної суми, виділяючи з суми доданки, номери яких можна порівняти за модулем nprocs з його номером (нагадаємо, що nprocs - загальне число процесів).). Завдяки такому розподілу, кількості доданків, що обчислюються кожним процесом, добре збалансовані (вони відрізняються один від одного не більше, ніж на один доданок). Таким чином кожен процес заповнює свою копію змінної sum певною частиною всієї суми, яку необхідно було обчислити.

У 23-му рядку отримана процесом сума множиться на 4h; таке множення дає змогу своєчасно "нормалізувати" результат обчислень у даному процесі для того, щоб, наскільки це можливо, він перебував у середині діапазону представлення чисел із плаваючою крапкою (після додавання всіх отриманих сум порушення цього правила, можливо, призвело б до погіршення точності обчислень завдяки зсуву до краю згаданого діапазону). У кожному процесі результат отримують у належній йому змінній term.

Рядок 24 призначений для додавання (отриманих процесами значень змінних з ім'ям term) частин суми, знайдених процесами, за допомогою процедури MPI_Reduce(&term, &pi, l, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD), де

перший параметр являє собою адресу оброблюваних змінних (у нашому випадку адреса term),

другий параметр є адресою змінної, якій присвоюється результат обробки (у цьому випадку, адресою змінної pi),

третій параметр - кількість переданих даних із кожного процесу (у цьому випадку передається по одному даному - по доданку term),

четвертий параметр визначає тип даних, що піддаються обробці (у цьому випадку MPI_DOUBLE),

п'ятий параметр визначає операцію обробки даних (у нашому випадку це операція додавання, що ідентифікується константою MPI_SUM).

Рядки 25-26 містять виведення отриманого значення π. У рядку 27 міститься вивід функції MPI_Finalize(), чим завжди завершується робота з бібліотекою MPI, а рядки 28, 29 містять стандартне завершення функції main.

Результати виконання завдання 5 наведенні на рис.9.

```
Number of iterations=1000
Computed value of pi=3.14159
```

Рисунок 9 - Результат обчислення числа π за допомогою функцій MPI

2.6 Приклад розв'язання завдання 6

Завдання 6. Написати програму знаходження добутку двох матриць з використанням директиви OMP #pragma omp for. Підрахувати час виконання програми для різної кількості процесів.

Розв'язання. На рис. 10 наведена реалізація програми з використанням технології OpenMP.

```
#include <stdio.h>
#include <omp.h>
#include <cstdlib>
#define N 1024
void main()
{
    int i, j, k, size;
    double t1, t2, s;
    /* Виділення  па'мяти під матриці */
    double** a = (double**)malloc(sizeof(double*) * N);
    for (i = 0; i < N; i++)
        a[i] = (double*)malloc(sizeof(double) * N);
    double** b = (double**)malloc(sizeof(double*) * N);
    for (i = 0; i < N; i++)
        b[i] = (double*)malloc(sizeof(double) * N);
    double** c = (double**)malloc(sizeof(double*) * N);
    for (i = 0; i < N; i++)
        c[i] = (double*)malloc(sizeof(double) * N);
    // Ініціалізациія матриць
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++) {
            a[i][j] = (i + j) * 0.0001e0;
            b[i][j] = (i - j) * 0.0001e0;
            c[i][j] = 0.0e0;
        }
    t1 = omp_get_wtime();
    // Основний обчислювальний блок
#pragma omp parallel shared(a, b, c) private(i, j, k)
    {
        size = omp_get_num_threads();
#pragma omp for schedule(static)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                for (k = 0; k < N; k++) c[i][j] += a[i][k] * b[k][j];
    }
    t2 = omp_get_wtime();
    /* Обчислення контрольної сумми і друк результату */
    for (i = 0, s = 0.0e0; i < N; i++)
        for (j = 0; j < N; j++)
            s += c[i][j];
    printf("N= %d, Nproc=%d, Sum=%lf, Time=%lf\n",
        N, size, s, t2 - t1);
}
```

Рисунок 10 - Паралельна програма знаходження добутку двох матриць з використанням технології OpenMP

Наведемо коментарі до програмного коду, наведеного на рис. 10.

В рядках 10-18 здійснюється динамічне виділення пам’яті для матриць a, b с розмірністю NxN. Для виділення пам’яті використовується функція malloc. В рядку 10 здійснюється виділення пам’яті для масиву показчиків на дані типу double. Функція malloc(sizeof(double*) * N) виділяє блок пам'яті розміром N покажчиків на тип даних double. Конструкція (double**) використовується для перетворення типу, щоб результат функції malloc мав тип double**. Далі в циклі виділяється пам’ять для кожного рядка матриці a. Кожен елемент масиву a[i] посилається на комірку пам’яті, виділену для і-го рядка матриці a.

В рядках 20-25 відбувається присвоювання значень елементам матриць a, b та c у подвійному циклі. Матриця c призначена для зберігання результату множення матриць a та b. Початкове значення елементів матриці C дорівнює 0. В рядку 26 запам’ятовуємо час початку обчислень в змінній t1.

Рядок 28 містить директиву #pragma omp parallel. Це означає, що наступний блок коду буде виконуватися паралельно. Змінні a, b та c є загальними для всіх створених паралельний потоків, змінні i, j та k є приватними для кожного потоку, тобто кожний потік має приватні значення цих змінних. В рядку 30 змінній size присвоюється значення функції omp_get_num_threads(), яка повертає кількість потоків в паралельній області. Рядок 31 містить директиву #pragma omp for, яка вказує компілятору, що наступний цикл треба розпаралелити. Опція schedule(static) означає, що всі ітерації будуть рівномірно розподілені між потоками. Рядки 32-34 містять обчислення добутку двох матриць за формулою

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \;\; i = 1, \dots, n, j = 1, \dots, n.$$

В рядку 36 запам’ятовуємо час закінчення обчислень в змінній t2. В рядках 38-40 отримаємо контрольну суму – суму елементів результуючої матриці. В рядках 41-42 друкуємо розмірність матриць, число паралельних потоків, знайдену контрольну суму та час обчислення.

Результати виконання завдання 6 наведені на рис.11.

```
N= 1024, Nproc=4, Sum=938249.027584, Time=1.778955
```

Рисунок 11 – Час обчислення добутку двох матриць розміром 1024x1024 для 4 потоків

На рис.12 наведено результати обчислення для 4-х потоків.

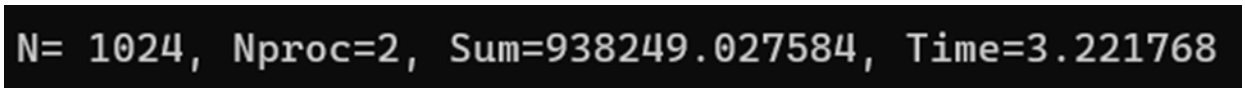


Рисунок 12 – Час обчислення добутку двох матриць розміром 1024x1024 для 2 потоків

Результати обчислення при відсутності розпаралелювання наведено на рис.13

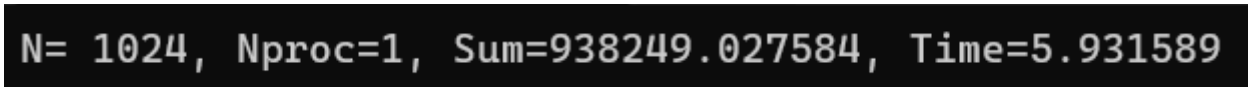


Рисунок 13 – Час обчислення добутку двох матриць розміром 1024x1024 при відсутності розпаралелювання

Аналізуючи рис.11-13, приходимо до висновку, що при збільшенні числа потоків з 1 до 4-х час виконання розрахунків зменшується з 5.93с до 1.78с, тобто спостерігається прискорення обчислень у 3,33 рази.

2.7 Приклад розв'язання завдання 7

Завдання 7. Використовуючи Azure Synapse Analytics, виконайте наступні дії:

1. Створіть робочу область Azure Synapse Analytics.
2. Ознайомтеся з можливостями Synapse Studio.
3. Створіть конвеєр (pipeline) і за його допомогою отримайте дані.
4. Проаналізуйте отримані дані, використовуючи безсерверний пул.

Розв'язання.

1. Робоче середовище Azure Synapse Analytics забезпечує централізоване керування даними та середовищами їх обробки. Ви можете підготувати робочу область за допомогою інтерактивного інтерфейсу на порталі Azure або розгорнути робочу область і ресурси в ній за допомогою сценарію або шаблону. У більшості виробничих сценаріїв найкраще автоматизувати ініціалізацію за допомогою сценаріїв і шаблонів, щоб ви могли включити розгортання ресурсів у повторюваний процес розробки та операцій (DevOps).

Виконайте наступні дії:

- У браузері увійдіть на портал Azure за адресою https://portal.azure.com.

- За допомогою кнопки [>_] праворуч від рядка пошуку у верхній частині сторінки створіть нову хмарну оболонку на порталі Azure, вибравши середовище PowerShell і створивши сховище, якщо з'явиться відповідний запит. Хмарна оболонка надає інтерфейс командного рядка на панелі в нижній частині порталу Azure, як показано тут(рис.14):

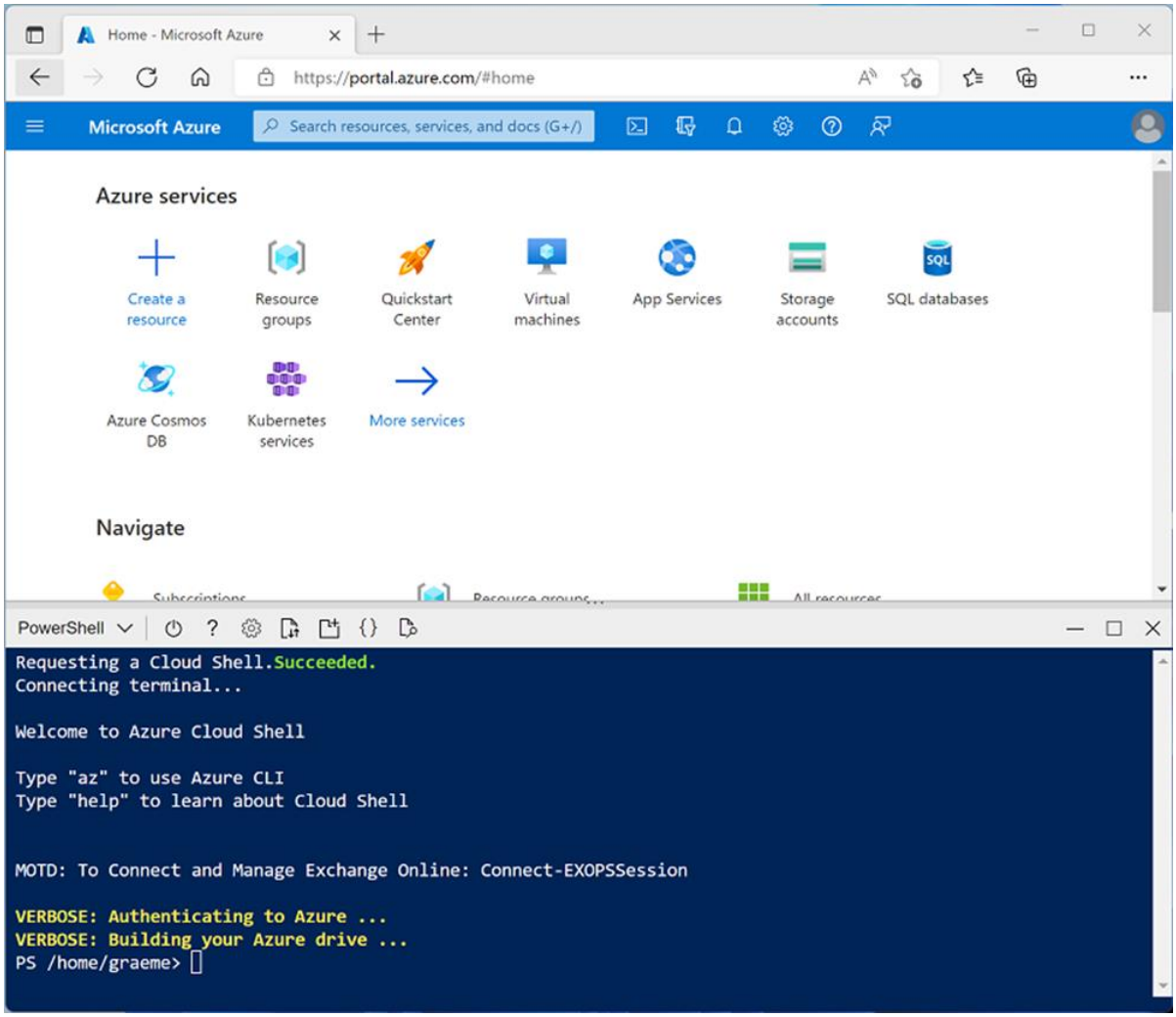


Рисунок 14 – Домашня сторінка порталу Azure з запущеним PowerShell

Зауважте, що розмір оболонки хмари можна змінити, перетягнувши смугу роздільника у верхній частині панелі або скориставшись піктограмами —, ☐ і X у верхньому правому куті панелі для згортання, розгортання та закриття панелі. Для отримання додаткових відомостей про використання Azure Cloud Shell перегляньте документацію Azure Cloud Shell.

- На панелі PowerShell введіть такі команди, щоб клонувати цей репозиторій:

```
rm -r dp-203 -f
git clone https://github.com/MicrosoftLearning/dp-203-azure-data-engineer dp-203
```

- Після того, як репозиторій буде клоновано, введіть наступні команди, щоб перейти до папки для цього завдання, і запустіть сценарій setup.ps1, який він містить:

```
cd dp-203/Allfiles/labs/01
./setup.ps1
```

- Якщо з'явиться відповідний запит, виберіть, яку передплату ви хочете використовувати (це станеться, лише якщо у вас є доступ до кількох підписок Azure).

- Коли з'явиться запит, введіть відповідний пароль, який буде встановлено для пулу SQL Azure Synapse.

Примітка: Обов'язково запам'ятайте цей пароль! Крім того, пароль не може містити повністю або частково ім'я для входу.

- Дочекайтеся завершення сценарію - зазвичай це займає близько 20 хвилин, але в деяких випадках може зайняти більше часу.

2. Synapse Studio – це веб-портал, на якому ви можете керувати ресурсами в робочій області Azure Synapse Analytics і працювати з ними.

Коли сценарій інсталяції завершить виконання, на порталі Azure перейдіть до створеної ним групи ресурсів dp203-xxxxxxx і зверніть увагу, що ця група ресурсів містить робоче середовище Synapse, обліковий запис сховища для озера даних, пул Apache Spark, пул Data Explorer і виділений пул SQL.

Виберіть робоче середовище Synapse і на сторінці «Огляд» на картці «Відкрити Synapse Studio» виберіть «Відкрити», щоб відкрити Synapse Studio в новій вкладці браузера. Synapse Studio — це веб-інтерфейс, який можна використовувати для роботи з робочим простором Synapse Analytics.

У лівій частині Synapse Studio за допомогою піктограми ›› розгорніть меню - відкриються різні сторінки в Synapse Studio, які ви будете використовувати для керування ресурсами та виконання завдань аналізу даних, як показано тут(рис.15):

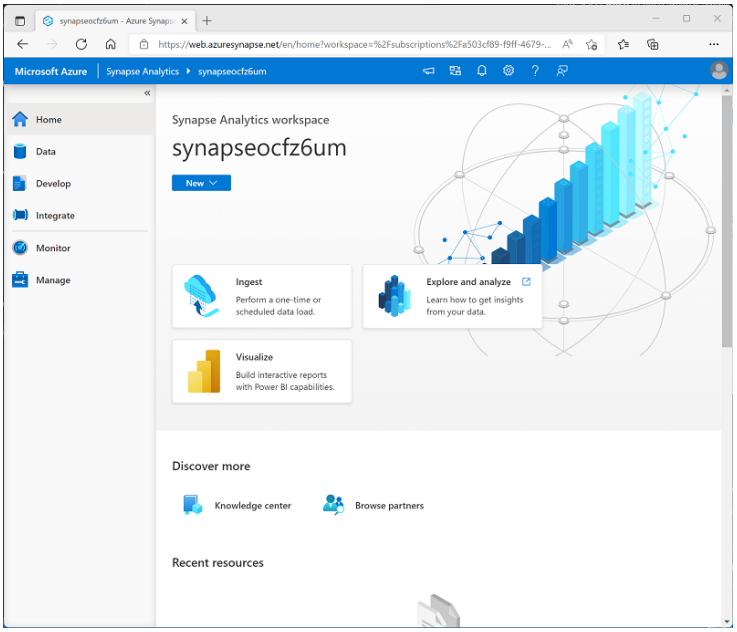


Рисунок 15 – Synapse Studio

Перегляньте сторінку Дані та зверніть увагу, що є дві вкладки, які містять джерела даних:

- Вкладка «Робоча область», що містить бази даних, визначені в робочій області (включно зі спеціальними базами даних SQL і базами даних Провідника даних)
- Зв'язана вкладка, що містить джерела даних, пов'язані з робочою областю, зокрема сховище Azure Data Lake.

Перегляньте сторінку «Розробка», яка наразі порожня. Тут ви можете визначити сценарії та інші ресурси, які використовуються для розробки рішень для обробки даних.

Перегляньте сторінку Integrate, яка також порожня. Ви використовуєте цю сторінку для керування отриманням даних та інтеграційними ресурсами; Наприклад, конвеєри для передачі та перетворення даних між джерелами даних.

Перегляньте сторінку «Монітор». Тут ви можете спостерігати за завданнями з обробки даних під час їх виконання та переглядати їхню історію.

Перегляньте сторінку Керування. Тут ви керуєте пулами, середовищами виконання та іншими ресурсами, які використовуються у вашому робочому середовищі Azure Synapse. Перегляньте кожну вкладку в розділі Пули Analytics і зверніть увагу, що ваша робоча область містить такі пули:

Пули SQL:

Вбудований: безсерверний пул SQL, який можна використовувати на вимогу для дослідження або обробки даних в озері даних за допомогою команд SQL.

sqlxx: Виділений пул SQL, який розміщує базу даних сховища реляційних даних.

Пули Apache Spark:

sparkxxxxxxx: що ви можете використовувати за запитом для дослідження або обробки даних в озері даних за допомогою мов програмування, таких як Scala або Python. <!— - Пули Data Explorer:

adxxxxxxxx: Пул Data Explorer, який можна використовувати для аналізу даних за допомогою Kusto Query Language (KQL). —>

Отримуйте дані за допомогою конвеєра

Одним із ключових завдань, які ви можете виконати за допомогою Azure Synapse Analytics, є створення конвеєрів, які передають (і, за необхідності, перетворюють) дані з широкого кола джерел у ваш робочий простір для аналізу.

3. У Synapse Studio на головній сторінці виберіть Ingest, щоб відкрити інструмент «Копіювати дані»

В інструменті «Копіювати дані» на кроці «Властивості» переконайтеся, що вибрано «Вбудоване завдання копіювання» та «Запустити один раз», і натисніть кнопку «Далі» >.

На кроці «Джерело» на підкроці «Набір даних» виберіть такі параметри:

Source type: All

Connection: Створіть нове підключення та на панелі Linked Service, що з'явиться, на вкладці Generic protocol виберіть HTTP. Потім продовжуйте та створіть підключення до файлу даних, використовуючи такі параметри:

Name: Products

Description Список продуктів через HTTP

Connect via integration runtime: AutoResolveIntegrationRuntime

Base URL: https://raw.githubusercontent.com/MicrosoftLearning/dp-203-azure-data-engineer/master/Allfiles/labs/01/adventureworks/products.csv

Server Certificate Validation: Enable

Authentication type: Anonymous

Після створення підключення на сторінці Сховище вихідних даних переконайтеся, що вибрано такі параметри, а потім натисніть Далі >:

Relative URL: Залиште порожнім

Request method: GET

Additional headers: залишити порожнім

Binary copy: Невиділений

Request timeout: Залиште порожнім

Max concurrent connections: Залиште порожнім

На кроці Джерело на підкроці Конфігурація виберіть Попередній перегляд даних, щоб переглянути дані про товари, які отримає конвейєр, а потім закрийте попередній перегляд.

Після попереднього перегляду даних на сторінці Настройки формату файлу переконайтеся, що вибрано такі параметри, а потім натисніть кнопку Далі >:

File format: Текст з роздільниками

Column delimiter: Кома (,)

Row delimiter: Перенос рядка (\n)

First row as header: Вибрати

Compression type: None

На кроці «Призначення» на підкроці «Набір даних» виберіть наведений нижче параметр

Destination type: Azure Data Lake Storage Gen 2

Connection: Виберіть наявне підключення до сховища озера даних (його було створено для вас під час створення робочої області).

After selecting the connection, on the Destination/Dataset step, ensure the following settings are selected, and then select Next >:

Folder path: files/product_data

File name: products.csv

Copy behavior: None

Max concurrent connections: Залиште порожнім

Block size (MB): Leave blank

На кроці Призначення, на підкроці Конфігурація, на сторінці Параметри формату файлу переконайтеся, що вибрано такі властивості. Потім виберіть Далі >:

File format: Текст з роздільниками

Column delimiter: Кома (,)

Row delimiter: Line feed (\n)

Add header to file: Selected

Compression type: None

Max rows per file: Leave blank

File name prefix: Leave blank

On the Settings step, enter the following settings and then click Next >:

Task name: Copy products

Task description Copy products data

Fault tolerance: Leave blank

Enable logging: Unselected

Enable staging: Unselected

На кроці Перегляд і завершення на підкроці Рецензування прочитайте підсумок і натисніть кнопку Далі >.

На кроці розгортання зачекайте, доки конвеєр буде розгорнуто, а потім натисніть кнопку Готово.

У Synapse Studio виберіть сторінку «Монітор» і на вкладці «Pipeline runs» зачекайте, поки конвеєр «Копіювати продукти» завершиться зі статусом «Успішно виконано» (ви можете скористатися кнопкою ↻ «Оновити» на сторінці «Запуск конвеєра», щоб оновити статус).

Перегляньте сторінку «Інтеграція» та переконайтеся, що вона містить конвеєр із назвою «Копіювати продукти».

На сторінці «Дані» виберіть вкладку «Linked» та розгорніть ієрархію контейнерів synapsexxxxxxx (Primary), доки не побачите сховище файлів для робочого середовища Synapse. Потім виберіть файлове сховище, щоб переконатися, що папка з іменем product_data містить файл з назвою products.csv була скопійована в це місце, як показано на рис. 16.

Клацніть правою кнопкою миші файл даних .csv продуктів і виберіть Попередній перегляд, щоб переглянути отримані дані. Потім закрийте попередній перегляд.

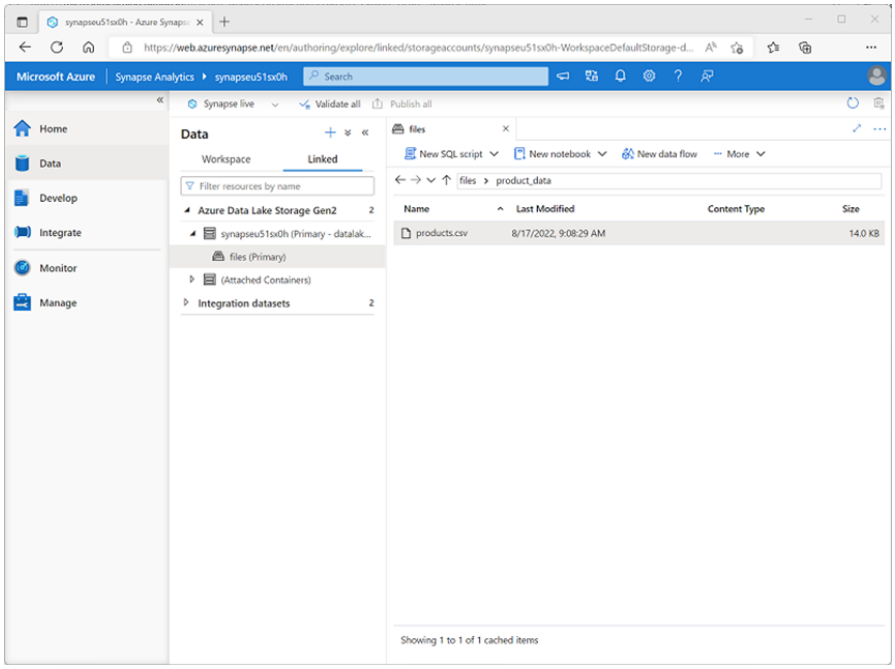


Рисунок 16 – Перегляд отриманих даних.

4. Тепер, коли ви завантажили деякі дані у свій робочий простір, ви можете використовувати Synapse Analytics для їх запиту та аналізу. Одним із найпоширеніших способів запиту даних є використання SQL, а в Synapse Analytics ви можете використовувати безсерверний пул SQL для запуску SQL-коду з даними в озері даних.

У Synapse Studio клацніть правою кнопкою миші файл продуктів.csv у файловому сховищі робочого середовища Synapse, наведіть вказівник миші на пункт Новий сценарій SQL і виберіть пункт Вибрати ТОП-100 рядків.

На панелі SQL Script 1, що відкриється, перегляньте згенерований SQL-код, який має бути подібним до цього:

```
- This is auto-generated code
SELECT
    TOP 100 *
FROM
    OPENROWSET(
        BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/product_data/products.csv',
            FORMAT = 'CSV',
            PARSER_VERSION='2.0'
    ) AS [result]
```

Цей код відкриває набір рядків з імпортованого текстового файлу та отримує перші 100 рядків даних.

У списку **Connect to** переконайтеся, що вибрано пункт **Built-in** - це відповідає вбудованому пулу SQL, створеному у вашому робочому середовищі.

На панелі інструментів скористайтеся кнопкою ▷ **Run**, щоб

запустити SQL-код і переглянути результати, які мають виглядати приблизно так (табл.1):

Таблиця 1 – Результати виконання SQL-коду

C1	C2	C3	C4
ProductID	ProductName	Category	ListPrice
771	Mountain-100 Silver, 38	Mountain Bikes	3399.9900
772	Mountain-100 Silver, 42	Mountain Bikes	3399.9900

Зверніть увагу, що результати складаються з чотирьох стовпців з назвами C1, C2, C3 і C4; і що перший рядок результатів містить імена полів даних. Щоб вирішити цю проблему, додайте параметри HEADER_ROW = TRUE до функції OPENROWSET, як показано тут (замінивши datalakexxxxxxx на ім'я вашого облікового запису сховища озера даних), а потім повторно запустіть запит:

```
SELECT
    TOP 100 *
FROM
    OPENROWSET(
        BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/product_data/products.csv',
            FORMAT = 'CSV',
            PARSER_VERSION='2.0',
            HEADER_ROW = TRUE
    ) AS [result]
```

Тепер результати виглядають так(табл.2):

Таблиця 2 – Результати запиту

ProductID	ProductName	Category	ListPrice
771	Mountain-100 Silver, 38	Mountain Bikes	3399.9900
772	Mountain-100 Silver, 42	Mountain Bikes	3399.9900

Змініть запит наступним чином (замінивши datalakexxxxxxxxx на назву вашого облікового запису в сховищі даних):

```
SELECT
    Category, COUNT(*) AS ProductCount
```

```
    FROM
        OPENROWSET(

        BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/product_data/produ
cts.csv',
            FORMAT = 'CSV',
            PARSER_VERSION='2.0',
            HEADER_ROW = TRUE
        ) AS [result]
    GROUP BY Category;
```

Запустіть модифікований запит, який повинен повернути набір результатів, що містить кількість продуктів у кожній категорії, наприклад:

Таблиця 3 – Результати модіфікованого запиту

Category	ProductCount
Bib Shorts	3
Bike Racks	1
...	...

На панелі «Властивості» для SQL Script 1 змініть ім'я на «Кількість продуктів за категоріями». Потім на панелі інструментів виберіть **Publish**, щоб зберегти сценарій.

Закрийте панель скриптів **Count Products by Category**.

У Synapse Studio виберіть сторінку « **Develop**» та зверніть увагу, що там збережено опублікований сценарій SQL « **Count Products by Category** ».

Виберіть сценарій SQL « **Count Products by Category** », щоб знову відкрити його. Потім переконайтеся, що скрипт підключено до вбудованого пулу SQL, і запустіть його, щоб отримати кількість продуктів.

В області Результати виберіть подання **Діаграма**, а потім виберіть такі настройки для діаграми

Chart type: Column
Category column: Category
Legend (series) columns: ProductCount
Legend position: bottom - center
Legend (series) label: Leave blank
Legend (series) minimum value: Leave blank
Legend (series) maximum: Leave blank
Category label: Leave blank
Отримана діаграма повинна нагадувати таку:

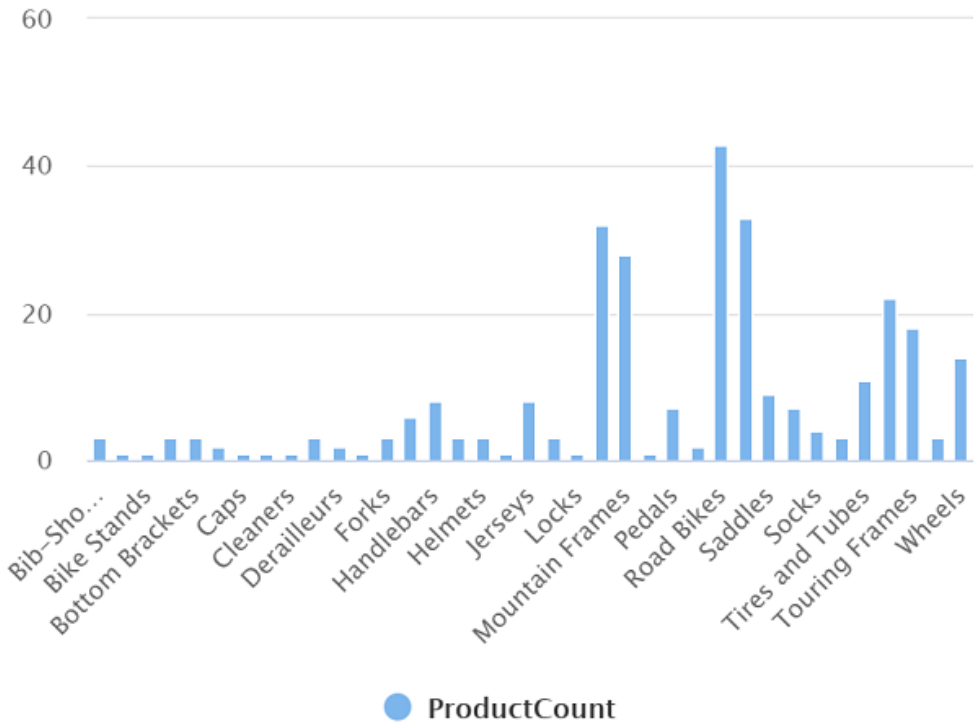


Рисунок 17 – Розподіл продуктів за категоріями

2.8 Приклад розв'язання завдання 8

Завдання 8. На хмарній платформі OpenStack створити топологію, використовуючи команди Neutron CLI.

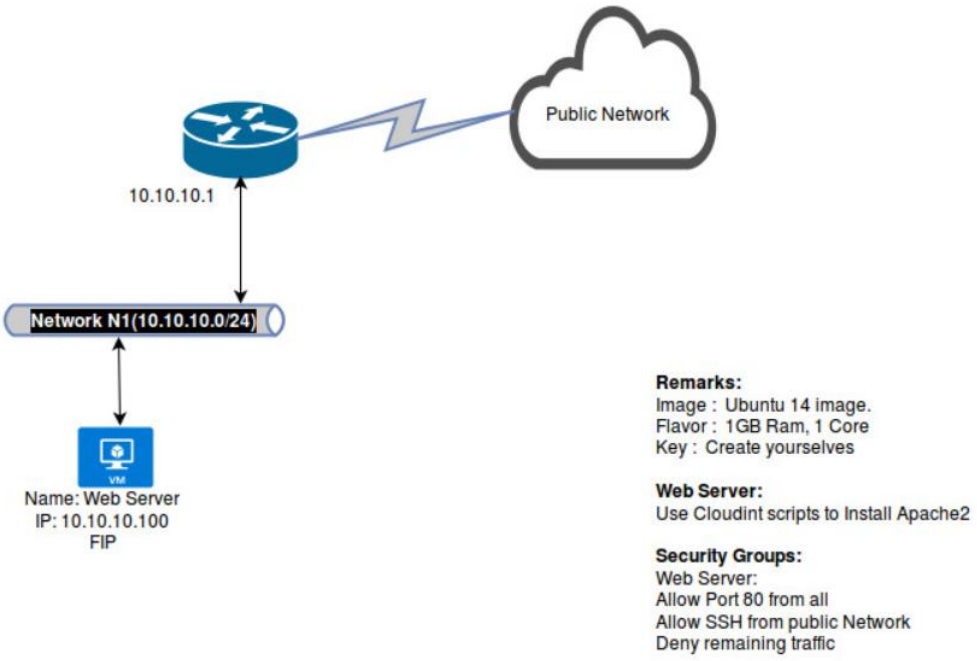


Рисунок 18 - Топологія

Ключові моменти :

Створіть хмарний ініціалізаційний скрипт, який буде використовуватися для встановлення веб-сервера

Напишіть групи безпеки відповідно до умов

Тести, які потрібно зробити:

З загальнодоступної мережі, ping веб-сервер - він повинен зазнати невдачі

З загальнодоступної мережі, http запит до веб-сервера - він повинен пройти

З публічної мережі, SSH до FIP веб-сервера - має пройти

Розв'язання

1. Створення мережі і роутера.

Створити мережу:

```
neutron net-create N1
```

Створення підмережі

```
neutron subnet-create --name S1 --gateway 10.10.10.1 --allocation-pool start=10.10.10.10,end=10.10.10.50 N1 10.10.10.0/24
```

Створення роутера

```
neutron router-create R1
```

Налаштування шлюзу до маршрутизатора

```
neutron router-gateway-set R1 public
```

Підключення підмережі до маршрутизатора

```
neutron router-interface-add R1 S1
```

2. Створення групи безпеки(Security Group)

Створення групу безпеки з назвою "webtraffic"

```
neutron security-group-create webtraffic
```

створення правила безпеки для дозволу порту 80 (HTTP) з будь-якої мережі

```
neutron security-group-rule-create --direction ingress --protocol tcp --port-range-min 80 --port-range-max 80 --remote-ip-prefix 0.0.0.0/0 webtraffic
```

створення правила безпеки для дозволу порту 22(ssh) з публічної (зовнішньої) мережі

Примітка: У моєму середовищі загальнодоступною мережею є 172.24.4.0/24

```
neutron security-group-rule-create --direction ingress --protocol tcp --port-range-min 22 --port-range-max 22 --remote-ip-prefix 172.24.4.0/24 webtraffic
```

3. Створення віртуальної машини(VM)

Створення скрипут для встановлення сервера apache2.

Запишіть наведений нижче bash-скрипт у файл (bootscript.sh). Зробіть його виконуваним.

```
#!/bin/bash
```

```
sudo apt-get update
sudo apt-get -y install apache2
sudo a2enmod ssl
sudo a2ensite default-ssl
sudo service apache2 restart
echo `hostname` | sudo tee /var/www/html/index.html
```

Створення порту з фіксованою IP-адресою 10.10.10.100/24 з мережі N1

```
neutron port-create --fixed-ip subnet_id=S1,ip_address=10.10.10.100 --security-group webtraffic N1
```

Створення віртуальної машини

Перевірка конфігурації, зображення, ключової пари

```
glance image-list
nova keypair-list
nova flavor-list
```

Примітка : Flavor : 6, keypair: testkey, image : ubuntu14, security-group : webtraffic, NIC : Port ID of 10.10.10.100

```
nova boot --flavor 6 --image ubuntu14 --key-name testkey --security-groups webtraffic --user-data bootscript.sh --nic port-id=297e952a-f491-40fc-b015-94e96f6ea864 Web-Server
```

Зв'яжіть вільний плаваючий IP з ідентифікатором порту 10.10.10.100:

```
neutron floatingip-associate dcb48f7f-cb4c-4289-a27b-62b4e3399647 297e952a-f491-40fc-b015-94e96f6ea864
```

Тестування:

HTTP-запит до веб-сервера з плаваючим IP із загальнодоступної (зовнішньої) мережі

```
curl 172.24.4.12
```

Повинна повернути web-сервер

Пінг веб-сервера з плаваючою IP-адресою із загальнодоступної (зовнішньої) мережі

```
ping 172.24.4.12
```

Він повинен не пройти.

SSH до веб-сервера з плаваючою IP-адресою з публічної (зовнішньої) мережі

```
ssh -i ../testkey ubuntu@172.24.4.12
```

Він повинен дозволяти входити на веб-сервер.

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Хмарні технології : навчальний посібник / О. В. Зінченко иа ін. Київ : ФОП Гуляєва В. М., 2020. 74 с.
2. Ількевич Н. С. Хмарні технології в освіті : навчально-методичний посібник для студентів фізико-математичного факультету. Житомир : Вид-во ЖДУ, 2021. 88 с.
3. Van Steen M., Tanenbaum A. S. Distributed systems. 4th edition. Pearson Prentice Hall, 2023. 686 p.
4. Хмарні та Грід-технології : конспект лекцій / уклад. В. Я. Юрчишин. Київ : КПІ ім. Ігоря Сікорського, 2019. 264 с.
5. BOINC – Berkeley Open Infrastructure for Network Computing : веб-сайт. URL: http://boinc.berkeley.edu (дата звернення: 25.02.2025).
6. Miller R. Who Has the Most Web Servers? : Data Center Knowledge. URL: http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-mostweb-servers (дата звернення: 25.02.2025).
7. PaaS, DBaaS, SaaS : Habr. URL: https://habr.com/ru/company/kingservers/blog/310022/ (дата звернення: 25.02.2025).
8. Wadiwala R. Cloud Database - DBaaS (Database as a Service) : SogetiLabs. URL: https://labs.sogeti.com/cloud-database-dbaas-database-as-a-service (дата звернення: 25.02.2025).
9. Hamza Y. A., Omar M.D. Cloud computing security: Abuse and nefarious use of cloud computing. International Journal of Computational Engineering Research, 2013. Vol. 03. Issue 6. P. 22-27.
10. Reference Model Public Review Draft 1.0(Feb) : Organization for the Advancement of Structured Information Standards (OASIS). URL: http://www.oasisopen.org/committees/download.php/16587/wdsoa-cd1ED.pdf
11. Cloud Computing Technology. Singapore : Springer Nature, 2022. 315 p. URL: https://read.kortext.com/library/books/2046241
12. Lynch N. A. Distributed Algorithms. Morgan Kaufmann Publishers, 1996. 899 p. URL: https://read.kortext.com/library/books/74770
13. Models and Analysis for Distributed Systems / S. Haddad et al. 1st Edition. John Wiley & Sons, 2013. 358 p. URL: https://read.kortext.com/library/books/905896
14. Kostikov A. A., Zaitsev N. D., Subotin O. V. Realisation of the double sweep method by using a Sleptsov net. International Journal of Parallel, Emergent and Distributed Systems. 2021. Vol. 36, Issue 6. P. 516–534. DOI:10.1080/17445760.2021.1945054.
15. Shmeleva T., Kostikov A. Verification of Square Lattices with Dedicated Channels by Infinite Petri Nets. IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T)2020. Kharkiv, Ukraine, 06-09 October 2020. Kharkiv, 2020. P. 388–392. DOI:

10.1109/PICST51311.2020.9468059.
16. Концепції хмарних обчислень. Частина 1 : Coursera : веб-сайт. URL: https://www.coursera.org/learn/cloud-computing (дата звернення: 25.02.2025).
17. Концепції хмарних обчислень. Частина 2 : Coursera : веб-сайт. URL: https://www.coursera.org/learn/cloud-computing-2 (дата звернення: 25.02.2025).
18. Основи хмарних обчислень (Хмара 101) : Coursera : веб-сайт. URL: https://www.coursera.org/learn/cloud-computing-basics (дата звернення: 25.02.2025).
19. Розподілені обчислення зі Spark SQL : Coursera : веб-сайт. URL: https://www.coursera.org/learn/spark-sql#modules (дата звернення: 25.02.2025).
20. Вступ до хмарних служб Microsoft Azure : Coursera : веб-сайт. URL: https://www.coursera.org/learn/microsoft-azure-cloud-services (дата звернення: 25.02.2025).
21. Open Source Cloud Computing Infrastructure : OpenStack : веб-сайт. URL: https://www.openstack.org/ (дата звернення: 25.02.2025).
22. Kortext : веб-сайт. URL: https://kortext.com/ (дата звернення: 20.08.2024).
23. Research4life : веб-сайт. URL: https://portal.research4life.org/ (дата звернення: 20.08.2024).
24. Інституційний репозитарій ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА» : веб-сайт. URL: https://dspace.mipolytech.education/home (дата звернення: 20.08.2024).
25. Центральна державна науково-технічна бібліотека гірничо-металургійного комплексу України : веб-сайт. URL: http://cgntb.dp.ua/ (дата звернення: 20.08.2024).

Навчально-методичне видання

Костіков Олександр Анатолійович

Сагайда Павло Іванович

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ ТА ХМАРНІ ТЕХНОЛОГІЇ

методичні рекомендації до виконання індивідуальних завдань

Самостійне електронне мережеве видання
Публікується в авторській редакції