НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.65; 004.75

До захисту допущено
Завідувач кафедри ММСА

Оксана ТИМОЩУК

«___» _____ 2022 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 124 «Системний аналіз»
на тему: «Моделі та методи організації розподілених баз даних з динамічною структурою»

Виконав:
студент 2 курсу, групи КА-13мп
Васильченко Ілля Вячеславович _____

Керівник:
професор кафедри ММСА
д-р. техн. наук, професор Мухін Вадим Євгенович _____

Рецензент:
декан ФІОТ
д-р техн. наук, Корнага Ярослав Ігорович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань
Студент (*підпис*): _____

Київ-2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність (ОПП) — 124 «Системний аналіз» («Системний аналіз фінансового ринку»)

ЗАТВЕРДЖУЮ
Завідувач кафедри ММСА
Оксана ТИМОЩУК
«___» _____ 2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту Васильченку Іллі Вячеславовичу

1. Тема дисертації: «Моделі та методи організації розподілених баз даних з динамічною структурою», науковий керівник дисертації Мухін Вадим Євгенович, професор кафедри ММСА, д-р. техн. наук, затверджені наказом по університету від «03» листопада 2022 р. № 4046-с

2. Термін подання студентом дисертації: 12.12.2022 р.

3. Об’єкт дослідження: моделі та методи розподілених баз даних.

4. Предмет дослідження: пошук нових моделей та методів організації та оцінки роботи розподілених баз даних.

5. Перелік завдань, які потрібно розробити:
1) здійснити огляд літератури за темою роботи;
2) дослідити актуальність обраної теми;
3) проаналізувати існуючі моделі та методи організації структур розподілених баз даних;
4) розробити та реалізувати систему, що моделює розподілену базу даних
5) провести аналіз результатів;
6) оформити пояснювальну записку.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
1) ознайомлення з розподіленими базами даних;
2) результати аналізу даних;

3) розрахункові таблиці з результатами.

7. Орієнтовний перелік публікацій:

1. Васильченко І.В, Мухін В.Є. Моделі та методи організації розподілених баз даних з динамічною структурою. 1-а Всеукраїнська Науково-практична конференція "Системний аналіз та інформатика", м. Київ, 22-29 листопада, 2022 року.

8. Дата видачі завдання: 01.09.2022

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.09.2022	Виконано
2	Збір інформації	12.09.2022	Виконано
3	Ознайомлення з літературою і підготовка теоретичної частини магістерської дисертації	26.09.2022	Виконано
4	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	07.10.2022	Виконано
5	Розробка програмного продукту	21.10.2022	Виконано
6	Проведення аналізу ринкових можливостей стартап-проекту	27.10.2022	Виконано
7	Отримання висновків після тестування	07.11.2022	Виконано
8	Оформлення магістерської дисертації	09.12.2022	Виконано
9	Отримання допуску до захисту та подача роботи до ДЕК	16.12.2022	Виконано

Студент Ілля ВАСИЛЬЧЕНКО

Науковий керівник дисертації Вадим МУХІН

РЕФЕРАТ

Магістерська дисертація: 104 с., 19 рис., 21 табл., 1 дод., 17 джерел.

МОДЕЛІ ТА МЕТОДИ ОРГАНІЗАЦІЇ РОЗПОДІЛЕНИХ БАЗ ДАНИХ З ДИНАМІЧНОЮ СТРУКТУРОЮ

Об’єкт дослідження – моделі та методи організації розподілених баз даних.

Мета роботи – проаналізувати існуючі моделі організації та оцінювання розподілених баз даних, розробити власну модель оцінювання розподілених баз даних.

Використані моделі – у розробці була використана модель оцінювання часових витрат для розподіленої бази даних.

Отримані результати – була побудована нова модель оцінювання часових витрат для розподіленої бази даних, що враховує навантаження системи користувачами. Також був створений програмний комплекс, що моделює архітектуру розподіленої бази даних з динамічною структурою, візуалізує граф моделі, та оцінює час обробки запиту в моделі.

В рамках подальшого дослідження пропонується враховувати інші різноманітні параметри системи, наприклад, моделі серверів, що обробляють дані.

ABSTRACT

Master's thesis: 104 p., 19 fig., 21 tabl., 1 append., 17 sources

MODELS AND METHODS OF ORGANIZING DISTRIBUTED DATABASES WITH DYNAMIC STRUCTURE

Object of study is the set of models and methods of organizing distributed databases.

Purpose - analyze the existing models of organization and assessment of distributed databases, to develop an own model of assessment of distributed databases.

Used models - a time cost estimation model for a distributed database was used in the development.

Results – a new model for estimating time costs for a distributed database, which takes into account the system load by users, was built. Also, a software complex was created that simulates the architecture of a distributed database with a dynamic structure, visualizes the graph of the model, and estimates the time of processing a request in the model.

Further research is proposed to take into account other various system parameters, for example, models of servers that process data.

ЗМІСТ

ВСТУП

У наш час бази даних відіграють ключову роль під час обробки великої кількості інформації. За їх допомогою інформацію можна зберігати, вилучати та обробляти, а також надавати різні рівні доступу певним користувачам для маніпуляції даними. Бізнеси, що мають хоча б якусь інформацію, наприклад, про продажі, клієнтів, товари тощо, зберігають її у різноманітних видах і формах: від паперових зошитів та блокнотів, до табличних процесорів та баз даних. Ті компанії, що бажають швидко та зручно отримувати інформацію про стан свого бізнесу, все частіше переходять до використання баз даних, чим пришвидшують обробку інформації та зменшують вірогідність помилки внаслідок втрати інформації, або людської помилки під час ручної маніпуляції з даними.

Через світовий прогрес інформація почала стрімко накопичуватися і тому почали використовувати розподілені бази даних, для швидкого та постійного доступу до даних.

Для прогнозування часу та ресурсів витраченого на роботу з даними, виникає потреба у прогнозуванні вищезазначених витрат на обробку інформації. Для цього створюються моделі, що допомагають оцінити витрати ресурсів, у тому числі й часу.

Окрім цього, навантаження системи у розподілених базах даних з динамічною структурою залежить від багатьох параметрів, наприклад, швидкість з’єднання, кількість вузлів, кількість активних користувачів, що працюють у системі тощо. Врахування усіх цих параметрів надає можливість спрогнозувати час обробки запиту в розподіленій базі даних.

Питання прогнозування часу обробки запиту в розподілених базах даних з динамічною структурою є актуальним, тому що надає можливість ефективно спланувати робочі процеси. А також, враховуючи той факт, що навантаженність системи користувачами впливає на час обробки запитів, то

можна розподілити навантаження таким чином, аби мінімізувати цей час, та оптимізувати ресурси системи. Окрім цього, у нас час, коли через війну зникає світло та багато людей працюють у поза робочий час, то навантаженність систем значно спадає, що дозволяє ефективніше та швидше оброблювати дані. Запропонована модель дозволяє це чисельно оцінити.

Наукова новизна полягає в тому, що запропоновано модифіковану модель оцінювання параметрів обробки інформації в розподілених системах з динамічною структурою, яка відрізняється врахуванням параметру навантаженості комп'ютерної системи, що дозволяє підвищити ефективність аналізу розподілу ресурсів компютерних систем в процесі обробки розподілених баз даних.

Мета даної роботи – це проаналізувати існуючі моделі організації та оцінювання розподілених баз даних, розробити власну модель оцінювання розподілених баз даних.

Предметом дослідження є розподілені бази даних з динамічною структурою.

Об’єктом дослідження є оцінювання параметрів обробки інформації в розподілених системах з динамічною структурою.

Буде проведено порівняльний аналіз організації розподілених баз даних з динамічною структурою, розглянуто методи та моделі організації та управління розподіленими базами даних, створено і описано програмний комплекс для підтримки ефективної обробки даних у розподілених баз даних з динамічною структурою та проведено експериментальний аналіз механізмів підтримки ефективної обробки даних у розподілених баз даних з динамічною структурою.

РОЗДІЛ 1. ПОРІВНЯЛЬНИЙ АНАЛІЗ ОРГАНІЗАЦІЇ РОЗПОДІЛЕНИХ БАЗ ДАНИХ З ДИНАМІЧНОЮ СТРУКТУРОЮ

1.1. Аналіз організації, обробки, зберігання даних у реляційних базах даних.

Система баз даних - це, по суті, не що інше, як комп'ютеризована система зберігання однотипних записів. Саму ж базу даних можна розглядати, як подібність електронної картотеки, тобто, сховище або контейнер для набору файлів даних, занесених до комп'ютера. [1]

Користувачам цієї системи надається можливість виконувати (або передавати системі запити на виконання) безліч різних операцій над такими файлами, наприклад [1]:

- додавати нові порожні файли до бази даних;
- вставляти нові дані у існуючі файли;
- отримувати дані з наявних файлів;
- видаляти дані з наявних файлів;
- змінювати дані у існуючих файлах;
- видаляти існуючі файли з бази даних.

Система управління базами даних складається з наступних компонентів: дані, апаратне забезпечення, програмне забезпечення, користувачі. Нижче буде розглянуто ці основні компоненти системи бази даних. [1]

Дані. Системи з базами даних застосовуються як на найменших комп'ютерах, так і на найбільших мейнфреймах. Безсумнівно, засоби, що надаються кожною конкретною системою, значною мірою залежать від потужності та можливостей базової машини. Зокрема, на великих обчислювальних машинах застосовуються в основному розраховані на багато користувачів системи ("великі системи"), а на малих комп'ютерах, як правило, - однокористувацькі системи ("малі системи"). Однокористувацька система (single-user system) - це система, в якій до бази даних може отримати доступ

одночасно тільки один користувач, а розрахована на багато користувачів система (multi-user system) - це така система, в якій до бази даних можуть отримати доступ відразу кілька користувачів. [1]

До апаратного забезпечення системи належить наступне:

- томи вторинної (зовнішньої) пам'яті (зазвичай це магнітні диски), які використовуються для зберігання інформації, а також відповідні пристрої введення-виводу (дисководи і т.п.), контролери пристроїв, канали введення-виводу тощо;
- апаратний процесор (або процесори) разом з оперативною (первинною) пам'яттю, призначені для підтримки роботи програмного забезпечення системи баз даних. [1]

Програмне забезпечення. Між власне фізичною базою даних (тобто даними, які реально зберігаються на комп'ютері) та користувачами системи розташовується рівень програмного забезпечення, який можна називати по-різному: диспетчер бази даних (database manager), сервер бази даних (database server) або що звичніше, система управління базами даних, СУБД (DataBase Management System — DBMS). Усі запити користувачів отримання доступу до бази даних обробляються СУБД. Всі наявні засоби додавання файлів (або таблиць), вибірки та оновлення даних у цих файлах або таблицях також надає СУБД. Основне завдання СУБД – дати користувачеві бази даних можливість працювати з нею, не вникаючи у всі подробиці роботи на рівні апаратного забезпечення. (Користувач СУБД більш відсторонений від цих подробиць, ніж прикладний програміст, що застосовує мовне середовище програмування.) Іншими словами, СУБД дозволяє кінцевому користувачеві розглядати базу даних як об'єкт вищого рівня в порівнянні з апаратним забезпеченням, а також надає в його розпорядження набір операцій, що виражаються у термінах мови високого рівня (наприклад, набір операцій, які можна виконувати за допомогою мови SQL). [1]

Необхідно відзначити ще дві описані нижче особливості.

- СУБД - це найважливіший, але не єдиний програмний компонент системи. Серед інших компонентів можна назвати утиліти, засоби розробки додатків, засоби проектування, генератори звітів та диспетчер транзакцій (transaction manager), або диспетчер обробки транзакцій (transaction) processing monitor - TP monitor).
- Термін СУБД також часто використовується щодо конкретних програмних продуктів конкретних виробників, наприклад, таких як DB2 Universal Database компанії IBM. Іноді у випадках, коли конкретна копія подібного продукту встановлюється для роботи на певному комп'ютері, використовується термін екземпляр. Безумовно, необхідно суворо розрізняти ці поняття. [1]

Користувачі. Користувачів можна розділити на три великі групи, що частково перекриваються.

- Перша група — прикладні програмісти, які відповідають за написання прикладних програм, що використовують базу даних. Для цього використовують такі мови, як COBOL, PL/I, C++, Java тощо. Прикладні програми отримують доступ до бази даних за допомогою видачі відповідного запиту до СУБД (зазвичай це якийсь оператор SQL). Подібні програми можуть бути простими пакетними додатками або інтерактивними додатками, призначеними для підтримки роботи кінцевих користувачів. Вони надають користувачам безпосередній оперативний доступ до бази даних через робочу станцію, термінал чи персональний комп'ютер. Більшість сучасних програм відноситься саме до цієї категорії. [1]
- Друга група - кінцеві користувачі, які працюють із системою баз даних в інтерактивному режимі, як зазначено у попередньому пункті. Кінцевий користувач може отримувати доступ до бази даних, застосовуючи один з інтерактивних додатків, згаданих вище, або інтерфейс, інтегрований у програмне забезпечення самої СУБД. Безумовно, подібний інтерфейс також підтримується інтерактивними програмами, але ці програми не створюються користувачами-програмістами, а є вбудованими у СУБД. Більшість СУБД

включає принаймні один такий вбудований додаток, а саме - процесор мови запитів, що дозволяє користувачеві в діалоговому режимі вводити запити до бази даних, наприклад, з операторами SELECT чи INSERT. Мова SQL є типовим прикладом мови запитів до бази даних. Крім мови запитів, у більшості систем додатково надаються спеціалізовані вбудовані інтерфейси, в яких користувач явно не використовує пропозиції, або команди з такими операторами, як SELECT та INSERT. Робота з базою даних здійснюється за рахунок вибору користувачем необхідних елементів меню або заповнення необхідних полів у формах. Такі некомандні інтерфейси засновані на меню та формах, полегшують роботу з базами даних для тих, хто не має досвіду роботи з інформаційними технологіями (або IT; часто використовується також скорочення ІС - інформаційні системи; ці поняття практично еквівалентні). Командний інтерфейс, тобто. мову запитів, навпаки, вимагає деякого професійного досвіду роботи з IT (але, безумовно, не такого великого, який необхідний для написання прикладних програм мовою програмування, подібною до COBOL). Однак командний інтерфейс гнучкіший, ніж некомандний, до того ж мови запитів зазвичай включають певні функції, відсутні в інтерфейсах, заснованих на використання меню чи форм. [1]

- Третя група - адміністратори бази даних, або АБД. Більш детально про цю групу користувачів буде описано нижче. [1]

Розглянемо докладніше концепцію централізованого управління. Передбачається, що в централізованому управлінні для підприємства, що використовує базу даних, є людина, яка відповідає за дані підприємства. Це адміністратор даних, або скорочено АД. У зв'язку з тим, що дані - це одна з головних цінностей підприємства, адміністратор повинен розумітися на них і розуміти потреби підприємства стосовно даних на рівні вищої ланки управління у керівництві підприємством. Сам адміністратор даних також повинен ставитись до цієї ланки. До його обов'язків входить прийняття рішень про те, які дані необхідно вносити у базу даних насамперед, а також вироблення вимог щодо супроводу та обробці даних після їх внесення до бази

даних. Прикладом таких вимог може бути розпорядження про те, хто і за яких обставин має право виконувати конкретні операції над тими чи іншими даними. Іншими словами, адміністратор даних повинен забезпечувати захист даних. [1]

Реляційна модель даних - це підхід до керування даними за допомогою структури та мови, які відповідають логіці предикатів першого порядку, що вперше був запропонований британським вченим співробітником компанії IBM Едгаром Франком Коддом (E. F. Codd) в 1970 році в статті «A Relational Model of Data for Large Shared Data Banks». При цьому підході - всі дані представлені в члени кортежів, згруповані у відносини. База даних, організована в термінах реляційної моделі, є реляційною базою даних. [2]

Мета реляційної моделі полягає в тому, щоб забезпечити декларативний метод для визначення даних і запитів: користувачі безпосередньо вказують, яку інформацію містить база даних і яку інформацію вони хочуть отримати від неї, а програмне забезпечення системи управління базою даних піклується про опис структур даних для зберігання дані та процедури пошуку для відповідей на запити. [2]

Більшість реляційних баз даних використовують мову визначення даних і запитів SQL. Ці системи реалізують те, що можна розглядати як інженерне наближення до реляційної моделі. Таблиця в схемі бази даних SQL відповідає предикатній змінній; вміст таблиці до відношення; ключові обмеження, інші обмеження та запити SQL відповідають предикатам. Однак бази даних SQL відрізняються від реляційної моделі в багатьох деталях, і Кодд запекло виступав проти відхилень, які компрометують вихідні принципи. [3]

Тип даних, який використовується в типовій реляційній базі даних, може бути набором цілих чисел, набором рядків символів, набором дат або двома логічними значеннями true і false тощо. Відповідними назвами типів для цих типів можуть бути рядки «int», «char», «date», «boolean» тощо. Однак важливо розуміти, що реляційна теорія не визначає, які типи мають підтримуватися; справді, в даний час очікується, що положення будуть доступні для типів, визначених користувачем, на додаток до вбудованих, наданих системою. [3]

Атрибут — це термін, який використовується в теорії для того, що зазвичай називають стовпцем. Так само таблиця зазвичай використовується замість теоретичного терміна відношення (хоча в SQL цей термін аж ніяк не є синонімом відношення). Структура даних таблиці визначається як список визначень стовпців, кожне з яких визначає унікальне ім’я стовпця та тип значень, дозволених для цього стовпця. Значення атрибута —це запис у певному стовпці та рядку. [3]

Кортеж — це в основному те саме, що й рядок, за винятком СУБД SQL, де значення стовпців у рядку впорядковані. [3]

Відношення — це визначення структури таблиці (визначення набору стовпців) разом із даними, які з’являються в цій структурі. Визначення структури — це заголовок, а дані, що містяться в ньому, — це тіло, набір рядків. Відносна змінна бази даних (змінна відношення) широко відома як базова таблиця. Заголовок його присвоєного значення в будь-який час відповідає декларації таблиці, а його тіло — те, що останнє присвоєно йому за допомогою виклику деякого оператора оновлення (як правило, INSERT, UPDATE або DELETE). Заголовок і тіло таблиці, отриманої в результаті оцінки деякого запиту, визначаються визначеннями операторів, які використовуються у виразі цього запиту. [3]

SQL, який спочатку став стандартною мовою для реляційних баз даних, відхиляється від реляційної моделі в кількох місцях. Поточний стандарт ISO SQL не згадує реляційну модель і не використовує реляційні терміни чи концепції. Однак можна створити базу даних, що відповідає реляційній моделі, використовуючи SQL, якщо не використовувати певні функції SQL. [3]

Відмічено наступні відхилення від реляційної моделі у SQL. Наразі небагато серверів баз даних повністю реалізують стандарт SQL і, зокрема, не допускають деяких із цих відхилень. У той час як NULL є всюдисущим, наприклад, дозволити дублювання імен стовпців у таблиці або анонімних стовпців є рідкістю. [3]

- Повторювані рядки. Один і той самий рядок може з’являтися в таблиці SQL кілька разів. Один і той самий кортеж не може з'являтися більше одного разу у відношенні. [3]
- Анонімні колонки. Стовпець у таблиці SQL може бути безіменним, тому на нього не можна посилатися у виразах. Реляційна модель вимагає, щоб кожен атрибут мав назву і на нього можна було посилатися. [3]
- Повторювані назви стовпців. Два або більше стовпців однієї таблиці SQL можуть мати однакові назви, тому на них не можна посилатися через очевидну неоднозначність. Реляційна модель вимагає, щоб на кожен атрибут можна було посилатися. [3]
- Значення порядку стовпців. Порядок стовпців у таблиці SQL визначений і важливий, одним із наслідків є те, що реалізації SQL декартового добутку та об’єднання є некомутативними. Реляційна модель вимагає, щоб будь-яке впорядкування атрибутів відношення не мало значення. [3]
- Таблиці без стовпців не розпізнані. SQL вимагає, щоб кожна таблиця мала принаймні один стовпець, але є два відношення нульового ступеня (мощності один і нуль), і вони необхідні для представлення розширень предикатів, які не містять вільних змінних. [3]
- NULL. Цей спеціальний знак може з’явитися замість значення скрізь, де значення може з’явитися в SQL, зокрема замість значення стовпця в якомусь рядку. Відхилення від реляційної моделі виникає через той факт, що реалізація цієї спеціальної концепції в SQL передбачає використання тризначної логіки, згідно з якою порівняння NULL із самим собою не дає значення true, а натомість дає третє значення істинності, невідоме; так само порівняння NULL з чимось іншим, ніж він сам, не дає false, а натомість дає невідоме. Саме через таку поведінку в порівняннях NULL описується як позначка, а не як значення. Реляційна модель залежить від закону виключеної середини, згідно з яким все, що не є істинним, є хибним, а все, що не є хибним, є істинним; він також вимагає, щоб кожен кортеж у тілі відношення мав значення для кожного атрибута цього відношення. Дехто заперечує це

конкретне відхилення хоча б тому, що сам Е. Ф. Кодд зрештою виступав за використання спеціальних позначок і 4-значної логіки, але це ґрунтувалося на його спостереженні про те, що існує дві різні причини, чому можна захотіти використовувати спеціальна позначка замість значення, яка спонукала противників використання такої логіки до виявлення більш чітких причин, і було зазначено щонайменше 19, які вимагали б 21-значної логіки. Сам SQL використовує NULL для кількох цілей, крім представлення "невідомого значення". Наприклад, сума порожнього набору дорівнює NULL, що означає нуль, середнє значення порожнього набору дорівнює NULL, що означає невизначений, а NULL, що з’являється в результаті LEFT JOIN, може означати “немає значення, оскільки немає відповідного рядка в правого операнда”. Існують способи проектування таблиць, щоб уникнути потреби в NULL, як правило, те, що можна вважати або нагадувати високий ступінь нормалізації бази даних, але багато хто вважає це непрактичним. [3]

Кожен рядок у таблиці має свій унікальний ключ. Рядки в таблиці можна зв’язати з рядками в інших таблицях, додавши стовпець для унікального ключа зв’язаного рядка (такі стовпці відомі як зовнішні ключі). [3]

Частина цієї обробки передбачає постійну можливість вибору або зміни одного й лише одного рядка в таблиці. Тому більшість фізичних реалізацій мають унікальний первинний ключ (primary key - PK) для кожного рядка в таблиці. Коли новий рядок записується в таблицю, генерується нове унікальне значення для первинного ключа; це ключ, який система використовує в основному для доступу до таблиці. Продуктивність системи оптимізовано для ПК. Інші, більш природні ключі також можуть бути ідентифіковані та визначені як альтернативні ключі (alternate keys - AK). Часто для формування AK потрібні кілька стовпців (це одна з причин, чому один цілий стовпець зазвичай робиться PK). І PK, і AK мають можливість однозначно ідентифікувати рядок у таблиці. Додаткова технологія може бути застосована для забезпечення унікального ідентифікатора в усьому світі, глобального унікального ідентифікатора, якщо існують ширші системні вимоги. [3]

Первинні ключі в базі даних використовуються для визначення зв’язків між таблицями. Коли PK переміщується в іншу таблицю, він стає зовнішнім ключем в іншій таблиці. Коли кожна комірка може містити лише одне значення, а PK переходить у звичайну таблицю сутностей, цей шаблон проектування може представляти зв’язок «один до одного» або «один до багатьох». Більшість проектів реляційних баз даних вирішують багато-до-багатьох зв’язки шляхом створення додаткової таблиці, яка містить PK з обох інших таблиць сутностей – зв’язок стає сутністю; потім таблиці роздільної здатності присвоюється відповідне ім’я, а два FK об’єднуються, щоб сформувати PK. Міграція PK в інші таблиці є другою основною причиною, чому призначені системою цілі числа зазвичай використовуються як PK; зазвичай немає ані ефективності, ані ясності в міграції купи інших типів стовпців. [3]

1.2. Аналіз популярності сучасних архітектур розподілених баз даних.

За даними DB-Engines, у жовтні 2022 року найпопулярнішими системами на сайті db-engines.com були: [4]

1. База даних Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL (безкоштовне програмне забезпечення)
5. IBM Db2
6. Microsoft Access
7. SQLite (безкоштовне програмне забезпечення)
8. MariaDB (безкоштовне програмне забезпечення)
9. Snowflake
10. Microsoft Azure SQL Database

11. Apache Hive (безкоштовне програмне забезпечення)
12. Teradata

1.3. Дослідження організації розподілених баз даних з динамічною структурою.

1.3.1. Загальний опис розподілених баз даних.

Розподілена база даних — це сукупність кількох взаємопов'язаних баз даних, фізично розподілених у різних місцях, які взаємодіють через комп'ютерну мережу. [4, 5]

Система управління розподіленою базою даних (DDBMS) — це централізована система програмного забезпечення, яка керує розподіленою базою даних таким чином, ніби всі вони зберігаються в одному місці. [4]

У системі управління розподіленими базами даних (розподілена СУБД або DDBMS) дані, процеси та компоненти інтерфейсу системи розподілені між кількома місцями, щоб забезпечити незалежне функціонування. Різні бази даних зберігаються на кількох комп'ютерах, і, відповідно, обробка робочого навантаження розподіляється. Ця мережева архітектура забезпечує набагато кращу продуктивність і підвищену надійність порівняно з централізованими системами керування базами даних. [4]

1.3.2. Переваги розподілених баз даних

Нижче наведено переваги розподілених баз даних перед централізованими базами даних [4]:

- Модульна розробка – якщо систему потрібно розширити до нових місць розташування або нових підрозділів у централізованих системах баз даних, дія потребує значних зусиль і порушення існуючого функціонування. Однак у розподілених базах даних робота просто вимагає додавання нових комп’ютерів і локальних даних до нового сайту та, нарешті, підключення їх до розподіленої системи без переривання поточних функцій.
- Більш надійні − у разі збою бази даних загальна система централізованих баз даних припиняє роботу. Однак у розподілених системах, коли компонент виходить з ладу, функціонування системи може продовжуватися зі зниженою продуктивністю. Тому DDBMS більш надійна.
- Краща відповідь – якщо дані розподіляються ефективним чином, тоді запити користувачів можуть задовольнятися з самих локальних даних, що забезпечує швидшу відповідь. З іншого боку, у централізованих системах усі запити мають проходити через центральний комп’ютер для обробки, що збільшує час відповіді.
- Нижча вартість зв’язку – у системах розподілених баз даних, якщо дані розташовані локально, де вони здебільшого використовуються, тоді витрати на зв’язок для маніпулювання даними можна мінімізувати. Це неможливо в централізованих системах.

1.3.3. Недоліки розподілених баз даних

Нижче наведено деякі недоліки, пов'язані з розподіленими базами даних [4]:

- Потреба у складному та дорогому програмному забезпеченні − DDBMS вимагає складного та часто дорогого програмного забезпечення для забезпечення прозорості даних та координації між кількома сайтами.

- Накладні витрати на обробку. Навіть прості операції можуть вимагати великої кількості зв’язків і додаткових обчислень, щоб забезпечити однаковість даних на сайтах.
- Цілісність даних – необхідність оновлення даних на кількох сайтах створює проблеми з цілісністю даних.
- Накладні витрати на неправильний розподіл даних − швидкість реагування на запити значною мірою залежить від правильного розподілу даних. Неправильний розподіл даних часто призводить до дуже повільної реакції на запити користувачів.

Однією з головних переваг використання розподілених систем керування базами даних є те, що програмам не потрібно знати, де зберігаються дані (також називається прозорим доступом до даних). Коли користувач виконує запит до розподіленої бази даних, кілька сайтів із різних центрів обробки даних можуть безперешкодно співпрацювати, щоб відповісти на запит користувача. [4]

На відміну від централізованої системи керування базами даних, розподілена база даних постачається з узгодженими протоколами фіксації, методами керування паралелізмом і методами відновлення, які є надзвичайно корисними для керування великими транзакціями користувачів або запобігання збоям системи баз даних. [4]

1.3.4. Особливості системи управління розподіленими базами даних.

Система керування розподіленою базою даних (DDBMS) забезпечує повну функціональність стандартної СУБД і забезпечує прозоре керування розподіленими та реплікованими даними. Це також [5]:

- Покращує надійність і доступність даних за допомогою розподілених транзакцій.

- Забезпечує більш просту та економічну систему для розширення бази даних.
- Зберігає конфіденційність і цілісність даних баз даних.
- Не залежить від апаратного забезпечення.
- Забезпечує швидкі відповіді на запити користувачів завдяки ефективному розподілу даних.
- Надає користувачам простий інтерфейс для відкриття, читання/запису записів і закриття файлів.
- Поставляється з можливостями декларативних запитів, керування транзакціями та забезпечення цілісності.

Окрім цього, існують ще певні особливості розподіленої бази даних [5]:

- Бази даних у колекції логічно взаємопов'язані між собою. Часто вони являють собою єдину логічну базу даних.
- Дані фізично зберігаються на кількох сайтах. Даними на кожному сайті може керувати СУБД, незалежна від інших сайтів.
- Процесори в сайтах підключені через мережу. Вони не мають багатопроцесорної конфігурації.
- Розподілена база даних не є слабозв'язаною файловою системою.
- Розподілена база даних включає обробку транзакцій, але вона не є синонімом системи обробки транзакцій.

1.3.5. Розподіл даних

Розподіл даних — це інтелектуальний розподіл фрагментів даних (так звані фрагменти даних) для підвищення продуктивності бази даних і доступності даних для кінцевих користувачів. Він спрямований на зниження загальних витрат на обробку транзакцій, а також швидке надання точних даних у ваших системах DDBMS. [4, 5]

Розподіл даних є одним із ключових кроків у створенні ваших систем розподілених баз даних. Для оптимального розподілу даних використовуються дві поширені стратегії: фрагментація даних і реплікація даних. У наступних розділах, присвячених фрагментації та реплікації в розподілених базах даних, ми обговорюємо обидва ці методи більш детально. [4, 5]

1.3.6. Фрагментація даних

Фрагментація — це процес поділу зв’язків або таблиць на кілька секцій на кількох сайтах. Він розділяє базу даних на різні підтаблиці та підзв’язки, щоб дані могли розподілятися та зберігатися ефективно. [4, 5]

Фрагментація бази даних може бути двох типів: горизонтальна і вертикальна. У горизонтальній фрагментації кожен кортеж відношення r призначається одному або декільком фрагментам. При вертикальній фрагментації схема для відношення r розбивається на численні менші схеми із загальним ключем-кандидатом і спеціальним атрибутом. [4, 5]

1.3.7. Реплікація даних

Реплікація розподіленої бази даних — це процес створення та підтримки кількох копій (надмірності) даних на різних сайтах. Головна перевага таблиці полягає в тому, що дублювання даних забезпечує швидший пошук. Це усуває окремі точки збою та проблеми з втратою даних, якщо один сайт не виконує запити користувачів, і, отже, надає відмовостійку систему. [4, 5]

Однак реплікація розподіленої бази даних також має деякі недоліки. Щоб забезпечити точні та правильні відповіді на запити користувачів, дані мають

постійно оновлюватися та синхронізуватися. Якщо цього не зробити, це призведе до неузгодженості даних, що може перешкодити досягненню бізнес-цілей. [5]

1.3.8. Типи розподілених баз даних

Розподілені бази даних можна в цілому класифікувати на однорідні та гетерогенні середовища розподілених баз даних, кожна з яких має додаткові підрозділи, як показано на рисунку 1.1.

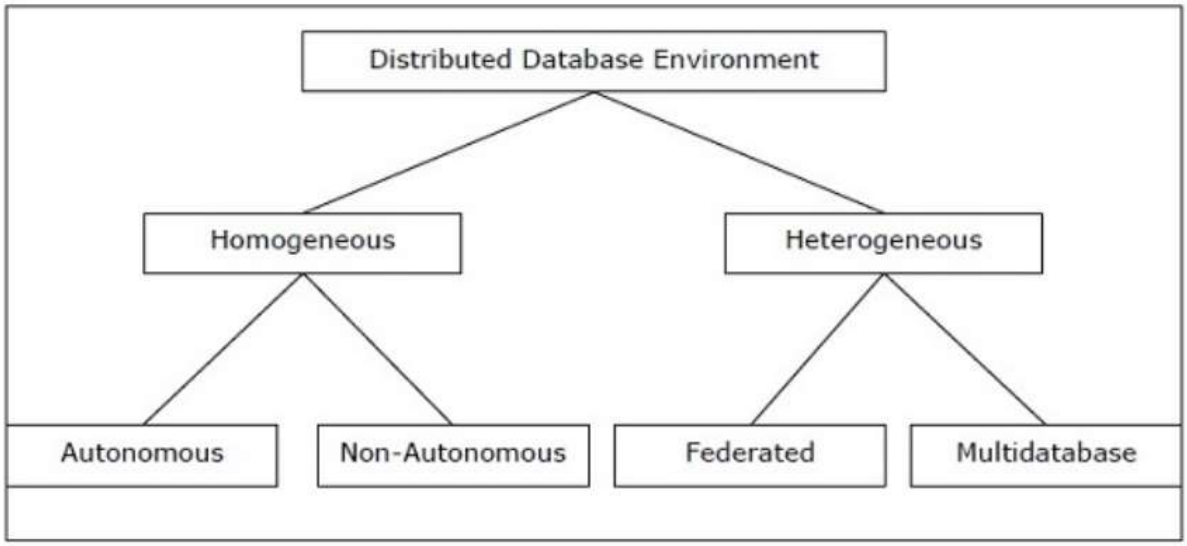

Рисунок 1.1 - Структура розподілених баз даних [5]

1.3.8.1. Однорідні розподілені бази даних

В однорідній розподіленій базі даних усі сайти використовують ідентичні СУБД та операційні системи.

Властивості однорідної розподіленої бази даних [5]:

- Сайти використовують дуже схоже програмне забезпечення.

- На сайтах використовується ідентична СУБД або СУБД одного постачальника.
- Кожен сайт знає про всі інші сайти та співпрацює з іншими сайтами для обробки запитів користувачів.
- Доступ до бази даних здійснюється через єдиний інтерфейс, наче це одна база даних.

Є 2 типи однорідної розподіленої бази даних [5]:

- Автономна – кожна база даних є незалежною та функціонує сама по собі. Вони інтегровані керуючою програмою та використовують передачу повідомлень для обміну оновленнями даних.
- Неавтономна – дані розподіляються між однорідними вузлами, а центральна або головна СУБД координує оновлення даних на сайтах.

1.3.8.2. Гетерогенні розподілені бази даних

У гетерогенній розподіленій базі даних різні сайти мають різні операційні системи, продукти СУБД і моделі даних.

Властивості гетерогенної розподіленої бази даних [5]:

- На різних сайтах використовуються різні схеми та програмне забезпечення.
- Система може складатися з різноманітних СУБД, таких як реляційна, мережева, ієрархічна або об’єктно-орієнтована.
- Обробка запитів є складною через різні схеми.
- Обробка транзакцій є складною через різне програмне забезпечення.
- Сайт може не знати про інші сайти, тому співпраця в обробці запитів користувачів обмежена.

Є 2 типи гетерогенних розподілених баз даних [5]:

- Об’єднані – гетерогенні системи баз даних є незалежними за своєю природою та інтегровані разом, щоб вони функціонували як єдина система баз даних.
- Необ’єднані – системи баз даних використовують центральний координаційний модуль, через який здійснюється доступ до баз даних.

1.3.9. Контроль бази даних

Контроль бази даних відноситься до завдання забезпечення дотримання правил для надання правильних даних справжнім користувачам і програмам бази даних. Для того, щоб правильні дані були доступні для користувачів, усі дані повинні відповідати обмеженням цілісності, визначеним у базі даних. Крім того, дані слід відсіювати від неавторизованих користувачів, щоб підтримувати безпеку та конфіденційність бази даних. Контроль бази даних є одним із основних завдань адміністратора бази даних (DBA). [5]

Існують три виміри контролю бази даних:

1. Аутентифікація
2. Права доступу
3. Обмеження цілісності

Розглянемо їх більш детально.

У системі розподіленої бази даних аутентифікація — це процес, за допомогою якого лише законні користувачі можуть отримати доступ до ресурсів даних. [5]

Автентифікацію можна застосувати на двох рівнях [5]:

1. Контроль доступу до клієнтського комп’ютера – на цьому рівні доступ користувача обмежено під час входу до клієнтського комп’ютера, який забезпечує інтерфейс користувача до сервера бази даних. Найпоширенішим

методом є комбінація імені користувача та пароля. Однак більш складні методи, такі як біометрична автентифікація, можуть використовуватися для даних високого рівня безпеки.

2. Контроль доступу до програмного забезпечення бази даних – на цьому рівні програмне забезпечення/адміністратор бази даних призначає користувачеві деякі облікові дані. За допомогою цих облікових даних користувач отримує доступ до бази даних. Одним із методів є створення облікового запису для входу на сервер бази даних.

Права доступу користувача стосуються привілеїв, наданих користувачеві щодо операцій СУБД, таких як права на створення таблиці, видалення таблиці, додавання/видалення/оновлення кортежів у таблиці або запит до таблиці. [5]

У розподілених середовищах, оскільки існує велика кількість таблиць і водночас велика кількість користувачів, неможливо призначити індивідуальні права доступу користувачам. Отже, DDBMS визначає певні ролі. Роль — це конструкція з певними привілеями в системі бази даних. Після визначення різних ролей окремим користувачам призначається одна з цих ролей. Часто ієрархія ролей визначається відповідно до ієрархії повноважень і відповідальності в організації. [5]

Контроль семантичної цілісності визначає та забезпечує дотримання обмежень цілісності системи бази даних. [5]

Обмеження цілісності наступні [5]:

- Обмеження цілісності типу даних
- Обмеження цілісності сутності
- Обмеження посилальної цілісності

Обмеження типу даних обмежує діапазон значень і тип операцій, які можна застосувати до поля з вказаним типом даних. [5]

Контроль цілісності сутності забезпечує виконання правил, щоб кожен кортеж можна було унікально ідентифікувати з інших кортежів. Для цього визначається первинний ключ. Первинний ключ — це набір мінімальних полів, які можуть однозначно ідентифікувати кортеж. Обмеження цілісності сутності

стверджує, що два кортежі в таблиці не можуть мати однакові значення для первинних ключів і що жодне поле, яке є частиною первинного ключа, не може мати значення NULL. [5]

Обмеження посилальної цілісності встановлює правила зовнішніх ключів. Зовнішній ключ — це поле в таблиці даних, яке є первинним ключем пов’язаної таблиці. Обмеження посилальної цілісності встановлює правило, згідно з яким значення поля зовнішнього ключа має або бути серед значень первинного ключа таблиці, на яку посилається, або бути повністю NULL. [5]

1.3.10. Архітектура розподіленої бази даних з динамічною структурою

У статті [7] запропонована архітектура дозволяє користувачам отримувати доступ до бази даних з будь-якої точки світу, не володіючи жодною технологічною інфраструктурою. До неї можна отримати доступ через: веб-браузер, мобільну програму або програму для робочого столу, поки база даних зберігається на серверах на віддалених сайтах. Це також дозволяє динамічно приймати рішення FAR під час виконання. Ці рішення базуються на шаблонах доступу та завантаженні сайтів після розподілу та міграції фрагментів, а також їх реплік. Пропонована архітектура показана на рисунку 1.2. Вона складається з двох рівнів: менеджера системи розподіленої бази даних і кластерів розподіленої бази даних.

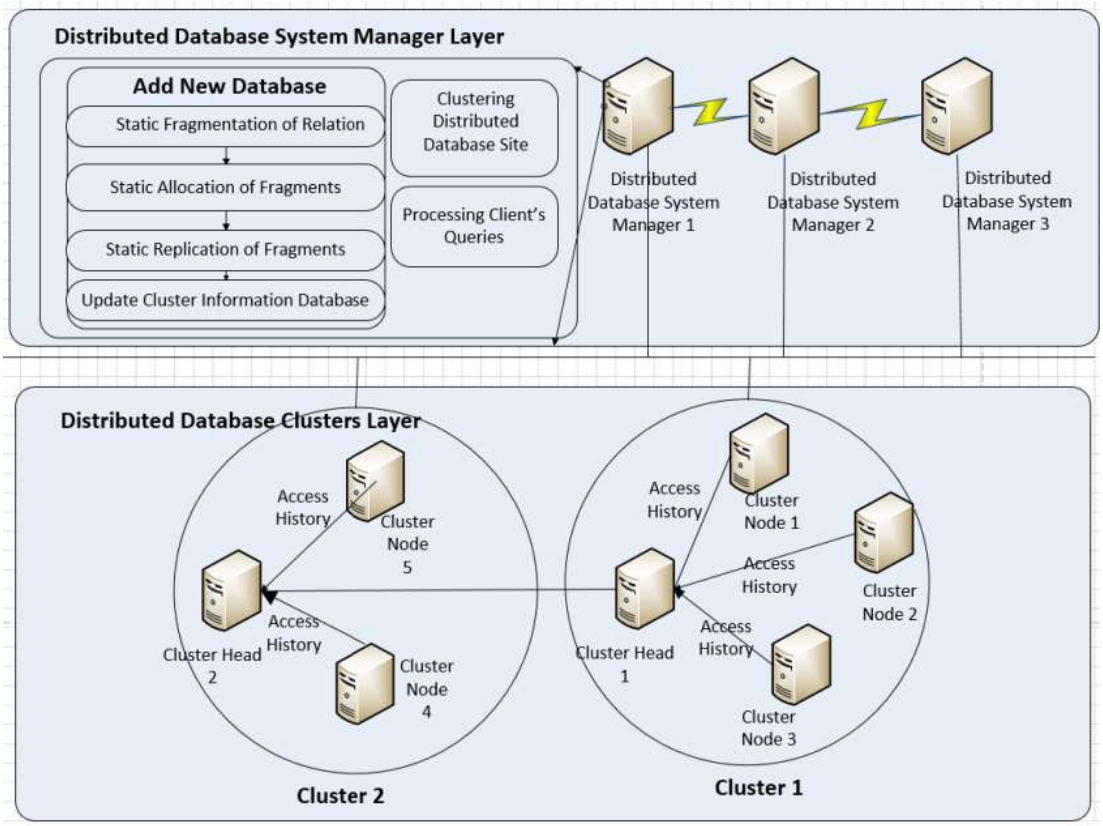

Рисунок 1.2 - Динамічна розподілена система баз даних у архітектурі хмарного середовища. [7]

1. **Рівень менеджера системи розподіленої бази даних.** Цей рівень складається з трьох модулів: додавання нової бази даних (add new database), кластеризація сайтів розподілених баз даних (clustering distributed database sites) і обробка клієнтських запитів (client queries processing). [7]

    - **Додавання нової бази даних.** Цей модуль використовується для додавання таблиць бази даних до DDBS. По-перше, нові таблиці фрагментовані за допомогою техніки статичної фрагментації, яку можна використовувати на початковому етапі розробки DDBS. По-друге, фрагменти розподіляються по сайтам кожного кластера. По-третє, деякі фрагменти тиражуються на різні сайти кожного кластера з метою підвищення продуктивності системи та підвищення доступності. Нарешті, інформаційна база даних кластера оновлюється, щоб зберегти розташування кожного фрагмента та репліки в системі розподіленої бази даних.

Інформаційна база даних кластера — це база даних, яка містить повну інформацію про кожен фрагмент або репліку на якому сайті чи кластері. [7]

- **Модуль кластеризації розподілених сайтів баз даних**. Системний менеджер розподілених баз даних використовуватиме техніку кластеризації для кластеризації сайтів розподілених баз даних у непересічні кластери. Він також розподіляє сайти для кожного кластера. Нарешті, він оновлює інформаційну базу даних кластера з останнім розташуванням кожного сайту в DDBS. [7]
- **Модуль обробки запитів клієнтів.** Коли клієнт спочатку надсилає запит диспетчеру системи розподіленої бази даних, менеджер системи розподіленої бази даних використовує базу даних інформації кластера, щоб визначити найближчий сайт, який містить необхідні дані. Потім він перенаправляє користувача на цей сайт, щоб отримати необхідні дані. [7]

2. **Рівень кластерів розподіленої бази даних.** Рівень кластерів розподіленої бази даних складається з більш ніж одного кластера. Кожен кластер має одну головку кластера та більше одного вузла кластера. Голова кластера — це вузол кластера, який має спеціальні та додаткові операції для керування іншими вузлами кластера. При кожному доступі до вузла кластера, по-перше, вузол кластера перевіряє, локальний чи віддалений доступ. По-друге, дозволяє користувачеві отримати доступ до локальної бази даних вузла для виконання запиту та отримання необхідних даних. По-третє, оновлюється запис доступу до сайту, щоб зберегти інформацію про доступ користувача. Кожен вузол кластера надсилає історію доступу голові кластера для прийняття будь-якого відповідного рішення, наприклад: створення або видалення репліки, рефрагментація або повторний розподіл фрагментів. Керівник кластера приймає рішення на основі історії доступу та завантаження кожного сайту після розподілу або міграції його фрагментів і реплік. Фрагменти та

репліки будуть надсилатися на сайт з меншим навантаженням. Голова кластера також збирає дані, які зберігає кожен вузол у своєму кластері. Після цього він надсилає його системному менеджеру розподіленої бази даних, щоб оновлювати базу даних кластера. [7]

1.4. Порівняльний аналіз організації розподілених баз даних з динамічною структурою.

Різниця між розподіленою базою даних з динамічною структурою та реляційною базою даних полягає в методі організації даних. А саме у способі зберігання та організації доступу до даних. У розподіленій базі даних інформація розділяється, фрагментується та реплікується для підвищення ефективності її обробки та швидкості доступу до даних в цілому.

Окрім цього, полегшується можливість масштабування бази даних за рахунок модульності. Однак, за рахунок складної структури виникають додаткові витрати на обробку даних та на обладнання.

Висновки до розділу 1

У розділі 1 було розглянуто організацію, обробку та зберігання даних у реляційних базах даних та у розподілених базах даних з динамічною структурою. Був проведений порівняльний аналіз їх організації.

РОЗДІЛ 2. МЕТОДИ ТА МОДЕЛІ ОРГАНІЗАЦІЇ ТА УПРАВЛІННЯ РОЗПОДІЛЕНИМИ БАЗАМИ ДАНИХ

2.1. Методи та моделі організації та управління розподіленими базами даних.

2.1.1. Основні принципи створення розподіленої бази даних

Твердження, що було сформульовано у [1, стор. 825], може розглядатися як фундаментальний принцип при створенні розподілених баз даних. А звучить воно так: “Для користувача розподілена система має виглядати так само, як нерозподілена система.”. Таким чином можна сказати, що кінцевий користувач базами даних не повинен відчувати різниці чи ця система є розподіленою, чи ні.

Як наслідок даного твердження, існують 12 додатковий цілей, яких варто дотримуватися при створенні розподілених баз даних [1]:

1. Локальна незалежність.
2. Відсутність залежність від центрального вузла.
3. Безперервне функціонування.
4. Незалежність від розташування.
5. Незалежність від фрагментації.
6. Незалежність від реплікації.
7. Обробка розподілених запитів.
8. Управління розподіленими транзакціями.
9. Апаратна незалежність.
10. Незалежність від операційної системи.
11. Незалежність від мережі.
12. Незалежність від типу СУБД.

Варто зазначити, що не всі ці цілі незалежні одна від одної. Крім того, вони не вичерпні та не всі однаково важливі (різні користувачі можуть надавати різне значення різним цілям у різних середовищах, і, крім того, деякі з цих цілей можуть бути взагалі незастосовні у деяких ситуаціях). Але ці цілі корисні

як основа розуміння самої розподіленої технології як і загальна схема опису функціональних можливостей конкретних розподілених систем. [1]

2.1.2. Модель R* (R - зірочка, R-star)

Перед тим, як описати структуру каталогу системи R*, необхідно сказати кілька слів про механізм іменування об'єктів у цій системі. Привласнення імен об'єктам це взагалі важливе питання для розподілених систем. Оскільки два окремі вузла X і У можуть мати об'єкт з одним і тим ім'ям, припустимо, R, потрібен деякий механізм (зазвичай - уточнення за допомогою імені вузла) для того, щоб усунути неоднозначність, тобто функцію. гарантувати унікальність імені в межах усієї системи. Але якщо уточнені імена, такі як X.R і Y.R, що надаватиметься системою користувачеві, буде порушено вимогу незалежності від розміщення. Тому необхідні засоби відображення імен для користувача відповідних системних імен. [1, 8, 9]

Тепер викладемо підхід до проблеми, прийнятий у системі R*. В цій системі розрізняються імена, за допомогою яких користувачі зазвичай звертаються до об'єктів (наприклад, в конструкції SELECT мови SQL), і загальносистемні імена, які є глобально унікальними внутрішніми ідентифікаторами для цих об'єктів. Загальносистемні імена мають чотири описані нижче компоненти. [1, 8, 9]

- Ідентифікатор автора, тобто ідентифікатор користувача, який вперше викликав виконання операцію CREATE для створення аналізованого об'єкта.
- Ідентифікатор вузла автора, тобто ідентифікатор вузла, на якому було введено відповідна операція CREATE.
- Локальне ім'я, тобто неуточнене ім'я об'єкта.

- Ідентифікатор вузла походження, тобто ідентифікатор вузла, на якому об'єкт зберігався спочатку.

Ідентифікатори користувача є унікальними на вузлі, тоді як ідентифікатори вузла унікальні глобально. Розглянемо, наприклад, таке загальносистемне ім'я. [1]

MAX @ LVIV. STATS @ KYIV

Воно позначає об'єкт (для визначеності вважатимемо, що це базова змінна відносини) з локальним ім'ям STATS, створений користувачем MAX на вузлі в LVIV і спочатку збережений на вузлі в KYIV. Це ім'я гарантовано захищено від будь-яких змін, навіть якщо об'єкт надалі буде переміщений на інший вузол. [1]

Як уже вказувалося, користувачі зазвичай посилаються на об'єкти за допомогою імен, що вводяться. Ім'я, як показано нижче, являє собою просте, неуточнене ім'я - або компонент "локальне ім'я" загальносистемного імені (у наведеному вище прикладі — STATS), чи синонім при цьому загальносистемного імені, що у системі R* визначається за допомогою спеціального оператора CREATE SYNONYM. [1]

CREATE SYNONYM MSTATS FOR MAX@ LVIV. STATS @ KYIV;

Після виконання цього оператора користувач зможе з однаковим успіхом ввести будь-який із двох наведених нижче операторів. [1]

SELECT … FROM STATS … ;

SELECT … FROM MSTATS … ;

У першому випадку, тобто при використанні локального імені система буде виходити з того, що всі компоненти загальносистемного імені визначаються за умовчанням, а саме — що об'єкт створено даним користувачем, створено на даному вузлі та збережено на даному вузлі. До речі, завдяки такому припущенню за промовчанням старі програми системи System R можуть виконуватися без будь-яких виправлень у новій версії системи R* (тобто після перевизначення даних системи System R для системи R*). [1]

У другому випадку, тобто при використанні синоніму, система визначення загальносистемного імені звертається до відповідної таблиці синонімів. Таблиці синонімів можна розглядати як перший компонент каталогу. Кожен вузол підтримує деякий набір таких таблиць, створених для кожного користувача даного вузла, що дозволяє системі відображати синоніми, що застосовуються користувачем, у відповідні загальносистемні імена. [1]

Крім таблиць синонімів, кожному вузлі підтримуються описані нижче дані. [1]

1. Запис каталогу для кожного об'єкта, місцем створення якого є зазначений вузол.

2. Запис каталогу для кожного об'єкта, що зберігається в даний час на даному вузлі.

Припустимо, що користувач видав запит, що містить посилання на синонім MSTATS. Спочатку система здійснить пошук відповідного загальносистемного імені у відповідній таблиці синонімів (виключно локальний перегляд). Після того як стане відомий вузол створення (у нашому прикладі це вузол у KYIV), можна буде опитати каталог вузла в KYIV (тут ми для спільноти маємо на увазі віддалений перегляд - перше віддалене звернення). Каталог вузла з KYIV міститиме запис для об'єкта відповідно до п. 1. Якщо об'єкт, як і раніше, зберігається на вузлі KYIV, він буде знайдений. Однак, якщо об'єкт переміщений, скажімо, на вузол LVIV, запис каталогу на вузлі KYIV міститиме відомості про це і система має опитати каталог на вузлі LVIV (друге віддалене звернення). [1]

Каталог на вузлі у LVIV міститиме запис для шуканого об'єкта згідно п. 2. Таким чином, для пошуку об'єкта буде виконано не більше двох віддалених звернень. [1]

Крім того, якщо об'єкт знову знадобиться перемістити, скажімо, на вузол у DNIPRO, системою будуть виконані наведені нижче дії.

- Вставлення запису в каталог на вузлі DNIPRO.
- Видалення запису з каталогу на вузлі LVIV.

- Оновлення запису каталогу на вузлі в KYIV, який тепер буде вказувати на вузол DNIPRO замість вузла LVIV.

В результаті об'єкт завжди може бути знайдений за допомогою не більше двох віддалених звернень. І це повністю розподілена схема - немає ніякого вузла з основним каталогом і немає єдиного джерела відмови всередині системи.

2.2. Математична модель оцінювання параметрів обробки інформації в розподілених системах з динамічною структурою в яких реалізовано гетерогенні розподілені бази даних

Для формалізації обробки запитів в розподіленій системі з використанням баз даних потрібно розробити математичну модель, яка опише всі процеси, що проходять при обробці пакетів з запитами, наложить обмеження на систему та дозволить використати її в подальшій розробці моделей та методів.

Система обробки запитів (Query Processing System) в розподіленій системі з використанням гетерогенних розподілених баз даних задається функцією від наступної множини характеристик [10, 11]:

$$\boldsymbol{QPS}(\boldsymbol{t},\boldsymbol{n}) = \boldsymbol{F}(\boldsymbol{SN},\boldsymbol{SC},\boldsymbol{SP},\boldsymbol{SPDB},\boldsymbol{SPCN}),$$

де SN – множина вузлів (Set of Nodes) розподіленої системи обробки даних;

SC – множина зв'язків (Set of Connections) між вузлами розподіленої системи обробки даних;

SP – множина параметрів (Set of Parameters) розподіленої системи обробки даних;

SPDB – множина параметрів (Set Parameters Database) бази даних;

SPCN – множина параметрів вузлів керування (Set Parameters of Control Nodes) розподіленою системою обробки даних;

t- одиниця часу;

n – кількість вузлів системи в даний момент часу.

Множину вузлів розподіленої системи обробки даних позначимо як функцію від кількох параметрів та опишемо їх [10, 11]:

$$\boldsymbol{SN(t,n) = fN(scn, sdbn, sein, srn)}$$

де scn — множина вузлів (Set of Control Nodes) в яких встановлено програмне забезпечення для керування розподіленою системою обробки даних;

sdbm — множина вузлів (Set of Database Nodes) в яких встановлено програмне забезпечення для гетерогенних розподілених баз даних;

sein – множина проміжних вузлів (Set of Intermediate Nodes), які виконують функцію маршрутизації пакетів;

srn — множина вузлів (Set of Requests Nodes), з яких надходять запити до розподіленої системи обробки даних.

Відома модель, в якій множина параметрів розподіленої системи обробки даних описана наступною функцією [10, 11]:

$$\boldsymbol{SP(t,n) = fp(P, V, R, DB, Tdb, Tc)}$$

де P— продуктивність вузла обробки даних в розподіленій системі обробки даних;

V – швидкість передачі даних по каналу зв'язку в розподіленій системі обробки даних;

R— надійність вузла розподіленої системи обробки даних;

DB — індикатор присутності в вузлі гетерогенної розподіленої бази даних;

Tdb– час обробки пакету – запиту в вузлі розподіленій системі обробки даних в якому знаходиться система управління базою даних;

Tc — час оновлення програмного забезпечення в вузлі керування розподіленої системи обробки даних.

Введемо обмеження на параметри розподіленої системи обробки даних, а саме на продуктивність вузла [1]:

$$\boldsymbol{P_{min} \leq P}$$

де P_{min} — залежить від типу обладнання на якому знаходиться вузол та його характеристик, але не повинна бути меншою за 0.5 терафлопси.

Опишемо початкові умови обмеження швидкості передачі даних по каналу зв'язку в розподіленій системі обробки даних. [10]

$$\boldsymbol{V_{min} \leq V},$$

де V_{min} — визначатиметься від типу вузлів, які під'єднанні до розподіленої системи, але не менше ніж 10мбіт/с. В зв'язку з тим, швидкість передачі впливає на час доставки пакетів між вузлами, то вона є ключовим параметром, як провідних розподілених системах обробки даних так і в безпровідних системах. [10]

Введемо обмеження на параметри розподіленої системи обробки даних, а саме на надійність вузла [10]:

$$\boldsymbol{R_{min} \leq R},$$

де R_{min} — визначається як безвідмовність вузла системи, яка визначається, як вірогідність безвідмовної роботи системи і залежить від часу t. Відповідно безвідмовність всієї системи обчислюється [10]:

$$\boldsymbol{R(n) = \prod_{i=1}^{n} e^{-\lambda_i t}},$$

де λ_i- інтенсивність виходу з ладу і-го елемента системи. Згідно з середніми статистичними даними, інтенсивність виходу з ладу елементів системи знаходяться в межах інтервалу від 10^{-3} до $5 * 10^{-3}$. Тоді R_{min} залежить від кількості вузлів, але не менше ніж 0,5. [10]

Приймемо, що признак присутності в вузлі гетерогенної розподіленої бази даних буде приймати значення [10]:

$$\boldsymbol{DB} = \begin{cases} \boldsymbol{0 - NoDB}; \\ \boldsymbol{1 - YesDB}. \end{cases}$$

де 0 — приймається у випадку, коли в вузлів відсутня система управління базою даних;

1 – приймається у випадку, коли в вузлі встановлене програмне забезпечення баз даних.[10]

Введемо обмеження на час обробки пакету — запиту в вузлі розподіленої системи обробки даних в якому розміщена система управління базою даних, а саме [10]:

$$\boldsymbol{T_{dbmin} \leq T_{db} \leq T_{dbmax},}$$

де T_{dbmax} — визначається як максимальний час затримки обробки пакетів в вузлі бази даних. А T_{dbmin} — повинно бути як найменшим і прямувати до нуля, але значення нуля ніколи не приймати. [10, 12]

Опишемо початкові умови оновлення програмного забезпечення в вузлі керування розподіленої системи обробки даних, а саме час затримки обробки пакетів [10, 12]:

$$\boldsymbol{0 \leq T_c \leq T_{cmax},}$$

де T_{cmax} — визначається як подвоєна сума максимальних часів затримки пакетів оновлення в вузлах на шляху до вузла керування та максимальний час оновлення програмного забезпечення (планувальник та брокер) в вузлі керування. [10, 12]

2.3. Модифікована модель оцінювання параметрів обробки інформації в розподілених системах з динамічною структурою в яких реалізовано гетерогенні розподілені бази даних з урахуванням навантаженості системи

Модифікація моделі зазначеної у розділі 2.3:

$$\boldsymbol{SP(t,n) = fp(P,V,R,DB,Tdb,Tc,U),}$$

де U - навантаження системи користувачами, тобто яка частина користувачів в один момент працює в розподіленій системі баз даних.

Для оцінки витрат часу на запит безпосередньо у моделі використаємо підхід, що буде описано наступною функцією:

$$\boldsymbol{T(u) = T_c + T_c * (scn - 1) * p_{c\_u} + T_{db} + T_{db} * (sdbn - 1) * p_{db\_u} + L/V,}$$

де T_c — час оновлення програмного забезпечення в вузлі керування розподіленої системи обробки даних;

scn - кількість вузлів керування;

p_{c_u} - ймовірність відмови прийняття запиту вузлом керування через перевантаженість;

Tdb– час обробки пакету – запиту в вузлі розподіленій системі обробки даних в якому знаходиться система управління базою даних;

sdbn - кількість вузлів з ПЗ для РБД;

p_{db_u} - ймовірність відмови прийняття запиту вузлом з ПЗ для РБД через перевантаженість;

V – швидкість передачі даних по каналу зв'язку в розподіленій системі обробки даних;

L - кількість передач запитів між вузлами, вираховується за наступною формулою:

$$\boldsymbol{L = 2 + 2 * (scn - 1) * p_{c\_u} + 2 + 2 * (sdbn - 1) * p_{db\_u}}$$

Можна помітити, що 2 - означає, що запит перейшов від вузла, що створив запит, до керуючого вузла, після чого повернувся назад з відповіддю. Потім запит попрямував до вузла з ПЗ для РБД, звідки взяв необхідну інформацію та повернувся назад до вузла - "творця" запиту. Добуток дозволяє врахувати відмови вузлів через перенавантаження користувачами. У разі відсутності навантаження на систему ми отримаємо лише значення 4, оскільки ймовірність відмови вузлів будуть дорівнювати нулю, тому добуток також буде дорівнювати 0.

Варто зазначити, як саме визначається ймовірність відмови прийняття запиту вузлом. Будемо використовувати наступну функцію:

$$\boldsymbol{p_{x\_u}(u) = 1 - \frac{amt(u)}{x * amt(u-1)}},$$

де x - кількість вузлів відповідного типу;

u - кількість користувачів;

amt - функція, що відображає кількість варіантів запитів від u користувачів до x вузлів.

$$\boldsymbol{amt(u)} = \sum_{\boldsymbol{i_1,\ldots,i_x=\underline{0,x\_max}}} \boldsymbol{C^{u}_{i_1,\ldots,i_x}},$$

$i_1+\ldots+i_x = u,$

де x_max - кількість користувачів, яких може обробити один вузол відповідного типу.

Висновки до розділу 2

У даному розділі було розглянуто існуючі моделі розподілених баз даних. Окрім цього, була розглянута модель оцінювання параметрів обробки інформації в розподілених системах з динамічною структурою в яких реалізовано гетерогенні розподілені бази даних. А також запропонована модифікація моделі оцінювання параметрів.

РОЗДІЛ 3. ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ПІДТРИМКИ ЕФЕКТИВНОЇ ОБРОБКИ ДАНИХ У РОЗПОДІЛЕНИХ БАЗ ДАНИХ З ДИНАМІЧНОЮ СТРУКТУРОЮ.

3.1. Архітектура розподіленої бази даних з динамічною структурою.

Система розподілених баз даних з динамічною структурою має 3 основних компоненти: модуль керування, сховища розподілених баз даних, інтерфейс користувача.

Модуль керування відповідає за надання доступу до баз даних, керує обробкою пакетів та відповідей від баз даних.

Сховища розподілених баз даних включають в себе як розподілені бази даних, так і зовнішні джерела.

Інтерфейс користувача дозволяє робити запити для отримання необхідної інформації. Для цього створюється відповідний запит до модуля керування, який, у свою чергу, пересилає запит до бази даних, звідки інформація потрапляє безпосередньо до користувача.

Для моделювання архітектури розподіленої бази даних з динамічною структурою будемо використовувати графи. Як вершини позначимо вузли бази даних, а ребра будуть слугувати зв’язками між вузлами. На рисунку 3.1 зображено приклад структури. У якості атрибутів для кожної вершини буде використовуватися тип вузла: scn (керуючий, червоний колір), sein (проміжний, синій колір), sdbn (з програмним забезпеченням (ПЗ) для розподілених баз даних (РБД), помаранчевий колір). Будемо вважати, що для запитів можемо використовувати лише проміжні вузли.

Для ребер, у якості вагів, вказується швидкість передачі інформації між вузлами. Для вузлів scn та sdbn використовується атрибут “максимальна кількість користувачів, які можуть підключитися в один момент”, якщо кількість користувачів більше, ніж даний ліміт, то виникає черга до даного

ресурсу. Передбачається, що в цілому не може бути вся система у черзі, можуть бути лише деякі вузли недоступні через перевантаженість.

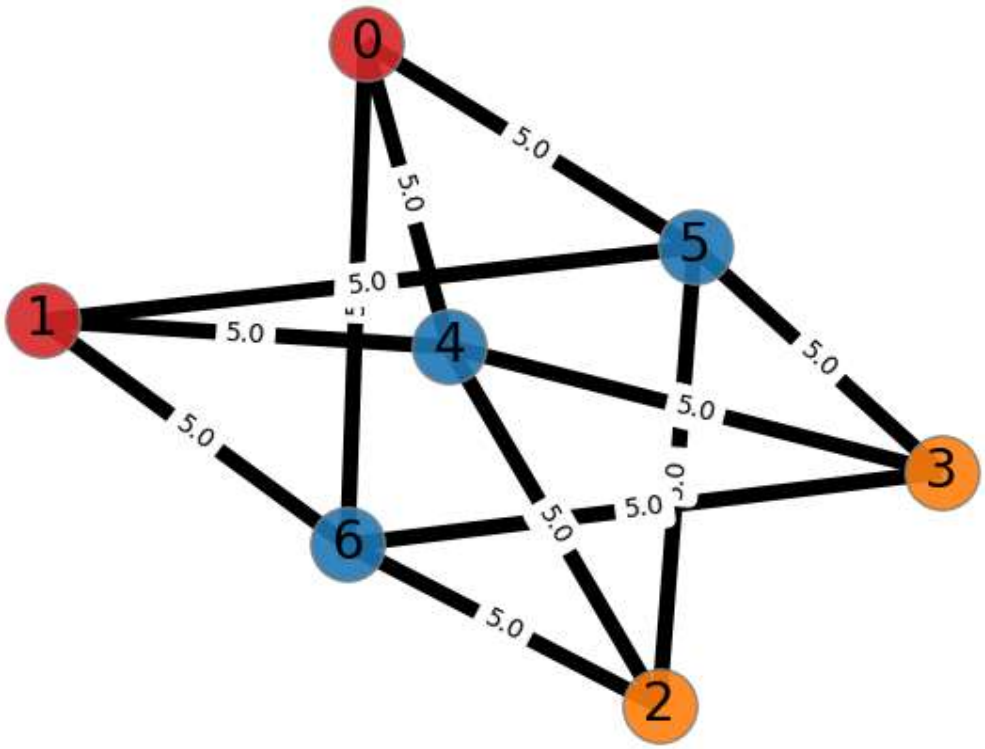

Рисунок 3.1 - Модель розподіленої бази даних.

Структура розподілених баз даних передбачає наявність мінімум двох керуючих вузлів, а також мінімум двох вузлів з ПЗ для РБД, оскільки при меншій кількості дана система перестає бути розподіленою та стає централізованою.

Передбачається, що проміжні вузли з’єднані з керуючими вузлами, та вузлами з ПЗ для РБД “кожен з кожним”. Таким чином, якщо один вузол перевантажений, то запит може піти до наступного вузла без будь-яких обмежень.

У моделі передбачається, що кожен раз запит направляється до будь-якого вузла випадковим чином. Якщо вузол перевантажений, то надсилається відповідь про те, що вузол не може прийняти запит через перевантаження, після чого надсилається запит на інший вузол.

3.2. Програмний комплекс для підтримки ефективної обробки даних у розподілених баз даних з динамічною структурою.

Модель системи розподіленої бази даних створено за допомогою мови програмування Python 3.10.3. Були використані наступні бібліотеки:

- networkx
- matplotlib.pyplot
- tkinter
- PIL

Перейдемо до опису кожної з них.

networkx - це бібліотека, що створює, оброблює та візуалізує графи.За її допомогою можна додавати вершини графа, зв’язки між вершинами, атрибути до кожного з елементів. Окрім цього, є досить зручним та зрозумілим інструментом для швидкої та гнучкої візуалізації графів. Під час створення програмного продукту дана бібліотека була використана для побудови графу, а також для його візуалізації. [13, 14]

matplotlib.pyplot - ця бібліотека дозволяє створювати візуалізацію, зокрема графів, графіків тощо. Є дуже гнучким інструментом та допомагає реалізувати майже будь-які побажання щодо візуального оформлення. Передаючи дані на вхід, за допомогою даної бібліотеки, на виході отримуємо вже готові візуальні об’єкти. Також дана бібліотека підходить для графічного аналізу результатів роботи. Я використав дану бібліотеку у своєму програмному комплексі для візуалізації графу, для побудови графіків залежності часу обробки запитів від кількості користувачів системи розподіленої бази даних. [15]

tkinter - бібліотека для створення графічного інтерфейсу. За допомогою даної бібліотеки можна створювати окремі вікна, в них текстові поля, кнопки, примітки, вводити і виводити інформацію у зручному та зрозумілому для користувача вигляді. А також виводити графічні елементи, наприклад, для відображення результатів роботи програмного комплексу. Дана бібліотека під

час створення програмного продукту була використана для створення графічного інтерфейсу, введення та виведення інформації у програмі, відображення графічних елементів, таких як графіки та графи. [16]

PIL - це open-source бібліотека для роботи з растровою графікою. Ця бібліотека забезпечує розширену підтримку форматів файлів, ефективне внутрішнє представлення та досить потужні можливості обробки зображень. Вона розроблена для швидкого доступу до даних, що зберігаються в кількох основних форматах пікселів. Забезпечує міцну основу для загального інструменту обробки зображень. Дана бібліотека була використана у програмному продукті для обробки, збереження та завантаження зображень, що генерувалися у програмі, під час її роботи. [17]

Перейдемо до опису кожного з модулів програми. На рисунку 3.2 представлено UML-діграму програмного продукту.

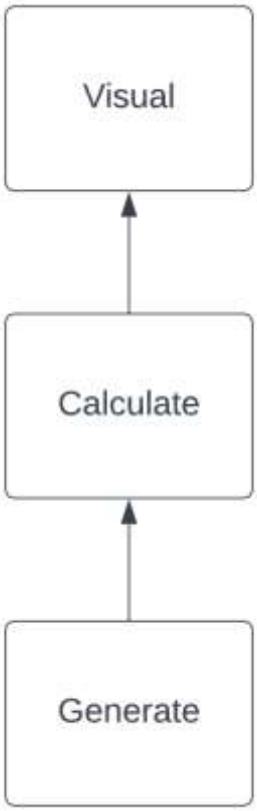

Рисунок 3.2 - UML - діаграма програмного продукту.

Модуль visual - створює інтерфейс для роботи з програмою. Використовувалась бібліотека для роботи з графічним інтерфейсом - tkinter. Було створено початкове вікно для роботи з програмою, яке зображено на рисунку 3.3.

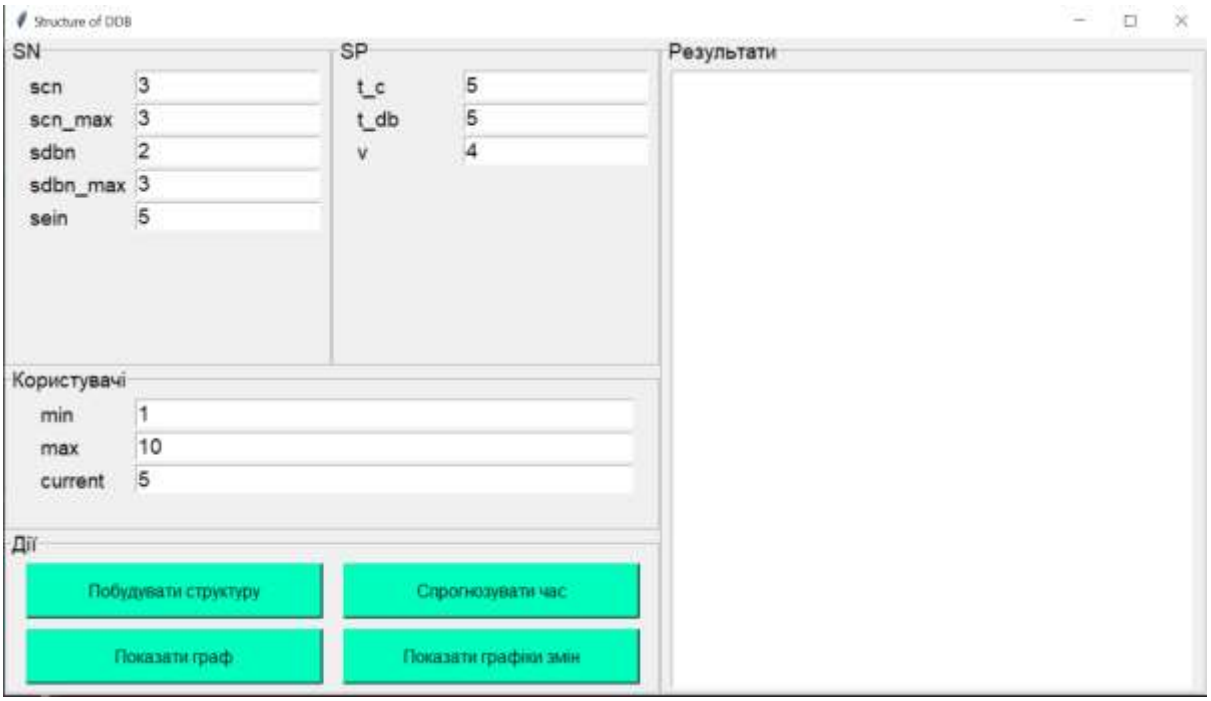

Рисунок 3.3 - Початкове вікно програми.

Тут можна задати структуру розподіленої бази даних, а саме: змінити кількість вузлів різних типів, а також надати обмеження на вузли. Окрім цього, можна задати швидкість обробки, та час обробки запитів вузлами.

Також можна задати обмеження на кількість користувачів, що працюють у даній системі та теперішню кількість користувачів, для підрахунку точного часу обробки запиту.

На рисунку 3.3. зображено також 4 кнопки:

- Побудувати структуру;
- Показати граф;
- Спрогнозувати час;
- Показати графіки змін.

“Побудувати структуру” - ця кнопка зчитує дані з полів для введення даних. Також будує структуру розподіленої бази даних. Якщо максимальна кількість користувачів буде більшою, ніж дозволяє обробити система, то буде

видано помилку про це, та застосовано максимально можливу кількість користувачів, враховуючи особливості системи.

“Показати граф” - створює графічну візуалізацію графу, який відображає модель системи розподіленої бази даних. Приклад такої системи зображено на рисунку 3.4. Червоним зображено керуючі вузли, помаранчевим - вузли з ПЗ для РБД, синім - проміжні вузли, з яких можуть надсилатися запити.

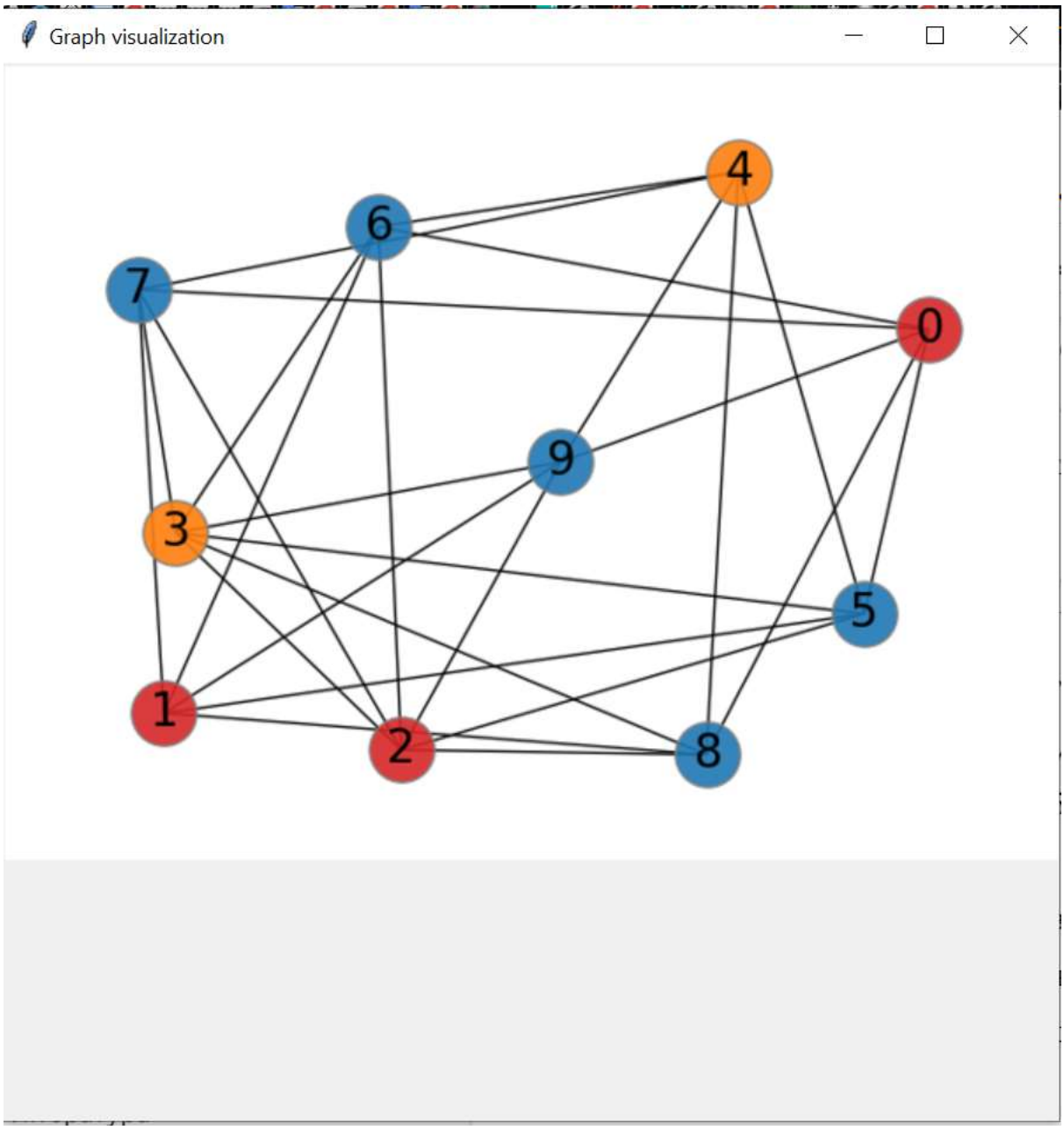

Рисунок 3.4 - Приклад візуалізації графу.

“Спрогнозувати час” - програмний комплекс створює прогноз часу обробки запиту користувача, враховуючи усі параметри системи. При чому час прогнозується для конкретної кількості користувачів, які в поточний момент

користуються системою. Приклад роботи даної кнопки зображено на рисунку 3.5.

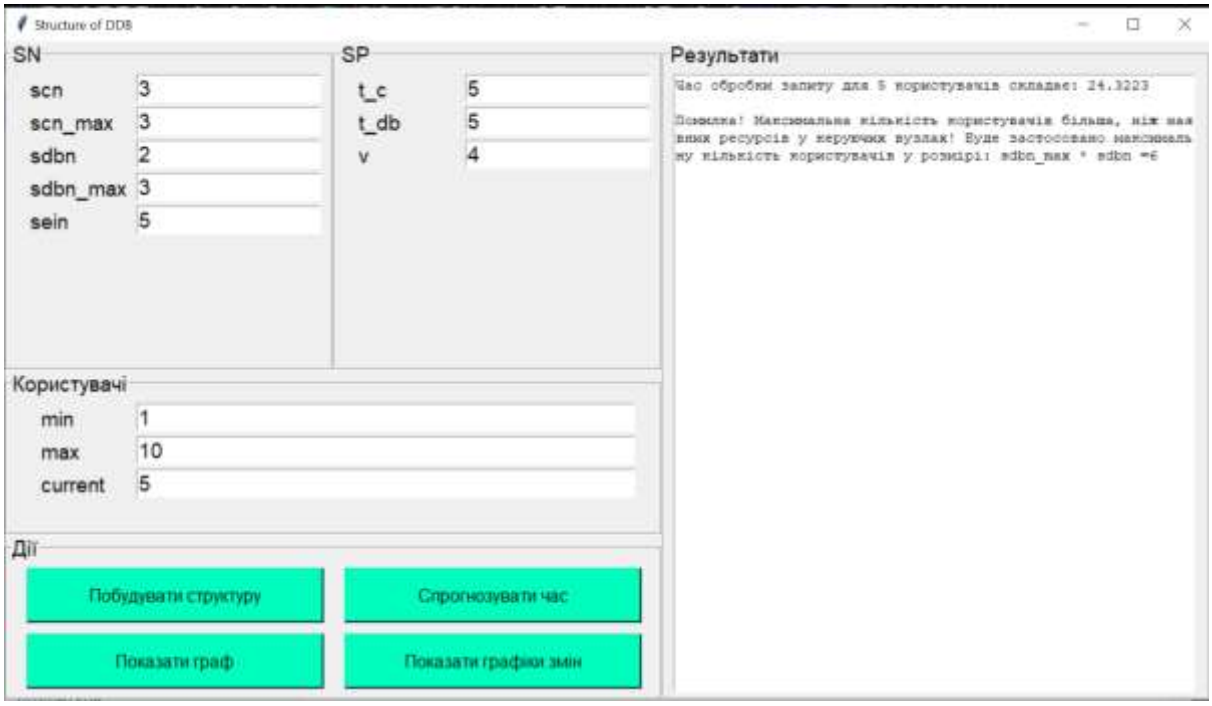

Рисунок 3.5 - Приклад прогнозування часу обробки запиту.

“Показати графіки змін” - створює графік, що відображає залежність часу обробки запитів від кількості користувачів. Обмеження мінімальної та максимальної кількості користувачів задається на початковому вікні. Приклад роботи зображено на рисунку 3.6.

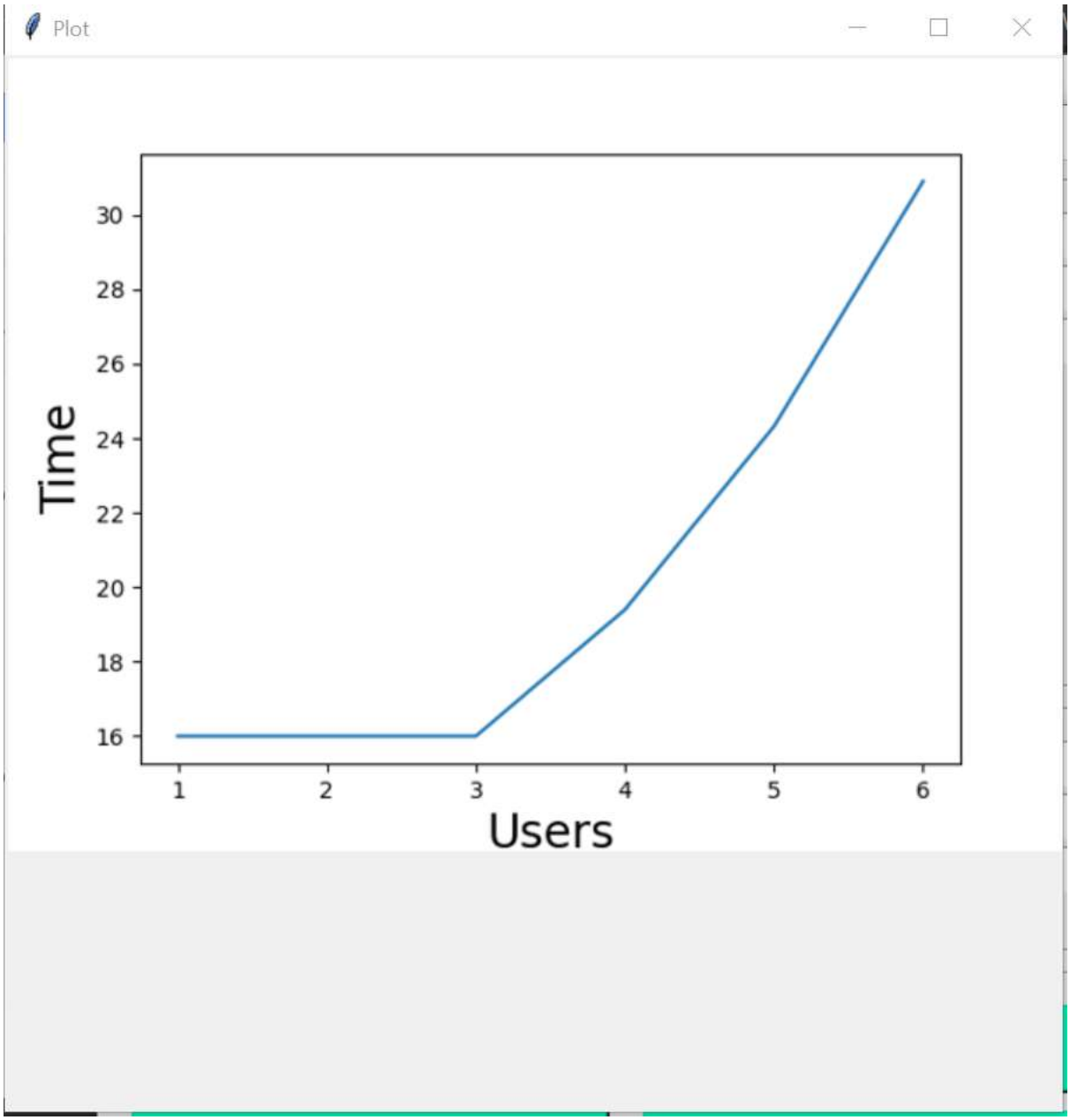

Рисунок 3.6 - Приклад побудови графіку.

При використані програми та при зміні структури системи важливо кожен раз будувати нову структуру натисканням на кнопку “Побудувати структуру”.

Модуль generate - генерує граф, що відображає модель розподіленої бази даних. Також створює візуалізацію структури. До нього входять наступні функції:

- create_graph - функція ініціалізує граф та задає параметри, що були задані у попередньому модулі через інтерфейс. При створені графу функція враховує тип вузлів, з’єднує всі проміжні вузли з кожним керуючим, та з кожним вузлом з ПЗ для РБД. На виході видає граф.

- plot_graph - створює візуалізацію моделі розподіленої бази даних та зберігає дане зображення.

Модуль calculate - підраховує витрати ресурсів у моделі під час запиту, використовуючи запропоновану модель оцінки, що враховує навантаженість користувачами. До нього входять наступні функції:

- combination - підраховує кількість комбінацій з повтореннями.
- prepear_for_combination - створює набір параметрів для підрахунку комбінацій, враховуючи особливості моделі.
- count - підраховує загальну кількість можливих підключень користувачами до вузлів.
- probability - підраховує ймовірність успішного підключення U-го користувача до вузла, при умові, що попередні U-1 користувач підключені до відповідних вузлів.
- length_way - визначає “довжину” шляху запиту, враховуючи навантаження системи іншими користувачами.
- expected_time - підраховує час виконання запиту у моделі розподіленої бази даних.

3.3. Опис методів та моделей, що застосовані у програмному комплексі для оцінки часу обробки даних у розподілених базах даних

Під час створення програмного комплексу було застосовано формули та функції для оцінки часу обробки запитів у системі розподілених баз даних.

У модулі calculate застосовано наступні функції:

- combination - підрахунок комбінацій з повторами:

$$C^{n}_{i_1,\dots,i_k} = \frac{n!}{i_1!*\dots*i_k!},$$

причому $i_1+\dots+i_k = n$

- prepear_for_combination - створює комбінації коефіцієнтів $i_1, \ldots, i_k$ для функції combination, що задовольняють наступним умовам:
  1. $i_1 + \ldots + i_k = u$, де u - кількість користувачів;
  2. k - кількість вузлів;
  3. $\forall j = 1, \ldots k: i_j < c\_max$, де c_max - кількість користувачів, яких може обробити 1 вузол.
- count - підраховує кількість варіантів підключення проміжних вузлів до вузлів керування, або до вузлів з ПЗ для РБД:

$count = \sum C^n_{i_1,\ldots,i_k}$, причому коефіцієнти $i_1, \ldots, i_k$ беруться з функції prepear_for_combination

- probability - підраховує ймовірність того, що наступний користувач успішно під'єднається до вузла:

Якщо u>1:

$$\boldsymbol{p(u) = \frac{count(u)}{k * count(u-1)}}$$

якщо u = 0 або u = 1: $p(u) = 1$, оскільки при нормальному функціонуванні системи завжди можна приєднати першого користувача. А для 0 користувачів - система просто існує.

- length_way - визначає довжину шляху запиту:

$$\mathbf{L = 4 + 2 * (scn - 1) * \left(1 - probability(u, scn, scn_{max})\right) + 2 * * (sdbn - 1) * (1 - probability(u, sdbn, sdbn\_max))}$$

Дана функція складається з суми 3х частин:

1. 4 - запит успішно надійшов від проміжного вузла до вузла керування, від вузла керування назад до проміжного вузла, потім до вузла з ПЗ для РБД і знов повернувся до проміжного вузла.
2. $2 * (scn - 1) * (1 - probability(u, scn, scn\_max)$- запит надійшов від проміжного вузла до вузла керування та повернувся з повідомленням про перевантаження вузла. $(scn - 1)$- відповідає за кількість вузлів керування, в які може надійти запит від проміжного вузла, але повернутися з помилкою про перевантаження.

Віднімаємо 1, оскільки в 1 вузол у будь-якому випадку запит надійде успішно (відповідно до постановки моделі).

$(1 - probability(u, scn, scn\_max))$- ймовірність того, що запит потрапить до перевантаженого вузла керування.

3. $2 * (sdbn - 1) * (1 - probability(u, sdbn, sdbn\_max))$ - запит надійшов від проміжного вузла до вузла з ПЗ для РБД та повернувся з повідомленням про перевантаження вузла. $(sdbn - 1)$- відповідає за кількість вузлів з ПЗ для РБД, в які може надійти запит від проміжного вузла, але повернутися з помилкою про перевантаження. Віднімаємо 1, оскільки в 1 вузол у будь-якому випадку запит надійде успішно (відповідно до постановки моделі).

   $(1 - probability(u, sdbn, sdbn\_max))$- ймовірність того, що запит потрапить до перевантаженого вузла з ПЗ для РБД.

- expected_time - функція, що робить оцінку часових витрат на запит:

$$\boldsymbol{t = tc + tc * (scn - 1) * \left(1 - probability(u, scn, scn_{max})\right) + tdb + tdb * (sdbn - 1) * \left(1 - probability(u, sdbn, sdbn_{max})\right) + \frac{length_{way(scn, sdbn, u, scn_{max}, sdbn_{max})}}{v}}$$

Дана функція також складається з 3х частин:

1. $tc + tc * (scn - 1) * (1 - probability(u, scn, scn\_max))$ - час обробки вузла керування. Складається з безпосереднього часу обробки запиту вузлом керування, а також з ймовірними відмовами через перевантаження певної кількості вузлів.
2. $tdb + tdb * (sdbn - 1) * (1 - probability(u, sdbn, sdbn\_max))$ час обробки вузла з ПЗ для РБД. Складається з безпосереднього часу обробки запиту вузлом з ПЗ для РБД, а також з ймовірними відмовами через перевантаження певної кількості вузлів.

3. $\frac{length_{way(scn,sdbn,u,scn_{max},sdbn_{max})}}{v}$ - кількість передач між вузлами, що діляться на швидкість передачі даних. Передбачається, що різниця у відстані між вузлами мала, та нею можна знехтувати.

У модулі generate застосовано наступні функції:

- create_graph - створює граф у відповідності з переданими параметрами. Зв'язки між вузлами будуються у відповідності до архітектури системи розподіленої бази даних. Тобто усі проміжні вузли з'єднані з усіма керуючими та з усіма вузлами з ПЗ для РБД.
- plot_graph - будує візуалізацію для графа, який був створений за допомогою попередньої функції.

Висновки до розділу 3

У даному розділі було описано архітектуру розподіленої бази даних з динамічною структурою, програмний комплекс, яка була застосована мова програмування та які бібліотеки були використані. Також було описано усі модулі програмного комплексу, усі функції та формули, що були створені для програми, а також наявний інтерфейс, який допомагає користувачам швидше, ефективніше та зручніше працювати з програмою.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНИЙ АНАЛІЗ МЕХАНІЗМІВ ПІДТРИМКИ ЕФЕКТИВНОЇ ОБРОБКИ ДАНИХ У РОЗПОДІЛЕНИХ БАЗ ДАНИХ З ДИНАМІЧНОЮ СТРУКТУРОЮ.

4.1. Постановка експериментів

Побудуємо різні моделі розподілених баз даних та розберемо як вони впливають на час обробки запитів, а також як впливає кількість користувачів.

Будемо змінювати кількість вузлів кожного типу, час обробки на вузлах, швидкість передачі даних, а також кількість користувачів, що користуються системою.

Частина експериментальних даних буде взята з джерела [1].

4.2. Експериментальні дослідження параметрів розподілених баз даних з динамічною структурою.

4.2.1. Експеримент 1.

Для початку побудуємо відносно просту модель з джерела [1] з такими параметрами:

- scn = 2;
- scn_max = 13;
- sdbn = 3;
- sdbn_max = 9;
- sein = 25;
- t_c = 3;
- t_db = 5;
- v = 5;
- min_users = 1;

- max_users = 25;
- current_users = 15.

Заповнення цих параметри у програмний комплекс показано на рисунку 4.1.

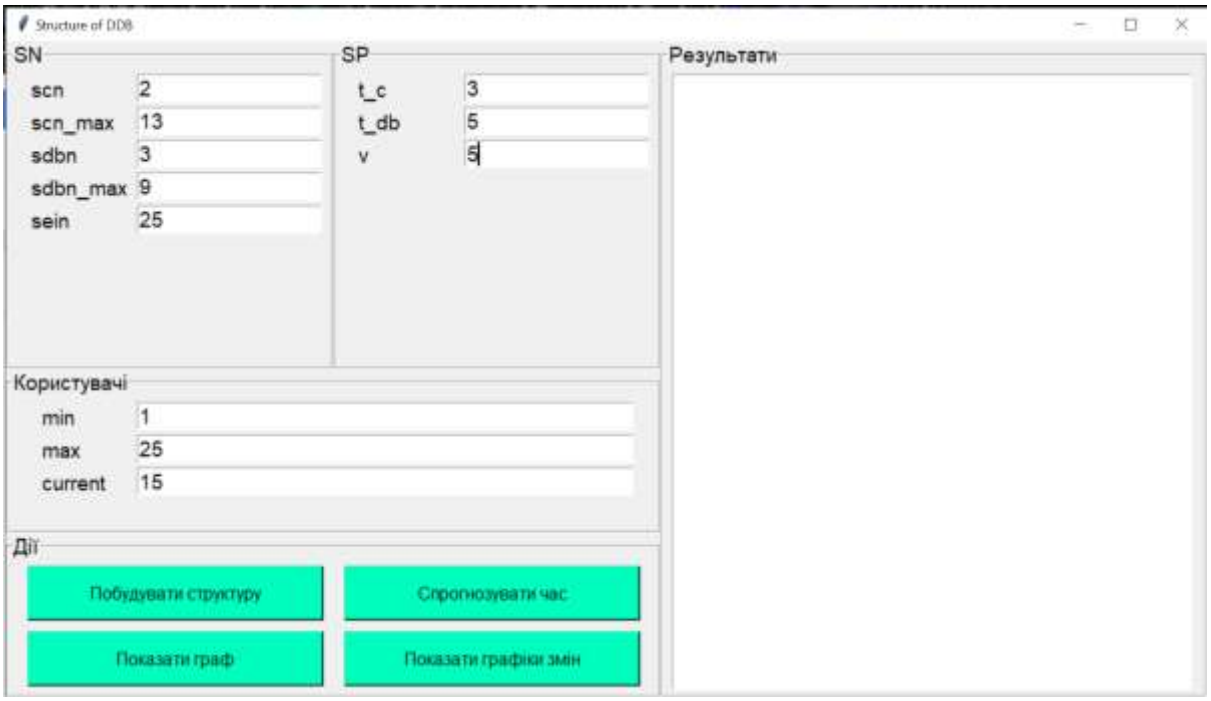

Рисунок 4.1 - Параметри моделі для експеримента 1.

Побудуємо граф. Його візуалізація представлена на рисунку 4.2.

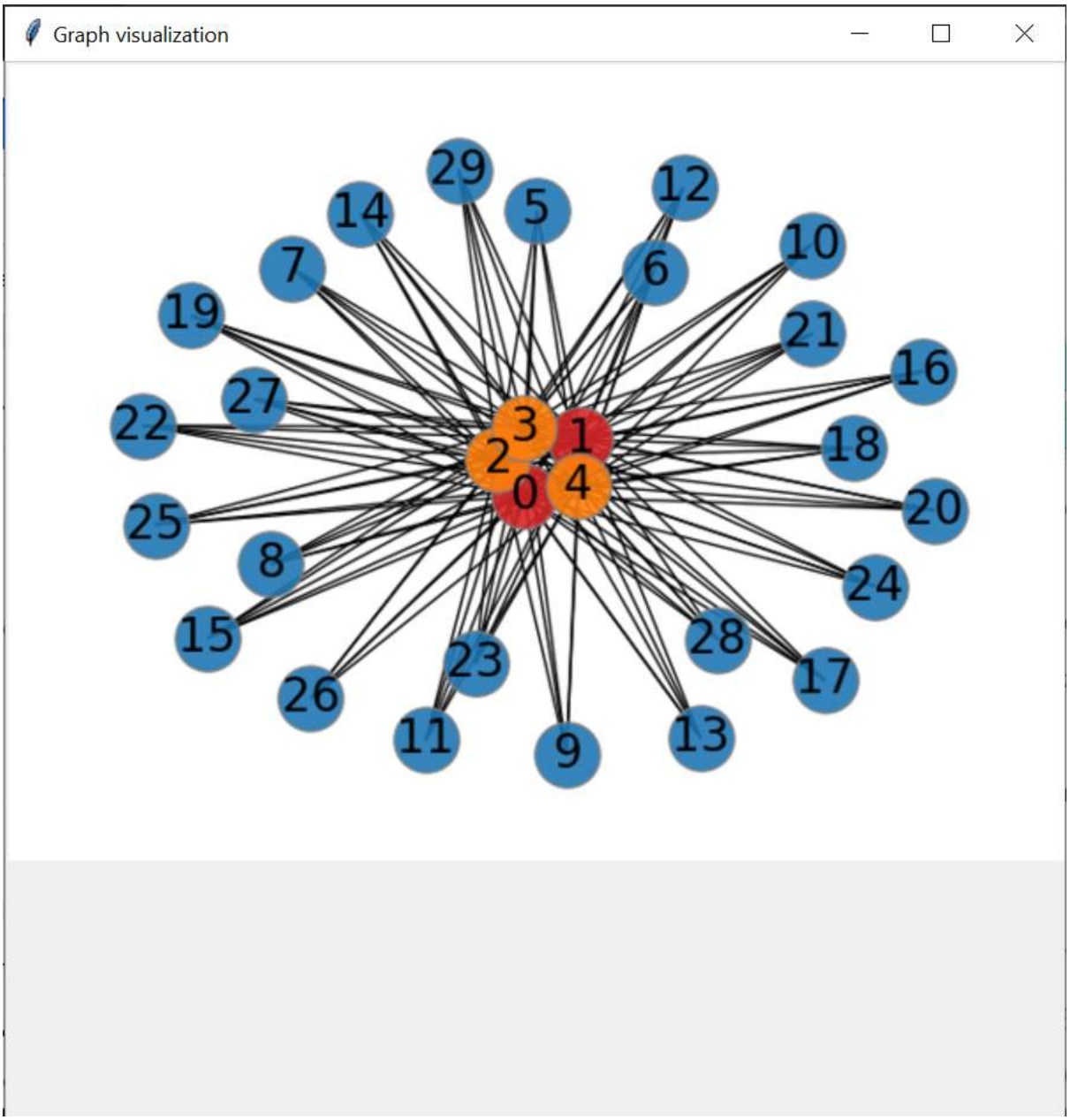

Рисунок 4.2 - Побудований граф для експерименту 1.

Прогнозний час обробки запиту для даної моделі, для 15 користувачів складає 8.9493 од. (рисунок 4.3)

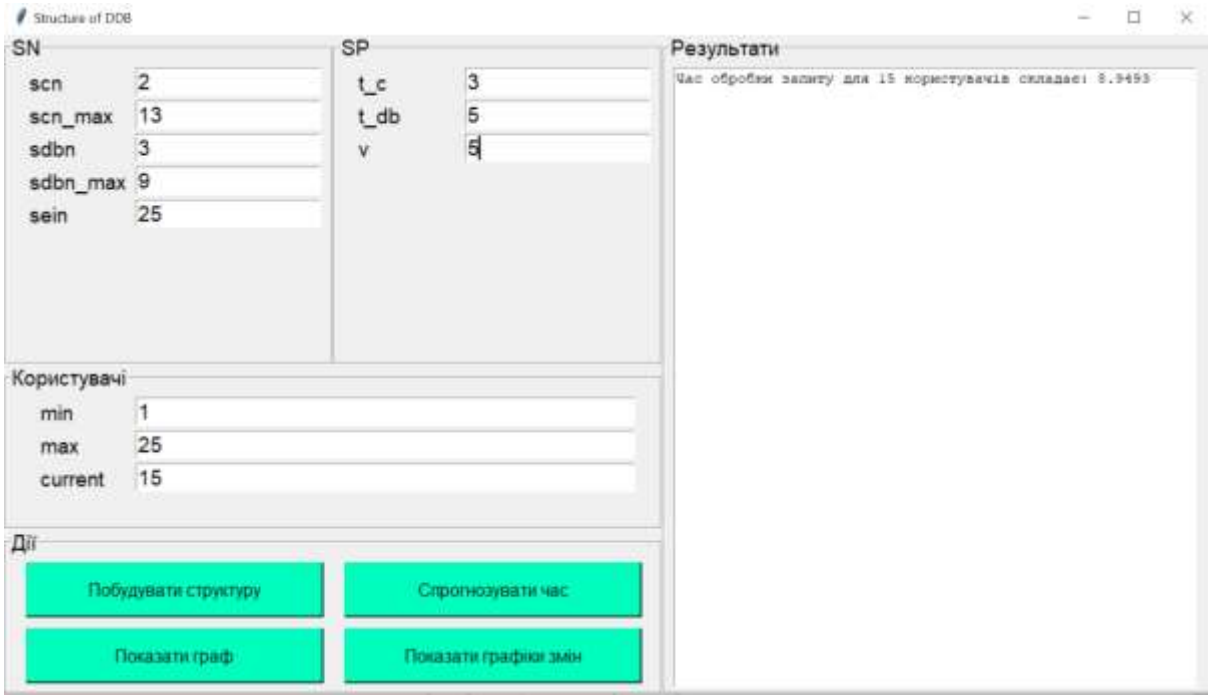

Рисунок 4.3 - Час обробки запиту у моделі розподіленої бази даних для 15 користувачів, для експерименту 1.

Графік залежності часу обробки від кількості користувачів зображено на рисунку 4.4.

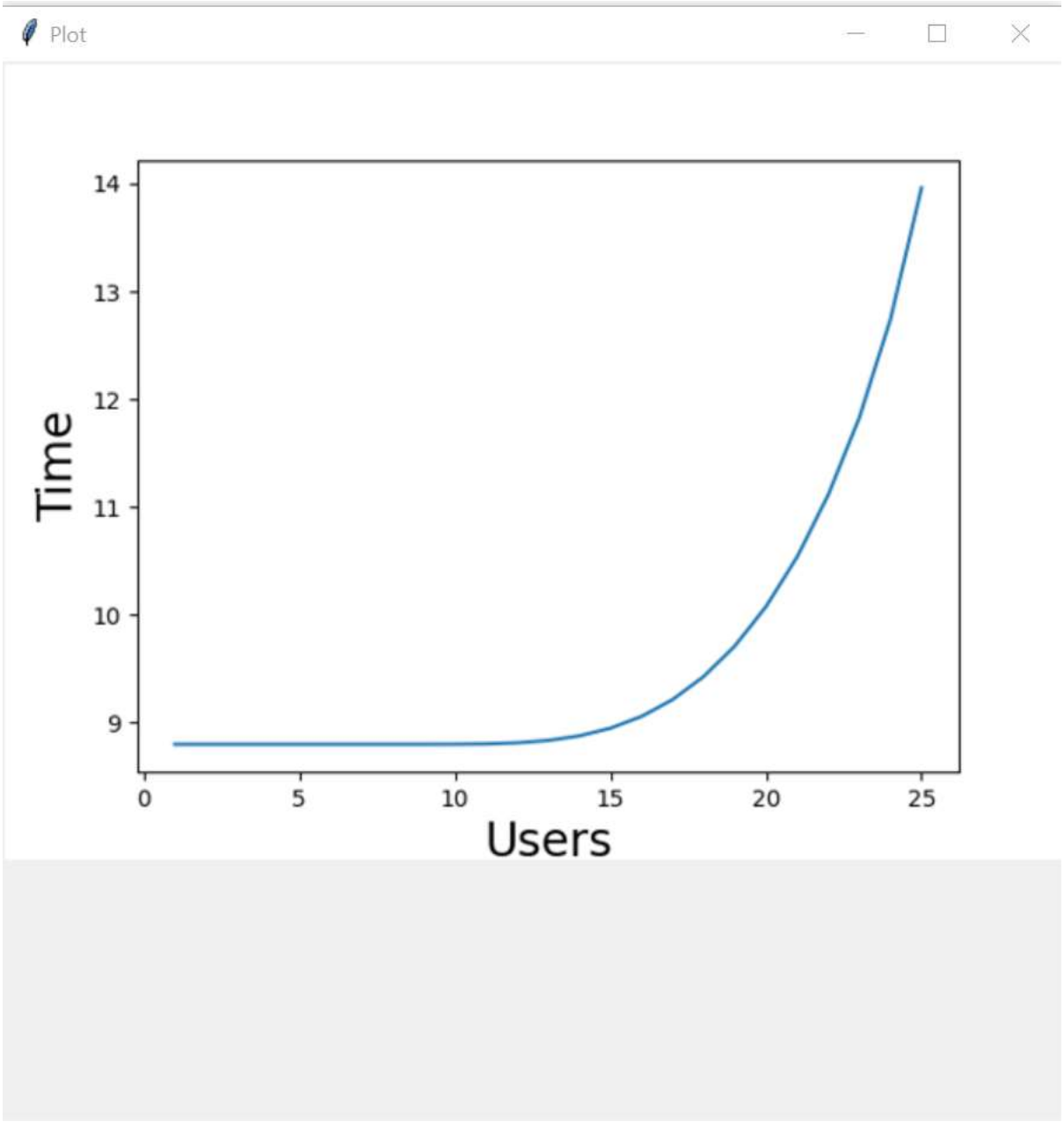

Рисунок 4.4 - Графік залежності часу обробки запиту від користувачів в розподіленій базі даних для експерименту 1.

Можна побачити, що час обробки починає різко зростати, майже експоненційно, коли кількість користувачів починає перевищувати ліміт обробки вузлів, тобто після 13 користувачів починається зростання часу обробки.

4.2.2. Експеримент 2.

Збільшимо кількість вузлів керування до 4. Граф моделі розподіленої бази даних зображено на рисунку 4.5.

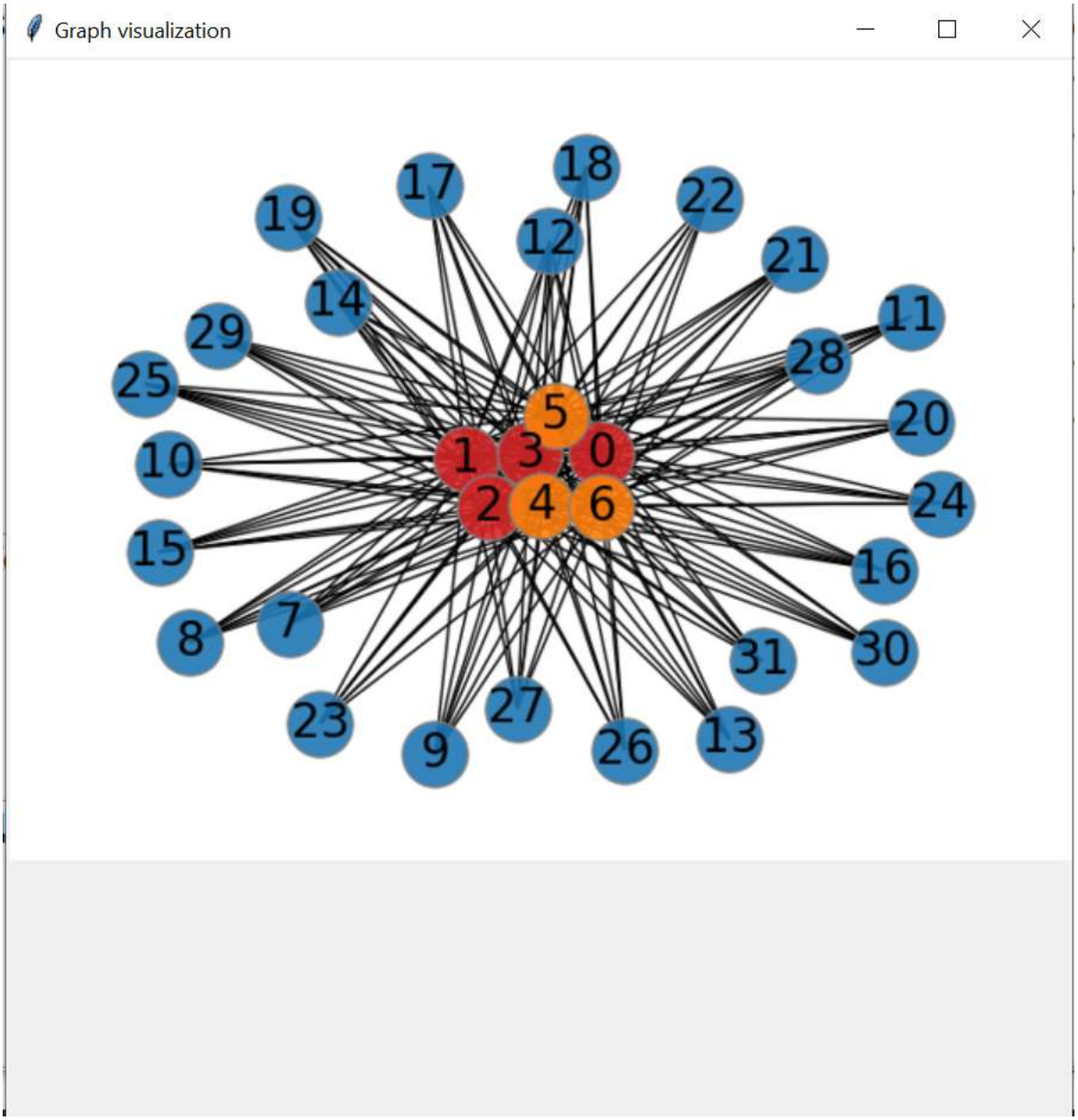

Рисунок 4.5 - Побудований граф для експерименту 2.

Час обробки запиту для нової моделі, та для тієї ж кількості користувачів склала 8.9464 од. Результат роботи програми зображено на рисунку 4.6.

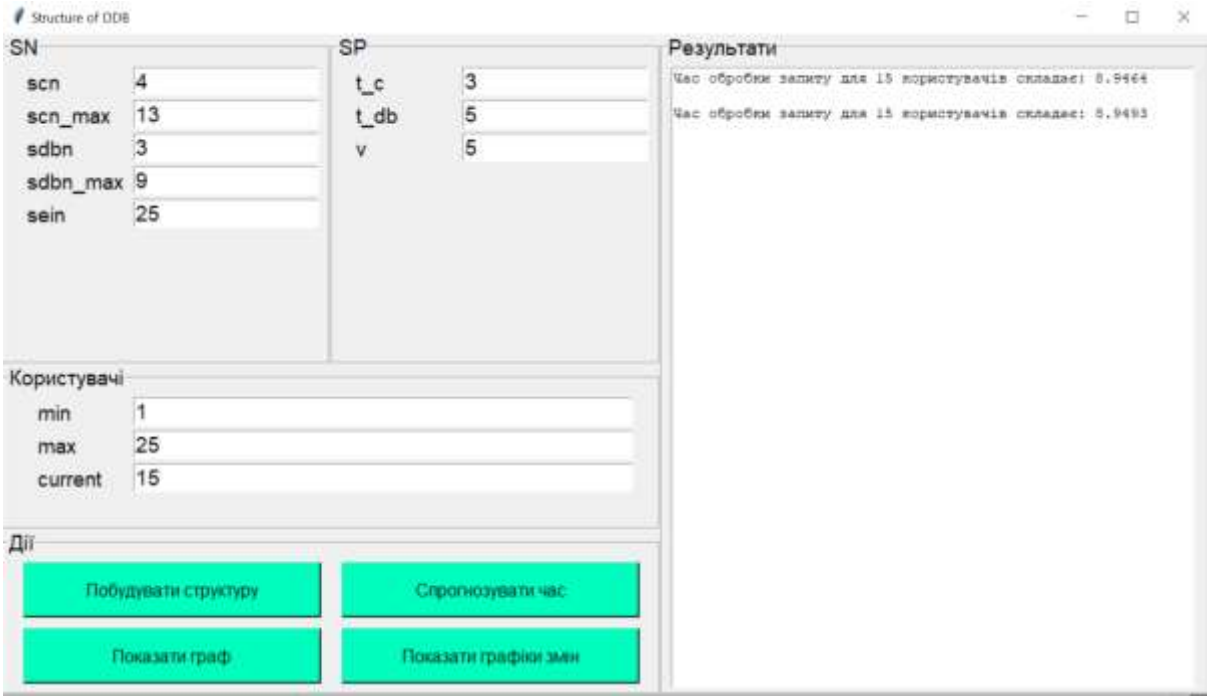

Рисунок 4.6 - Час обробки запиту у моделі розподіленої бази даних для 15 користувачів, для експерименту 2.

З рисунку 4.7 можна побачити на графіку залежності часу обробки від кількості користувачів, що збільшення кількості керуючих вузлів майже не дало впливу на час обробки.

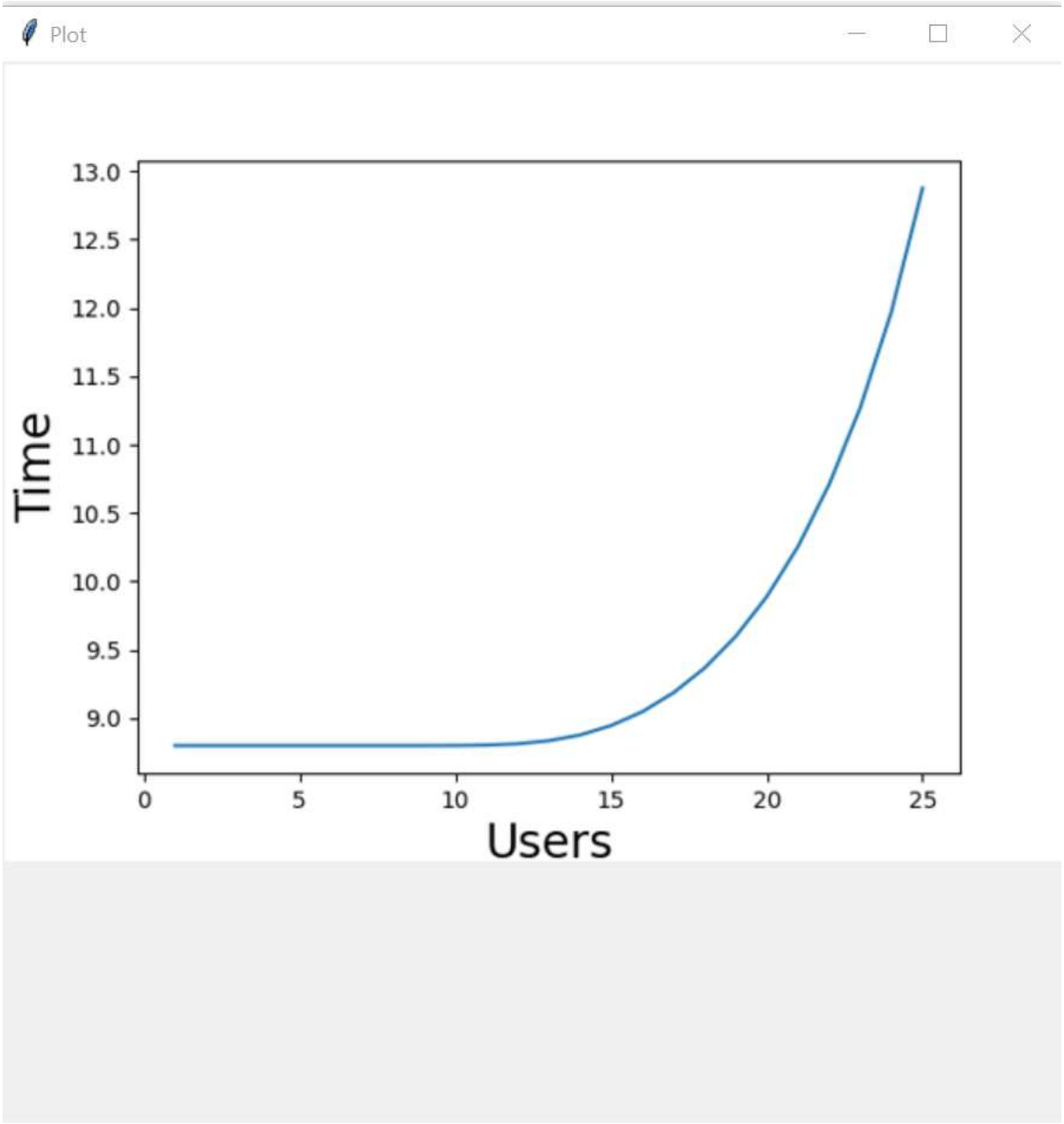

Рисунок 4.7 - Графік залежності часу обробки запиту від користувачів в розподіленій базі даних для експерименту 2.

4.2.3. Експеримент 3.

Тепер модифікуємо 1 модель, збільшивши кількість користувачів, що може обробити вузол з ПЗ для РБД до 13. Параметри у програмі зображено на рисунку 4.8.

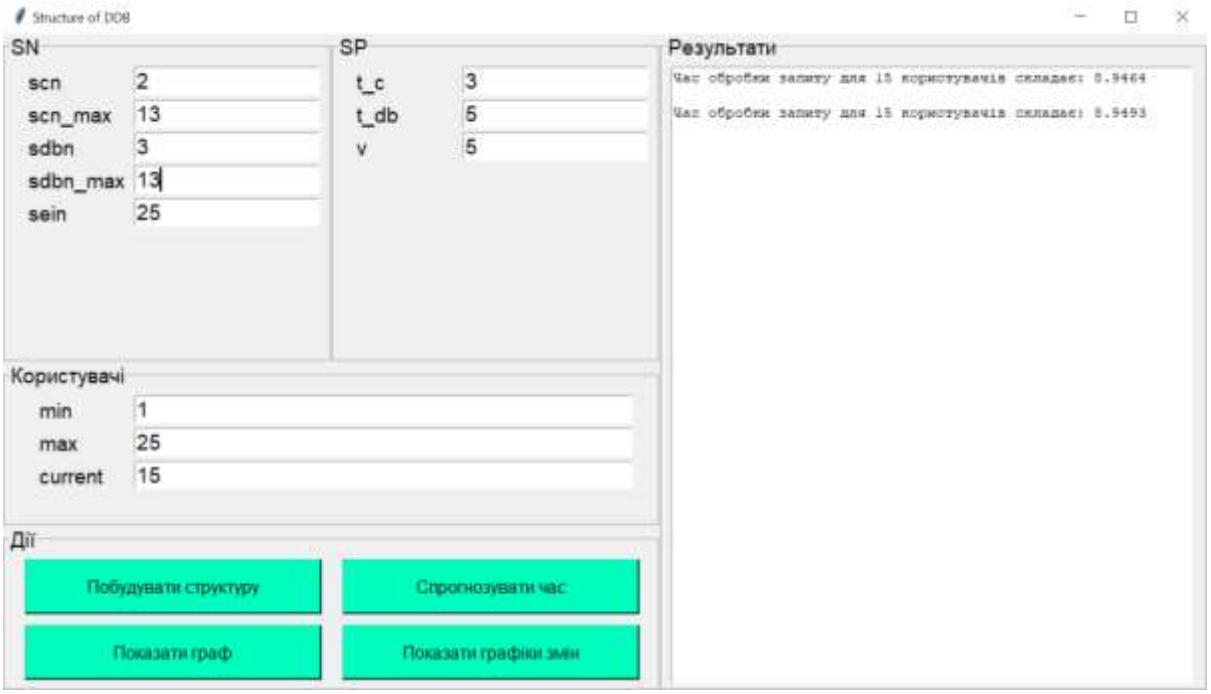

Рисунок 4.8 - Параметри для експерименту 3.

Час обробки для поточної моделі, для 15 користувачів склав 8.8030 од. Результат роботи програми зображено на рисунку 4.9.

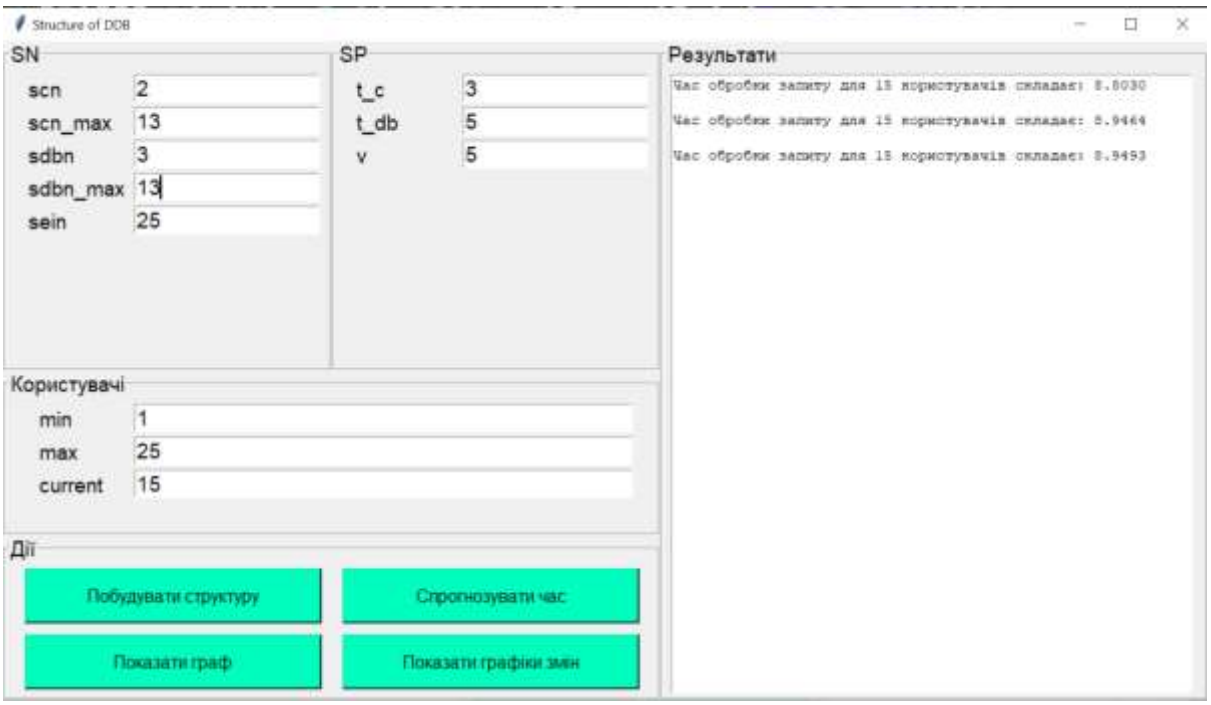

Рисунок 4.9 - Час обробки запиту у моделі розподіленої бази даних для 15 користувачів, для експерименту 3.

Графік залежності часу від кількості користувачів, що зображено на рисунку 4.10, показує, що збільшення обмеження кількості користувачів на

вузлі з ПЗ для РБД більше вплинуло на час обробки. В даному випадку час почав зростати після 17 користувачів.

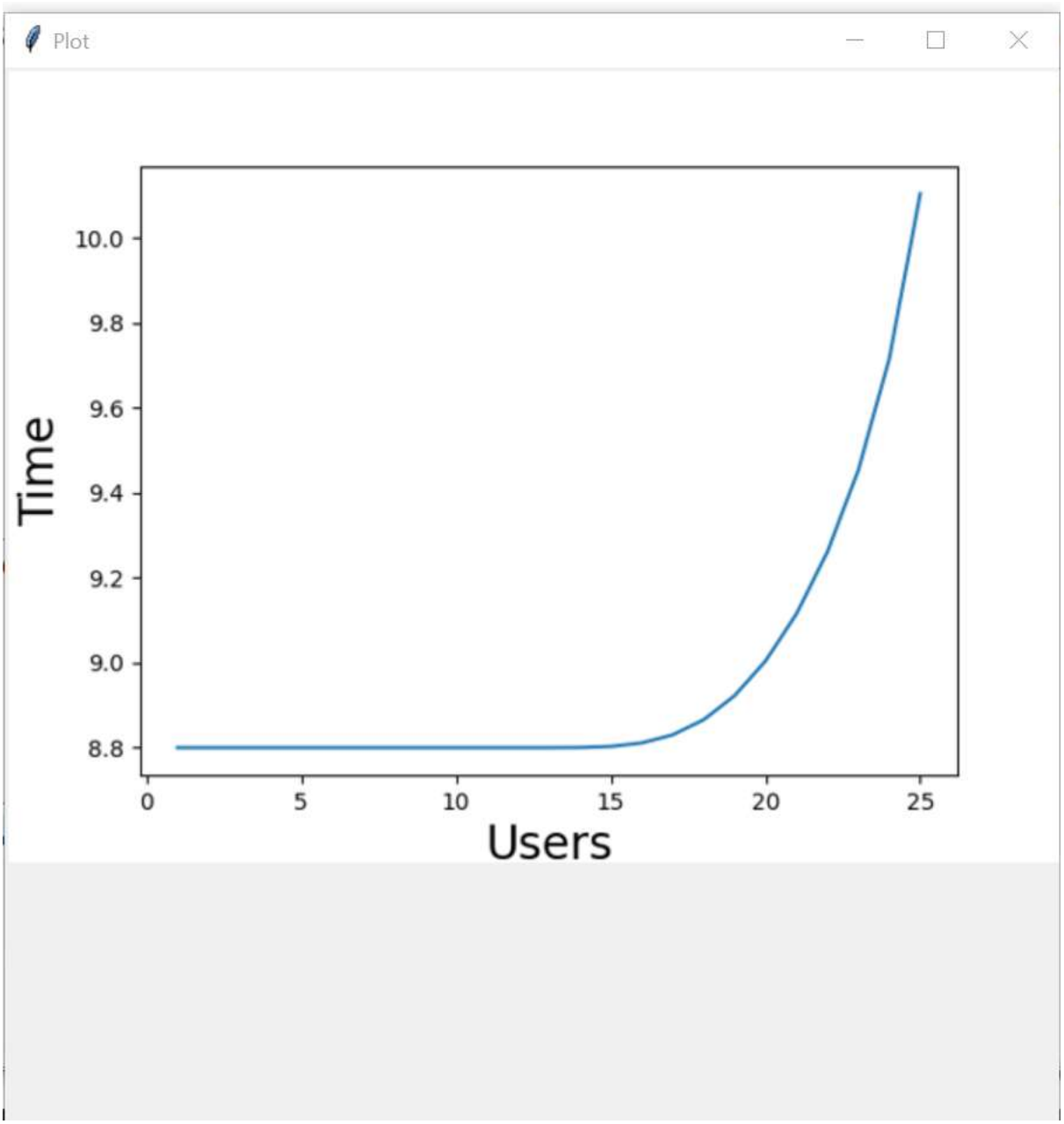

Рисунок 4.10 - Графік залежності часу обробки запиту від користувачів в розподіленій базі даних для експерименту 3.

4.2.4. Експеримент 4.

Спробуємо збільшити кількість керуючих вузлів та вузлів з ПЗ для РБД до 5. Обмеження на кількість користувачів, що можуть приєднатися до вузлів вкажемо по 13. Кількість користувачів збільшимо до 50.

Як видно з графіку, що на рисунку 4.11, то значне зростання часу обробки сталося після 30 користувачів.

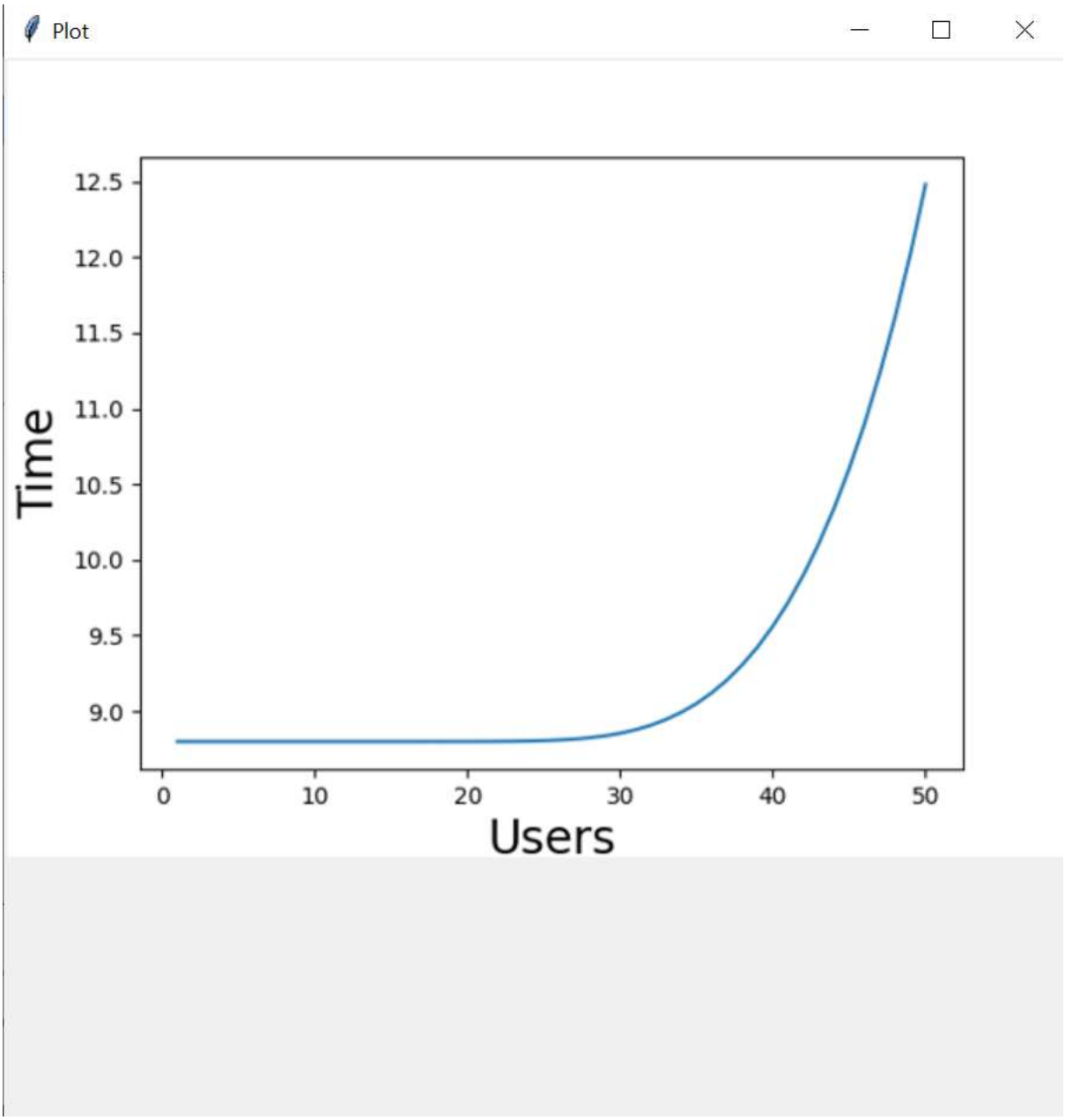

Рисунок 4.11 - Графік залежності часу обробки запиту від користувачів в розподіленій базі даних для експерименту 4.

4.3. Консолідований аналіз результатів.

Побудуємо таблицю 4.1 зі змінними параметрами та часом обробки запиту для фіксованої кількості користувачів.

Таблиця 4.1. Аналіз результатів

№	srn	srn_max	sdbn	sdbn_max	sein	t_c	t_db	v	users	time
1	2	13	3	9	25	3	5	5	20	10.0767
2	4	13	3	9	25	3	5	5	20	9.8896
3	6	13	3	9	25	3	5	5	20	9.8889
4	2	15	3	9	25	3	5	5	20	9.9141
5	2	17	3	9	25	3	5	5	20	9.8900
6	2	13	3	11	25	3	5	5	20	9.1718
7	2	13	3	13	25	3	5	5	20	9.0040
8	4	13	3	13	25	3	5	5	20	8.8169
9	4	13	3	13	25	5	5	5	20	10.8173
10	4	13	3	13	25	5	3	5	20	8.8113
11	3	13	3	13	25	5	3	5	20	8.8263
12	3	13	3	13	25	5	5	5	20	10.8323
13	3	13	3	13	25	3	5	5	20	8.8263
14	2	13	3	9	25	3	5	7	20	9.8188
15	2	13	3	9	25	3	5	3	20	10.6786
16	2	13	3	9	25	3	5	5	19	9.7123
17	2	13	3	9	25	3	5	5	21	10.5376

Можна помітити, що у рядках 1-3 змінювалась кількість вузлів керування. Вплив на час обробки є незначним.

У рядках 1, 4-5 змінювалось обмеження на вузлі керування. Вплив на час обробки також є незначним.

У рядках 11-13 змінювався час обробки на вузлах. При чому, інші характеристики вузлів були ідентичні. Як видно з результатів, враховуючи побудову моделі, то зміни симетричні.

У рядках 1, 6-7 змінювалось обмеження на вузлі з ПЗ з РБД. Але варто зосередити увагу на тому, що час обробки на вузлі з ПЗ для РБД більший, ніж на вузлі керування, тому і вплив на час обробки також є більшим.

У рядках 8-10 змінювався час обробки на вузлах. Проте варто зазначити, що інші обмеження та кількість вузлів є різною. Тому ці зміни не є симетричними, відносно вузлів керування та вузлів з ПЗ для РБД.

У рядках 1, 14-15 змінювалась швидкість передачі інформації. Як видно з таблиці, то вплив є, однак не дуже значним.

У рядках 1, 16-17 змінювалась кількість користувачів. Як видно з результатів, то даний вплив є суттєвим.

Отже, для того, аби вплив на час обробки запиту був найбільшим - варто змінювати декілька параметрів, наприклад, час обробки запиту вузлом, кількість тих чи інших вузлів, обмеження на кількість користувачів, що може обробити вузол, а також швидкість передачі інформації. Однак, враховуючи те, що швидкість передачі інформації мало залежить від структури системи розподіленої бази даних, та час обробки запиту вузлом також не є параметром, який легко змінити, то варто звернути увагу на кількість вузлів та на обмеження на користувачів одного вузла. Але, якщо створити робочий процес таким чином, аби користувачі використовували систему розподілених баз даних у різний час, тобто не навантажували її, то час обробки обробки запитів значно зменшується.

Висновки до розділу 4

У даному розділі були наведені результати експериментів із запропонованою моделлю. Також був проведений аналіз графіків та числових значень змін результатів в залежності від змін параметрів. Було виявлено, що найбільший вплив мають кількість користувачів. Але при комплексних змінах системи також є значний вплив на час обробки запиту.

Звідси можна зробити висновок, що якщо зменшити кількість користувачів, які працюють в один момент, то можна значно прискорити обробку запитів.

РОЗДІЛ 5. РОЗРОБКА ВЛАСНОГО СТАРТАП ПРОЕКТУ

Зараз у сучасному світі існує велика потреба ефективного використання баз даних. Наразі є можливість застосування розподілених баз даних. Для ефективної роботи з ними – важливо оцінювати час обробки запитів. Тому в результаті дослідження отримуємо модель, що допомагає оцінити часові витрати на обробку даних.

Все це означає, що проблема, яку вирішує стартап, являється актуальною і може стати успішним на ринку та стати основним вибором для переважної кількості цільової аудиторії.

5.1 План розробки стартапу та масштабування його на ринок

Наведемо план розробки стартапу та виведення його на ринок.

Спочатку треба провести маркетинговий аналіз, який включає в себе:

- конкурентний аналіз, щоб зрозуміти, якими методами вирішення проблем вже користуються люди;
- формування ідеї самого проекту та виділення цільової аудиторії;
- розробити стратегію виведення товару на ринок, базуючись на аналізі ринкового середовища.
- Наступним кроком являється організація самого стартапу. На цьому етапі мають бути:
- складений весь план та побудований таймлайн розробки та запуску продукту;
- запланований обсяг виробництва та оцінений потенційний обсяг ресурсу, який буде потрібен для виконання плану;

- розраховані витрати, необхідні для реалізації проекту, та витрати на запуск проекту.
- Далі необхідно виконати фінансово-економічний аналіз та оцінити ризики стартап-проекту, в межах якого:
- визначити обсяг інвестиційних втрат;
- розрахувати основні фінансово-економічні показники проекту (собівартість, ціну продукту/послуги, податковий збір та чистий прибуток) та визначити показники інвестиційної привабливості проекту (рентабельність продажів, період окупності проекту);
- визначити основні ризики проекту та способи для їх запобігання.

Фінальним кроком являється розробка заходів з комерціалізації продукту. Цей крок являється важливим для масштабування та збільшення розмірів продукту.

Для того, щоб залучити інвесторів та знайти різні способи фінансування проекту, необхідно:

- провести дослідження на предмет інтересів потенційних інвесторів та бізнесів;
- скласти інвестиційну пропозицію, яка включає в себе як опис самого продукту та його теперішні розміри, так і можливі шляхи розширення та розвитку;
- обрати канали комунікації із потенційно зацікавленими персонами.
- Далі наведемо результати виконання кожного з описаних кроків.

5.2 Опис ідеї стартап-проекту

Стартап-проект полягає у оцінці часу, який витрачається на обробку запиту у розподілених базах даних. Суть продукту стартапу полягає у тому, що

ми мажмо якомога краще прогнозувати даний час, враховуючи навантаження користувачів на систему.

У таблиці 5.1 наведена інформаційна карта стартапу.

Таблиця 5.1 – Інформаційна карта стартап-проекту

Назва проекту	ExpTime
Автори проекту	Васильченко Ілля Вячеславович
Коротка анотація	Модель буде оцінювати час обробки запиту в розподіленій базі даних
Термін реалізації проекту	9 місяців
Необхідні ресурси	Приміщення з комп'ютерами, доступом до Інтернету, доступ до електромережі Програмне забезпечення для розробки Фінансові кошти на оплату заробітної плати виконавцям на термін 9 місяців, а також на такі витрати як: оренда приміщення, комунальні послуги, оренда хмарного сховища тощо
Опис проблеми, яку вирішує проект	Продукт вирішує задачу оцінки часових витрат на запит у розподілених базах даних.
Головні цілі та завдання проекту	Метою проекту є створення програмного продукту, який на основі даних про розподілену базу даних буде спрогнозовано час обробки запиту, враховуючи навантаження користувачів на систему.

Продовження таблиці 5.1

Очікувані результати	Привернення технологічних компаній до нашого стартапу, покращення оцінки часу для запитів у розподілених базах даних.

5.3 Технологічний аудит ідеї проекту

Тепер можна розібрати ідею стартапу та провести конкурентний аналіз. У таблиці 5.2 наведений опис ідеї стартапу.

Таблиця 5.2 – Опис ідеї стартапу

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Основна ідея являється створення комплексної системи, яка буде приймати на початкові значення та повертатиме та результат моделювання	Оцінка часових витрат на запити до розподілених баз даних	Можна буде ефективно планувати робочі процеси, оптимізовувати час очікування
	Система збору відгуків про якість моделей для подальшого покращення системи	Система буде оновлюватись і якість прогнозування буде збільшуватись

Далі проведемо порівняльний аналіз конкурентів проекту та наведемо результати у таблиці 5.3.

Таблиця 5.3 – Порівняльний аналіз конкурентів проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів		W	N	S
		Власний проект	Запропонована модель оцінки			
1	Якість генерації передбачення	Надає доволі хороший результат	Свій власний алгоритм			+
2	Доступність по ціні	Безкоштовний при розробці	Потребує розробки під конкретну систему		+	
3	Персоналізація під регіон	Присутня	Відсутній		+	

Далі аналізуємо реальність технічно здійснити ідею проекту (таблиця 5.4).

Таблиця 5.4 – Технологічна здійсненність продукту

№ п/п	Ідея проекту	Технології і реалізації	Наявність технологій	Доступність технологій
1	Створення комплексної системи, яка буде приймати на початкові значення та повертатиме та результат моделювання	Використання мови програмування Python	Наявні	Доступні
2		Використання мови програмування C++	Не наявні, необхідні допрацювання	Доступні
3		Використання мови програмування C#	Наявні	Доступні
Обрана технологія реалізації ідеї проекту: Python				

5.4 Аналіз ринкових можливостей запуску стартап-проекту

Далі проведемо попередній аналіз ринку для запуску стартап-проекту (таблиця 5.5).

Таблиця 5.5 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники ринку (найменування)	Характеристика
1	Кількість головних гравців, од	1
2	Загальний обсяг продаж, грн/ум.од	5000
3	Динаміка ринку (якісна оцінка)	Позитивна, зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	12%

Тепер проведемо характеристику потенційних клієнтів, які можуть бути зацікавлені в проекті (таблиця 5.6).

Таблиця 5.6 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреби, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Оцінка часових витрат на запит до розподіленої бази даних	Користувачі баз даних	Цікавить загалом час очікування	Простота використання
2	Ефективне планування робочих процесів	Проектні менеджери	Особлива потреба у передбачені навантаження на систему	Динамчіна можливість планування результатів моделювання
3	Оптимізація витрат	Системні адміністратори / DEV OPS	Необхідність бачити можливості для оптимізації ресурсів, при чому не втрачаючи продуктивності	Можливість вносити зміни до структури

Обрахуємо фактори загроз (таблиця 4.7) та можливостей (таблиця 4.8). Проаналізуємо загрози, щоб зрозуміти можливі перешкоди при запуску продукт на ринок. Фактори можливостей же треба обрахувати, щоб знати усі сприятливі умови та по можливості ними скористатися.

Таблиця 5.7 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Хоча ринок є відкритим і неосвоїним, на ньому вже є кілька великих гравців, які відомі на ринку, які вже мають свою цільову групу	Знайти точки додаткової цінності для користувача
2	Ціна збуту	Конкуренти можуть коштувати менше через нижчу якість	Сфокусуватися на якості роботи моделі
3	Якість аналізу	Через комплексність задачі, моделі можуть мати проблеми на певних предметних областях	Мати достатній штаб і ресурси, для побудови різних моделей для різних предметних областей

Таблиця 5.8 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Універсальність	Продукт не залежить від конкретної структури розподілених баз даних	Зробити акцент при маркетингу, продовжувати розвиток як окремого продукту

Продовження таблиці 5.8

2	Простота у використанні	Від користувача треба лише ввести початкові дані	Реалізувати зручний інтерфейс для цього
3	Якість та гарантії	Надавати найбільш якісні послуги	Пропонувати моделі з найкращими результатами, а також надавати усю необхідну технічну підтримку
4	Безкоштовний сервіс при MVP	Максимально швидко набрати базу своїх клієнтів та заявити про себе на ринку	Розгорнути широкий маркетинг, а також активно боротися за клієнтів конкурентів

Далі розглянемо питання конкуренції, а саме визначимо її тип та рівень (таблиця 5.9).

Таблиця 5.9 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: недосконала конкуренція	Представлено мало продуктів та експертів	Зробити максимальним збут застосунку

Продовження таблиці 5.9

2. За рівнем конкурентної боротьби: міжнародний	Наявні проекти, розроблені та можуть бути доступні у всьому світі	Розширити цільову аудиторію, розробити інтерфейс на різних мовах
3. За галузевою ознакою: внутрішньогалузева	Можуть працювати з різними галузями	Покращити персоналізацію
4. Конкуренція за видами товарів: товарно-родова	Конкуренція з аналізами моделей	Підтримувати та покращувати якість існуючих факторів
5. За характером конкурентних переваг: нецінова	Різні компанії пропонують різну якість	Розробляти якісніші алгоритми і моделі
6. За інтенсивністю: марочна	Вже представлені компанії із сильним брендом	Предметно створити комунікаційну стратегію для вибудови свого бренду

Далі необхідно виконаємо аналіз конкуренції за моделлю 5 сил конкуренції Майкла Портера (таблиця 5.10).

Таблиця 5.10 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти у галузі	Потенційні конкуренти	Постачальники	Клієнти	Товаровиробники
	Інші існуючі системи та продукти	Якість, ціни, кількість користувачів, капіталовкладення	Фактори сили постачальників	Контроль якості, порівняння цін	Сила бренду, якість, ціна, масштаби
Висновки	Конкуренція з невеликою інтенсивністю, а також підігрітий ринок	Можливості входження на ринок, нові потенційні конкуренти	Постачальники відсутні	Клієнти не диктують умови роботи на ринку	Товаровиробники відсутні

Маючи результати аналізу конкуренції (таблиця 5.10), характеристики ідеї стартап-проекту (таблиця 5.5), характеристики потенційних клієнтів і їх вимоги до продукту (таблиця 5.6) та фактори ринкового середовища (таблиці 5.7 і 5.8) було сформульовано та обґрунтовано перелік факторів конкурентоспроможності (таблиця 5.11).

Таблиця 5.11 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Універсальність	Продукт не залежить від структури розподіленої бази даних
2	Простота у використанні	Від користувача треба лише ввести початкові дані
3	Якість та гарантії	Надавати найбільш якісні послуги та сервіси
4	Безкоштовний сервіс при MVP	Максимально швидко набрати базу своїх клієнтів та заявити про себе на ринку

Тепер можна провести аналіз сильних та слабких сторін продукту (таблиця 5.12).

Таблиця 5.12 – Порівняльний аналіз сильних та слабких сторін системи

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів						
			-3	-2	-1	0	1	2	3
1	Універсальність	20	+						
2	Простота у використанні	16		+					
3	Якість та гарантії	12		+					
4	Безкоштовний сервіс при MVP	15			+				

Далі проведемо SWOT-аналіз продукту (таблиця 5.13).

Таблиця 5.13 – SWOT-аналіз стартап-проекту

Сильні сторони Універсальність Простота у використанні Якість та гарантії Безкоштовний сервіс при MVP	Слабкі сторони Відсутність сильного бренду Не сформована база клієнтів Не підключені альтернативні канали маркетингу
Можливості Покращення системи	Загрози Нові системи та експерти

Дякуючи проведенню SWOT-аналізу, ми змогли визначити сильні та слабкі сторони, можливості та загрози, пов'язані з конкуренцією та плануванням стартап-проекту. Далі спроектуємо альтернативну ринкову поведінку для інтеграції стартап-проекту на ринок та приблизний час реалізації системного комплексу, з урахуванням потенційних проектів, що можуть бути виведені на ринок та наведемо результати у таблиці 5.14.

Таблиця 5.14 – Альтернативи ринкового впровадження стартап проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Вихід на ринок з нижче якістю	70%	3 місяці
2	Пропонувати одразу платне використання	50%	5 місяців

Продовження таблиці 5.14

3	Представлення користувачам системи без інтерфейсу	60%	6 місяців

У даному пункті був проведений детальний аналіз ринку та продукту. Також відповідно до результатів проведеного конкурентного аналізу, визначених факторів ринку та його сприятливість, описання ідеї та характеристик стартап-проекту, робимо висновок висновок, що існують дуже сприятливі умови для виходу продукту на ринок.

5.5 Розроблення ринкової стратегії стартап-проекту

Для розробки ринкової стратегії продукту, у першу чергу, необхідно проаналізувати цільову аудиторію проекту (таблиця 5.15).

Таблиця 5.15 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Користувачі баз даних	Висока	50%	Висока	Середня
2	Проектні мереджери	Середня	25%	Середня	Середня

Продовження таблиці 5.15

3	Системні адміністратори/ DEV OPS	Середня	25%	Середня	Середня
Які цільові групи обрано: 1, 3					

Маючи аналіз цільових груп, далі визначимо базову стратегію розвитку продукту (таблиця 5.16).

Таблиця 5.16 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	1та 3	Диференційованого маркетингу	Масштабування та максимізація	Оптимальних витрат

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиці 5.17, 5.18).

Таблиця 5.17 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
Ні	Так	Ні	Виклику лідера

Таблиця 5.18 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Універсальність Простота у використанні Якість результатів	Оптимальних витрат	Універсальність Простота у використанні Якість та гарантії Безкоштовне використання при MVP	Система, яка краще всіх показує передбачення Система з простим інтерфейсом

5.6 Розроблення маркетингової програми стартап-проекту

Після проведеного комплексного аналізу, можемо повноцінно описати ключові переваги концепції потенційного товару (таблиця 5.19) та побудувати концепцію маркетингових комунікацій (таблиця 5.20).

Таблиця 5.19 – Ключові переваги концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Якісне передбачення	Якісні результати прогнозування	Постійне покращення та перенавчання моделей, які зможуть аналізувати більше прикладних областей
2	Універсальність	Система не залежить від структури системи	Такого виду систему може використовувати будь-який користувач
3	Простий інтерфейс	Система дуже проста у використанні	Система із інтуїтивно зрозумілим інтерфейсом, який вимагає всього лише вести початкові дані

Таблиця 5.20 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламно о повідомлення	Концепція рекламного звернення
1	Пошук спеціалізованих систем	b2b продажі Зв'язок через теплі контакти Таргетована реклама у соціальних мережах Публікація в спеціалізованих виданнях, журналах	Точність Якість Універсальність	Поєднати повідомлення про те, що це якісна система, яка є незалежною	Таргетована реклама на цільову аудиторію
2	Пошук доступного та дешевого продукту	Рекламні банери в Інтернеті, форуми, реклами від інфлюенсерів	Простота Безкоштовне використання MVP	Вселити довіру у бренд та продукт	Реклама у лідерів думок Вивіски в публічних місцях Таргетована реклама на цільову аудиторію

Висновки до розділу 5

Даний розділ був присвячений дослідженню стартап-проекту. В якості такого була представлена система для оцінки часу обробки запиту в розподілених базах даних.

У рамках розділу було досліджено розробку стратегій виходу на ринок та маркетинг-стратегії для цього. Зокрема, даний ринок являється сприятливим з невеликою кількістю представлених компаній конкурентів. Оскільки вони дають лише частину функцій, а запропонована система є універсальною та доступною, то у стартап-проекту є всі шанси стати монополістами на ринку.

Також були опрацьовані сильні та слабкі сторони проекту, SWOT аналіз, аналіз конкурентів та цільової аудиторії. На основі всіх досліджень був сформований концепт маркетингової стратегії для обраних цільових аудиторій.

ВИСНОВКИ

У результаті виконання магістерської дисертації було створено модель для оцінки часу, витраченого на запит у розподіленій базі даних з динамічною структурою. Дана модель враховує навантаженність системи користувачами, а також саму структуру моделі.

Було створено програмний комплекс на мові програмування Python, за допомогою якого можна вказати параметри та створити модель структури розподіленої системи, візуалізувати її за допомогою графів. Даний програмний комплекс також будує оцінку часу обробки інформаційного запиту, а також будує графік, що відображає залежність часу від кількості користувачів, що створюють навантаження на розподілену систему з динамічною структурою.

Після експериментів прийшли до висновку, що для оптимізації часу варто комплексно підходити до зміни структури розподілених баз даних, а також варто приділити увагу на розподіл робочих процесів таким чином, аби користувачі в один момент часу не навантажували систему, оскільки після певного користувача час обробки починає значно зростати.

Дану модель можна використовувати в прикладних цілях для прогнозування часових витрат на робочі процеси, а також для оптимізації ресурсів у розподілених базах даних з динамічною структурою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005. — 1328 с.: ил. — Парал. тит. Англ.

2. Codd, E. F (1990), The Relational Model for Database Management, Addison-Wesley, pp. 371–388, ISBN 978-0-201-14192-4

3. Ярцев В.П. Розподілені бази даних: навчальний посібник. - К. ДУТ 2018. - 97с.

4. DB-Engines Ranking. DB-Engines. URL: https://db-engines.com/en/ranking/relational+dbms (дата звернення: 05.12.2022).

5. Allocation Fragmentation and Replication In Distributed Databases: A Quick Start Guide - Learn | Hevo. Learn | Hevo. URL: https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/ (дата звернення: 05.12.2022).

6. Distributed DBMS - Quick Guide. Online Tutorials Library. URL: https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_quick_guide.htm (дата звернення: 05.12.2022).

7. Raouf A. E. A., Badr N. L., Tolba M. F. Dynamic Distributed Database over Cloud Environment. Communications in Computer and Information Science. Cham, 2014. С. 67–76. URL: https://www.researchgate.net/publication/289052459_Dynamic_Distributed_Database_over_Cloud_Environment (дата звернення: 06.12.2022).

8. Lindsay B. G. et al. Notes on Distributed Databases // IBM Research Report RJ2571. — July 1979

9. Williams R. et al. R*: An Overview of the Architecture // P. Scheuermann. Improving Database Usability and Responsiveness. — New York, N.Y.: Academy Press, 1982

10. Корнага Я. І. Моделі та методи організації та управління гетерогенними розподіленими базами даних з динамічною структурою на основі мережецентричного підходу : дис. докт. техн. наук : 05.13.06 / Корнага Ярослав Ігорович – Київ, 2020. – 328 с.

11. V. Mukhin, Y. Kornaga, V. Bondarenko, V. Zavgorodnii, O. Herasymenko and O. Sholokhov, "Mathematical Model for Heterogeneous Databases Parameters Estimation in Distributed Systems with Dynamic Structure," 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT), 2020, pp. 158-161, doi: 10.1109/ATIT50783.2020.9349331.

12. E. A. Mikrin, S. K. Somov, “The optimal online information backup in the data processing systems based on computer networks redundancy”, Probl. Upr., 2016, no. 5, 47–56.

13. NetworkX – NetworkX documentation. NetworkX – NetworkX documentation. URL: https://networkx.org/ (дата звернення: 05.12.2022).

14. networkx. PyPI. URL: https://pypi.org/project/networkx/ (дата звернення: 05.12.2022).

15. Matplotlib – Visualization with Python. Matplotlib – Visualization with Python. URL: https://matplotlib.org/ (дата звернення: 05.12.2022).

16. Python interface to Tcl/Tk – Python 3.11.0 documentation. 3.11.0 Documentation. URL: https://docs.python.org/3/library/tkinter.html (дата звернення: 05.12.2022).

17. Pillow. Pillow (PIL Fork) 9.3.0 documentation. URL: https://pillow.readthedocs.io/en/stable/ (дата звернення: 05.12.2022).

ДОДАТОК А. ЛІСТИНГИ ПРОГРАМ

Visual.py

```
from tkinter import *
from PIL import ImageTk, Image
from calculate import *

SN = [0] * 5
SP = [0] * 5

if __name__ == '__main__':
    window = Tk()
    window.title("Structure of DDB")
    window.geometry('1100x600+20+10')

    frame_sn = LabelFrame(window, text='SN', width=300, height=300, font='Arial 14')
    frame_sn.grid(row=0, column=0)

    frame_sp = LabelFrame(window, text='SP', width=300, height=300, font='Arial 14')
    frame_sp.grid(row=0, column=1)

    frame_user = LabelFrame(window, text='Користувачі', width=600, height=150, font='Arial 14',
)
    frame_user.grid(row=1, column=0, columnspan=2)

    frame_results = LabelFrame(window, text='Результати', width=500, height=600, font='Arial 14')
    frame_results.grid(row=0, column=2, rowspan=3)

    frame_manipulating = LabelFrame(window, text='Дії', width=600, height=150, font='Arial 14')
    frame_manipulating.grid(row=2, column=0, columnspan=2)

    label_scn = Label(text='scn', font='Arial 14')
    label_scn.grid()
    label_scn.place(x=20, y=30)
```

```
text_scn = Text(window, height=1, width=15, font='Arial 14')
text_scn.grid()
text_scn.place(x=120, y=30)
text_scn.insert(1.0, '3')

label_scn_max = Label(text='scn_max', font='Arial 14')
label_scn_max.grid()
label_scn_max.place(x=20, y=60)

text_scn_max = Text(window, height=1, width=15, font='Arial 14')
text_scn_max.grid()
text_scn_max.place(x=120, y=60)
text_scn_max.insert(1.0, '3')

label_sdbn = Label(text='sdbn', font='Arial 14')
label_sdbn.grid()
label_sdbn.place(x=20, y=90)

text_sdbn = Text(window, height=1, width=15, font='Arial 14')
text_sdbn.grid()
text_sdbn.place(x=120, y=90)
text_sdbn.insert(1.0, '2')

label_sdbn_max = Label(text='sdbn_max', font='Arial 14')
label_sdbn_max.grid()
label_sdbn_max.place(x=20, y=120)

text_sdbn_max = Text(window, height=1, width=15, font='Arial 14')
text_sdbn_max.grid()
text_sdbn_max.place(x=120, y=120)
text_sdbn_max.insert(1.0, '3')

label_sein = Label(text='sein', font='Arial 14')
label_sein.grid()
```

```
label_sein.place(x=20, y=150)

text_sein = Text(window, height=1, width=15, font='Arial 14')
text_sein.grid()
text_sein.place(x=120, y=150)
text_sein.insert(1.0, '5')

label_tc = Label(text='t_c', font='Arial 14')
label_tc.grid()
label_tc.place(x=320, y=30)

text_tc = Text(window, height=1, width=15, font='Arial 14')
text_tc.grid()
text_tc.place(x=420, y=30)
text_tc.insert(1.0, '5')

label_tdb = Label(text='t_db', font='Arial 14')
label_tdb.grid()
label_tdb.place(x=320, y=60)

text_tdb = Text(window, height=1, width=15, font='Arial 14')
text_tdb.grid()
text_tdb.place(x=420, y=60)
text_tdb.insert(1.0, '5')

label_v = Label(text='v', font='Arial 14')
label_v.grid()
label_v.place(x=320, y=90)

text_v = Text(window, height=1, width=15, font='Arial 14')
text_v.grid()
text_v.place(x=420, y=90)
text_v.insert(1.0, '4')

label_min_users = Label(text='min', font='Arial 14')
```

```
label_min_users.grid()
label_min_users.place(x=30, y=330)

text_min_users = Text(window, height=1, width=41, font='Arial 14')
text_min_users.grid()
text_min_users.place(x=120, y=330)
text_min_users.insert(1.0, '1')

label_max_users = Label(text='max', font='Arial 14')
label_max_users.grid()
label_max_users.place(x=30, y=360)

text_max_users = Text(window, height=1, width=41, font='Arial 14')
text_max_users.grid()
text_max_users.place(x=120, y=360)
text_max_users.insert(1.0, '10')

label_curr_users = Label(text='current', font='Arial 14')
label_curr_users.grid()
label_curr_users.place(x=30, y=390)

text_curr_users = Text(window, height=1, width=41, font='Arial 14')
text_curr_users.grid()
text_curr_users.place(x=120, y=390)
text_curr_users.insert(1.0, '5')

result = Text(width=59, height=35)
result.grid()
result.place(x=610, y=30)


def work_create_graph():
    global SN, SP, u_max, u_min, u_curr
```

```
    scn = int(text_scn.get(1.0, END))
    scn_max = int(text_scn_max.get(1.0, END))
    sdbn = int(text_sdbn.get(1.0, END))
    sdbn_max = int(text_sdbn_max.get(1.0, END))
    sein = int(text_sein.get(1.0, END))
    tc = float(text_tc.get(1.0, END))
    tdb = float(text_tdb.get(1.0, END))
    v = float(text_v.get(1.0, END))
    u_min = int(text_min_users.get(1.0, END))
    u_max = int(text_max_users.get(1.0, END))
    u_curr = int(text_curr_users.get(1.0, END))


    if scn_max * scn < sdbn_max * sdbn:
        if u_max > scn_max * scn:
            result.insert(1.0,
                          'Помилка! Максимальна кількість користувачів більша, ніж наявних ресурсів у керуючих вузлах! Буде застосовано максимальну кількість користувачів у розмірі: scn_max * scn = ' + str(
                              scn_max * scn) + '\n\n')
            u_max = scn_max * scn
    else:
        if u_max > sdbn_max * sdbn:
            result.insert(1.0,
                          'Помилка! Максимальна кількість користувачів більша, ніж наявних ресурсів у вузлах з програмним забезпеченням для розподілених баз даних! Буде застосовано максимальну кількість користувачів у розмірі: sdbn_max * sdbn =' + str(
                              sdbn_max * sdbn) + '\n\n')
            u_max = sdbn_max * sdbn

    if scn_max * scn < sdbn_max * sdbn:
        if sein > scn_max * scn:
            result.insert(1.0,
```

```
                'Помилка! Максимальна кількість проміжних вузлів більша, ніж наявних ресурсів у керуючих вузлах! Буде застосовано проміжну кількість вузлів у розмірі: scn_max * scn = ' + str(
                    scn_max * scn) + '\n\n')
            sein = scn_max * scn
        else:
            if sein > sdbn_max * sdbn:
                result.insert(1.0,
                'Помилка! Максимальна кількість проміжних вузлів більша, ніж наявних ресурсів у вузлах з програмним забезпеченням для розподілених баз даних! Буде застосовано проміжну кількість вузлів у розмірі: sdbn_max * sdbn =' + str(
                    sdbn_max * sdbn) + '\n\n')
                sein = sdbn_max * sdbn
        if scn_max * scn < sdbn_max * sdbn:
            if u_curr > scn_max * scn:
                result.insert(1.0,
                'Помилка! Максимальна кількість поточних користувачів більша, ніж наявних ресурсів у керуючих вузлах! Буде застосовано поточну кількість користувачів у розмірі: scn_max * scn = ' + str(
                    scn_max * scn) + '\n\n')
                u_curr = scn_max * scn
        else:
            if u_curr > sdbn_max * sdbn:
                result.insert(1.0,
                'Помилка! Максимальна кількість поточних користувачів більша, ніж наявних ресурсів у вузлах з програмним забезпеченням для розподілених баз даних! Буде застосовано поточну кількість користувачів у розмірі: sdbn_max * sdbn =' + str(
                    sdbn_max * sdbn) + '\n\n')
                u_curr = sdbn_max * sdbn
        SN = [scn, sdbn, sein, scn_max, sdbn_max]
        SP = [tc, tdb, v, u_curr]
        G = create_graph(SN, SP)


    def build_graph():
```

```
    # Toplevel object which will
    # be treated as a new window
    newWindow = Toplevel(window)
    # sets the title of the
    # Toplevel widget
    newWindow.title("Graph visualization")

    # sets the geometry of toplevel
    newWindow.geometry("600x600+40+40")

    # A Label widget to show in toplevel
    G = create_graph(SN, SP)
    plot_graph(G)
    path = 'graph.png'
    global photo
    image = Image.open(path)
    image.thumbnail((600, 600))
    photo = ImageTk.PhotoImage(image)
    w1 = Label(newWindow, image=photo)
    w1.grid()
    w1.place(x=0, y=0)


def time():
    # result.delete(1.0, END)
    scn = SN[0]
    sdbn = SN[1]
    sein = SN[2]
    scn_max = SN[3]
    sdbn_max = SN[4]

    t_c = SP[0]
    t_db = SP[1]
    v = SP[2]
    u = SP[3]
```

```
        exp_time = expected_time(t_c, t_db, scn, sdbn, u, scn_max, sdbn_max, v)
        exp_time = str('%2.4f' % exp_time)
        res_to_print = str("Час обробки запиту для " + str(u) + " користувачів складає: " +
exp_time + "\n\n")
        result.insert(1.0, res_to_print)


    def build_graphic():
        scn = SN[0]
        sdbn = SN[1]
        sein = SN[2]
        scn_max = SN[3]
        sdbn_max = SN[4]

        t_c = SP[0]
        t_db = SP[1]
        v = SP[2]
        u = SP[3]
        users = [i for i in range(u_min, u_max + 1)]
        t_list = []
        for i in users:
            t = expected_time(t_c, t_db, scn, sdbn, i, scn_max, sdbn_max, v)
            t_list.append(t)
        plt.close()
        plt.plot(users, t_list)

        plt.xlabel('Users', fontsize=20)
        plt.ylabel('Time', fontsize=20)
        plt.savefig('graphics.png')
        newWindow2 = Toplevel(window)
        newWindow2.title("Plot")
        newWindow2.geometry("600x600+40+40")

        path = 'graphics.png'
```

```
        global photo
        image = Image.open(path)
        image.thumbnail((600, 600))
        photo = ImageTk.PhotoImage(image)
        w1 = Label(newWindow2, image=photo)
        w1.grid()
        w1.place(x=0, y=0)


    button_create_graph = Button(window, text='Побудувати структуру', width=29, height=2, bg='#00ffbf',
                                 fg='#000000', font='arial 12', command=work_create_graph)
    button_create_graph.grid()
    button_create_graph.place(x=20, y=480)

    button_show_graph = Button(window, text='Показати граф', width=29, height=2, bg='#00ffbf',
                               fg='#000000', font='arial 12', command=build_graph)
    button_show_graph.grid()
    button_show_graph.place(x=20, y=540)

    button_time = Button(window, text='Спрогнозувати час', width=29, height=2, bg='#00ffbf',
                         fg='#000000', font='arial 12', command=time)
    button_time.grid()
    button_time.place(x=310, y=480)

    button_graphics = Button(window, text='Показати графіки змін', width=29, height=2, bg='#00ffbf',
                             fg='#000000', font='arial 12', command=build_graphic)
    button_graphics.grid()
    button_graphics.place(x=310, y=540)

    window.mainloop()
```

calculate.py

```
from generate import *
```

```
def combination(n, m):
    res = 1
    for i in m:
        res *= factorial(i)
    return factorial(n) / res


def prepear_for_combination(u, c, c_max):
    list_of_m = []
    for m in product(range(c_max + 1), repeat=c):
        if sum(m) == u:
            list_of_m.append(m)

    return list_of_m


def count(u, c, c_max):
    res = 0
    for i in prepear_for_combination(u, c, c_max):
        res += combination(u, i)
    return res


def probability(u, c, c_max):
    if u > 1:
        try:
            return count(u, c, c_max) / (c * count(u - 1, c, c_max))
        # return count_u(u,c, c_max) / (count_u(u - 1,c, c_max) * c)
        except ZeroDivisionError:
            return 0.
    elif u == 1 or u == 0:
        return 1.
    else:
```

```
        return None


def length_way(scn, sdbn, u, scn_max, sdbn_max):
    L = 4 + 2 * (scn - 1) * (1 - probability(u, scn, scn_max)) + 2 * (sdbn - 1) * (1 - probability(u,
sdbn, sdbn_max))
    return L


def expected_time(tc, tdb, scn, sdbn, u, scn_max, sdbn_max, v):
    t = tc + tc * (scn - 1) * (1 - probability(u, scn, scn_max)) + tdb + tdb * (sdbn - 1) * (1 -
probability(u, sdbn, sdbn_max)) + length_way(
        scn, sdbn, u, scn_max, sdbn_max) / v
    return t
# u = 8
```

Generate.py

```
import matplotlib.pyplot as plt
import networkx as nx
from combinations import *


def create_graph(SN, SP):
    G = nx.Graph()
    scn = SN[0]
    sdbn = SN[1]
    sein = SN[2]
    scn_max = SN[3]
    sdbn_max = SN[4]

    t_c = SP[0]
    t_db = SP[1]
    v = SP[2]
    u = SP[3]
```

```
    counter = 0
    for i in range(scn):
        G.add_node(counter, type='control', t_c=t_c, scn_max=scn_max)
        counter += 1
    for i in range(sdbn):
        G.add_node(counter, type='database', t_db=t_db, sdbn_max=sdbn_max)
        counter += 1
    for i in range(sein):
        G.add_node(counter, type='intermediate')
        counter += 1

    for j in [i for i in G.nodes if G.nodes[i]['type'] == 'intermediate']:
        for k in [i for i in G.nodes if G.nodes[i]['type'] == 'database']:
            if j != k:
                G.add_edge(j, k, weight=v)

    for j in [i for i in G.nodes if G.nodes[i]['type'] == 'intermediate']:
        for k in [i for i in G.nodes if G.nodes[i]['type'] == 'control']:
            if j != k:
                G.add_edge(j, k, weight=v)

    G.add_edge(scn - 1, scn, weight=v)
    G.add_edge(scn + sdbn - 1, scn + sdbn, weight=v)
    return G

def plot_graph(G):
    plt.close()
    pos = nx.spring_layout(G, seed=1)

    # setting size of edge according to weight
    elarge = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] > 0.5]
    esmall = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] <= 0.5]

    options = {"edgecolors": "tab:gray", "node_size": 800, "alpha": 0.9}
```

```
    # drawing nodes
    nx.draw_networkx_nodes(G, pos, nodelist=[i for i in G.nodes if G.nodes[i]['type'] == 'control'],
                           node_color="tab:red",
                           **options)
    nx.draw_networkx_nodes(G, pos, nodelist=[i for i in G.nodes if G.nodes[i]['type'] == 'intermediate'],
                           node_color="tab:blue", **options)
    nx.draw_networkx_nodes(G, pos, nodelist=[i for i in G.nodes if G.nodes[i]['type'] == 'database'],
                           node_color="tab:orange", **options)
    # drawing edges
    nx.draw_networkx_edges(G, pos, edgelist=elarge, width=1)
    nx.draw_networkx_edges(
        G, pos, edgelist=esmall, width=1, alpha=0.5, edge_color="b", style="dashed"
    )

    # node labels
    nx.draw_networkx_labels(G, pos, verticalalignment='center', font_size=20, font_family="sans-serif")
    # edge weight labels

    ax = plt.gca()
    ax.margins(0.08)

    plt.axis("off")
    plt.tight_layout()
    plt.savefig('graph.png')
```